

Министерство образования Кузбасса
Государственное профессиональное образовательное учреждение
«Сибирский политехнический техникум»

УЧЕБНАЯ ДИСЦИПЛИНА
«МДК 01.03. Разработка мобильных приложений»

специальность

09.02.07 Информационные системы и программирование

Курс лекций

Кемерово, 2021

Курс лекций разработан в соответствии с программой учебной дисциплины «МДК 01.03. Разработка мобильных приложений» для специальностей среднего профессионального образования 09.02.07 Информационные системы и программирование

Разработчик:

Щербакова К.А. преподаватель специальных дисциплин государственного профессионального образовательного учреждения «Сибирский политехнический техникум»

СОДЕРЖАНИЕ

ЛЕКЦИЯ №1. ОСНОВНЫЕ ПЛАТФОРМЫ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ, СРАВНИТЕЛЬНАЯ ХАРАКТЕРИСТИКА.	5
ЛЕКЦИЯ №2-№3. ВВЕДЕНИЕ, ИСТОРИЯ РАЗВИТИЯ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ.	13
ЛЕКЦИЯ №4-№5. ОБЗОР СОВРЕМЕННЫХ МОБИЛЬНЫХ УСТРОЙСТВ.	23
ЛЕКЦИЯ №6. ТЕХНОЛОГИИ РАЗРАБОТКИ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ НА ПЛАТФОРМЕ ANDROID.	51
ЛЕКЦИЯ №7. ТЕХНОЛОГИИ РАЗРАБОТКИ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ НА ПЛАТФОРМЕ IPHONE.	54
ЛЕКЦИЯ №8. ТЕХНОЛОГИИ РАЗРАБОТКИ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ НА ПЛАТФОРМЕ WINDOWS PHONE.	64
ЛЕКЦИЯ №9. КРОССПЛАТФОРМЕННЫЕ ПРИЛОЖЕНИЯ, ИХ ОБЛАСТИ ПРИМЕНЕНИЯ.	68
ЛЕКЦИЯ №10-№11. НАТИВНЫЕ ПРИЛОЖЕНИЯ, ИХ ОБЛАСТИ ПРИМЕНЕНИЯ.	73
ЛЕКЦИЯ №12. ГИБРИДНЫЕ ПРИЛОЖЕНИЯ, ИХ ОБЛАСТИ ПРИМЕНЕНИЯ.	75
ЛЕКЦИЯ №13-№14. ОСНОВНЫЕ ЯЗЫКИ ДЛЯ РАЗРАБОТКИ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ.	80
ЛЕКЦИЯ №15-№16. ИНСТРУМЕНТАРИЙ СРЕДЫ РАЗРАБОТКИ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ.	89
ЛЕКЦИЯ №17. СТРУКТУРА ТИПИЧНОГО МОБИЛЬНОГО ПРИЛОЖЕНИЯ.	97
ЛЕКЦИЯ №18-№19. ЭЛЕМЕНТЫ УПРАВЛЕНИЯ И КОНТЕЙНЕРЫ.	106
ЛЕКЦИЯ №20-№21. РАБОТА СО СПИСКАМИ.	111
ЛЕКЦИЯ №22-№23. ОБРАБОТКА СОБЫТИЙ.	121
ЛЕКЦИЯ №24. СПОСОБЫ ХРАНЕНИЯ ДАННЫХ.	128
ЛЕКЦИЯ №25-№26. СТРУКТУРА ПРОЕКТА ANDROID. КАТАЛОГИ РЕСУРСОВ. ФАЙЛ R.JAVA.	133
ЛЕКЦИЯ №27. ГРАФИЧЕСКИЙ ИНТЕРФЕЙС ПОЛЬЗОВАТЕЛЯ В ANDROID-ПРИЛОЖЕНИЯХ.	162
ЛЕКЦИЯ №28-№29. АРХИТЕКТУРА ПЛАТФОРМЫ ANDROID.	167
ЛЕКЦИЯ №30. XML-РАЗМЕТКА ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ.	173
ЛЕКЦИЯ №31-№32. БАЗОВЫЕ ЭЛЕМЕНТЫ УПРАВЛЕНИЯ.	200
ЛЕКЦИЯ №33-№34. РЕСУРСЫ В ANDROID-ПРИЛОЖЕНИЯХ.	236
ЛЕКЦИЯ №35-№36. АКТИВНОСТИ И ИНТЕНТЫ.	255
ЛЕКЦИЯ №37-№38. ОБРАБОТКА ПОЛЬЗОВАТЕЛЬСКОГО ВВОДА. КАСАНИЯ, ВВОД ТЕКСТА.	261
ЛЕКЦИЯ №39-№40. ТИПЫ КОМПОНОВОК ГРАФИЧЕСКОГО ИНТЕРФЕЙСА.	266
ЛЕКЦИЯ №41. ДИАЛОГОВЫЕ ОКНА.	288
ЛЕКЦИЯ №42-№43. МНОГОПОТОЧНЫЕ ПРИЛОЖЕНИЯ В ANDROID.	303
ЛЕКЦИЯ №44-№45. ПРОЦЕССЫ В ANDROID. ОБЪЕКТЫ ACTIVITY.	315
ЛЕКЦИЯ №46-№47. ИСПОЛЬЗОВАНИЕ ОБЪЕКТОВ INTENT.	323
ЛЕКЦИЯ №48-№49. ИСПОЛЬЗОВАНИЕ РЕСУРСОВ. ССЫЛКИ НА РЕСУРСЫ.	338
ЛЕКЦИЯ №50-№51. ФАЙЛОВАЯ СИСТЕМА ANDROID. ЧТЕНИЕ И ЗАПИСЬ ФАЙЛОВ.	343

ЛЕКЦИЯ №52-№53. АДАПТЕРЫ ДАННЫХ.	360
ЛЕКЦИЯ №54-№55. ПОЛЬЗОВАТЕЛЬСКИЕ НАСТРОЙКИ. ИСПОЛЬЗОВАНИЕ SHAREDREFERENCES.	368
ЛЕКЦИЯ №56-№57. РАБОТА С ГРАФИКОЙ. DRAWABLE И CANVAS.	392
ЛЕКЦИЯ №58-№59. РАБОТА С АНИМАЦИЕЙ. TWEEN ANIMATION И FRAME ANIMATION.	415
ЛЕКЦИЯ №60-№61. СЛУЖБЫ В ANDROID. КОМПОНЕНТ SERVICE.	423
ЛЕКЦИЯ №62-№65. ДАТЧИКИ МОБИЛЬНЫХ УСТРОЙСТВ. УПРАВЛЕНИЕ ДАТЧИКАМИ В ПРИЛОЖЕНИИ.	438
ЛЕКЦИЯ №66-№67. ПРОГРАММНЫЙ ДОСТУП К ДИСПЛЕЮ УСТРОЙСТВА.	453
СПИСОК ЛИТЕРАТУРЫ	473

ЛЕКЦИЯ №1. ОСНОВНЫЕ ПЛАТФОРМЫ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ, СРАВНИТЕЛЬНАЯ ХАРАКТЕРИСТИКА.

Давно хотите создать мобильное приложение или сайт, но почему-то не решаетесь? На самом деле всё не так уж и сложно. В этой лекции мы узнаем о самых лучших и доступных решениях.



Страх перед программированием сковывает энтузиазм и заставляет нас отложить поиск ПО для создания приложений в долгий ящик. С другой стороны, войти на рынок мобильных приложений сегодня проще, чем когда-либо: для этого не требуются ни знания программирования, ни опыт.

Вне зависимости от целей — бизнес, продукт, блог, сервис и т.д. — есть ряд подходящих программ-создателей приложений. С небольшими затратами времени и готовностью учиться вы можете создавать и управлять своим мобильным сайтом или приложением, используя одну из платформ, перечисленных ниже.

[Appery.io](https://appery.io)



Appery – это облачный разработчик мобильных приложений, который можно использовать для создания приложений для Android или iOS; он также включает Apache Cordova (Phone Gap), Ionic и jQuery Mobile с доступом к их встроенным компонентам.

Поскольку Appery работает в облаке, нет необходимости что-либо загружать или устанавливать: приступить к работе довольно легко. В разработчике приложений есть визуальный редактор, использующий компоненты перетаскивания для создания пользовательского интерфейса. Appery автоматически генерирует код для любых компонентов, которые вы загружаете. Есть возможность подключиться к любому REST API и использовать его в своём приложении, а также мгновенно добавить облачную базу данных и сервер в своё приложение, если вам нужно хранить данные.

Также можно использовать каталог плагинов Appery или создать собственные. Если вы работаете с командой, то можете легко поделиться своим проектом с ними и сотрудничать в режиме реального времени за дополнительную плату.

Цена: От \$60/месяц до \$135/месяц для команды и пользовательских цен для корпоративных решений.

Mobile Roadie



Mobile Roadie – конструктор приложений, который позволяет любому создавать и управлять своим iOS или Android-приложением. Примечательно, что процесс разработки происходит в визуальном формате. Платформа поддерживает все типы носителей, автоматически импортируя RSS, Twitter или ключевые слова из Google News, а также автоматически обновляя fan wall для общения с пользователями в реальном времени.

Вы можете точно просмотреть, как будет выглядеть приложение с устройства пользователя. Mobile Roadie поможет пройти процесс рассмотрения в App Store и проверит качество и уместность контента.

Этот конструктор приложений даёт возможность отправлять push-уведомления (как контент с вашего сайта, так и с самой платформы). Платформа в целом является языково-независимой, поэтому можно передавать данные в различных форматах, включая XML, JSON, PHP, CSV и HTML. В самом начале вам предлагается несколько вариантов макетов, которые настраиваются по своему усмотрению.

Цена: от \$145/месяц.

TheAppBuilder



TheAppBuilder предлагает набор приложений для сотрудников, клиентов, событий и брошюр, используя 2 подхода: платформа, с которой вы собираетесь работать, если разрабатываете приложение в качестве интрасети для компании; также можно создать приложение с помощью онлайн-инструментария, либо самого TheAppBuilder, чтобы наполнять приложение контентом и устанавливать его структуру.

Используя AppLibrary, обеспечите своим пользователям доступ к окну с несколькими приложениями, которое можно настроить его в стиле своего бренда. Есть возможность защитить как общедоступные, так и частные приложения с помощью пользовательских имён и паролей и распределить их через магазин приложений, используя интеграцию Active Directory TheAppBuilder.

Обновление структуры и контента приложения происходит легко, даже после того, как вы опубликуете его, так как можно вносить неограниченное количество изменений и публиковать на нескольких мобильных платформах одним щелчком мыши. Платформа поддерживает iPhone, iPad и Android, а

обновления активируется в течение 60 секунд после внесения изменений. Синхронизация обновлений не отличается от других служб.

Цена: по запросу

Good Barber



Good Barber предоставляет платформу для создания приложений для iPhone и Android, а также оптимизированных веб-приложений. Вы можете контролировать каждую деталь приложения для любой из платформ, не написав ни одной строки кода. На старте предлагается несколько настраиваемых шаблонов оформления, множество интересных иконок и доступ к Google Fonts.

Эти прогрессивные веб-приложения теоретически могут заменить текущий сайт, поскольку они могут быть оптимизированы для настольных компьютеров, мобильных устройств и планшетов.

Вы получаете немедленную визуальную обратную связь каждый раз, когда настраиваете какие-либо параметры в своём приложении. Отправляйте уникальные, эффективные push-уведомления и превращайте своих читателей в соавторов, позволяя им отправлять статьи, фотографии и видео. Вы также можете организовать своё приложение и добавлять разделы, начиная от видео, фотографий, звуков, прямых трансляций и т.д.

Цена: от \$32/месяц для Android-приложений и \$96/месяц для iOS-приложений.

Appy Pie



Appy Pie — это облачный инструмент для создания мобильных приложений (по принципу DIY – do it yourself или «сделай сам»), который позволяет пользователям без навыков программирования создавать приложение практически для любой платформы и публиковать его. Не нужно ничего устанавливать или загружать – просто перетащите страницы, чтобы создать собственное мобильное приложение онлайн. После того как процесс будет завершен, вы получите гибридное приложение на базе HTML5, которое будет работать со всеми платформами, включая iOS, Android, Windows и даже Progressive app. Все изменения происходят в режиме реального времени, с возможностью отправки push-уведомлений, монетизации с рекламой, просмотра живой аналитики и отслеживания местоположения с помощью GPS. Вы также можете интегрировать каналы социальных сетей, блоги, веб-сайты, аудио, радио и многое другое.

При использовании этой платформы вам предлагаются разные темы: от ресторана до радиостанции. У них также есть инструмент планировщика встреч, который особенно полезен для таких компаний, как врачи, салоны или спа с контактными функциями, например, входящий вызов одним нажатием. Используя кодовую страницу, можно встраивать собственный код и встроенный фрейм.

Цена: от бесплатного плана с рекламой до \$50/месяц.

[AppMachine](#)



AppMachine — это простая в использовании платформа для создания и разработки профессиональных нативных приложений для iOS и Android. Используя drag-and-drop интерфейс, вы можете комбинировать различные строительные блоки, которые предлагают множество функций: информация, фотографии и видео. Кроме того, они позволяют связать приложение с Facebook, Twitter или интернет-магазинами. Создавайте свой уникальный дизайн, выбирая пути навигации, цвета, шрифты и значки, а также полностью контролируйте макет и следите за ходом работы с помощью Previewer.

Эта платформа даёт возможность сканировать веб-сайт для ключевого контента, который может переноситься в приложение для iOS или Android.

Проверяйте приложение при его создании и проверяйте его прогресс на своём компьютере, мобильном телефоне или планшете. После того, как приложение будет полностью протестировано и готово к работе, вы сможете публиковать и продвигать его, а также анализировать пользовательские данные. AppMachine позаботится обо всём, что нужно для того, чтобы ваше приложение оказалось в App Store и в Google Play.

Цена: от \$49/месяц за приложение до \$69/месяц за приложение.

[GameSalad](#)



GameSalad позволяет создавать и публиковать игры для различных платформ, включая iOS, Android, HTML5 и macOS. Интерфейс перетаскивания позволяет быстро начать работу, не требуя каких-либо знаний о программировании, при этом давая возможность обучаться компьютерной науке в процессе работы.

Создатель игрового приложения имеет редактор сцен и актёров, с помощью которого можно помещать актёров в сцену и создавать атрибуты, изображения, звуки и навигацию между актёрами. Активный форум сообщества – огромный плюс, с доступом к помощи и советам от независимых разработчиков игр.

GameSalad предлагает два плана: один предназначен для преподавателей, а другой для разработчиков. Эти оба конструктора приложений позволяют создавать игру без написания кода.

Цена: по запросу для разработчиков и преподавателей.

BiznessApps



BiznessApps – это платформа, которая помогает с лёгкостью создавать мобильные приложения для малого бизнеса. Она предоставляет множество функций, включая оформление заказа, корзину покупок, отзывы, сообщения,

динамический контент, сторонние интеграции, push-уведомления, всестороннюю аналитику и многое другое.

Простая в использовании система управления контентом позволяет создавать приложение за считанные минуты и настраивать всё при помощи собственного дизайна. Также есть опция предварительного просмотра в режиме реального времени, которая помогает проверить прогресс при разработке приложения.

Вы можете мгновенно обновлять своё приложение в интернете и изменять всё внутри, без необходимости обновлять приложение с Apple или Google. Помимо этого, BiznessApps предлагает простой способ создания рекламных материалов. Тут есть еженедельные вебинары о том, как создавать красивые мобильные приложения, используя платформу BiznessApps, а также доступ к сотням руководств и статей, которые помогут сделать процесс ещё проще.

Цена: от \$99/месяц до \$250/месяц.

AppMakr



AppMakr, платформа для разработки приложений по принципу DIY, которая позволяет любому создавать приложения для iOS, Android и HTML5 без каких-либо знаний по программированию. Она основана в 2009 году и недавно была приобретена компанией Infinite Monkeys, став после этого одной из крупнейших в мире платформ для публикации приложений.

Вы можете сделать столько приложений, сколько хотите, с неограниченными обновлениями и использовать ряд функций, включая push-уведомления, фотогалереи высокого разрешения, живые обновления, потоковую музыку и видео, чаты, интеграцию с Google Maps, календари общих событий, встроенный в приложение шоппинг и многое другое.

Вы отслеживаете все приложения через панель мониторинга, легко настраиваете внешний вид и функциональность своего приложения,

добавляете контент с вкладками, предпросматриваете приложение в режиме реального времени и публикуете его в один клик.

Цена: от бесплатного конструктора мобильных сайтов до \$99/месяц.

ShoutEm



ShoutEm предлагает конструктор приложений с полным управлением контентом, мощными инструментами взаимодействия с пользователем и вариантами монетизации в сочетании с лёгким процессом публикации. Всё это гарантирует, что сценарий создания приложения будет хорошим.

Разработчик интерфейса предоставляет множество параметров настройки, и каждое приложение может быть тесно интегрировано с существующими источниками контента, такими как WordPress, Foursquare, Facebook, Twitter, SoundCloud и т. д. Множественные варианты монетизации означают, что вы можете быстро создать поток поступления дохода из своего приложения, предоставить опыт электронной коммерции, подписки внутри приложения, сделки и купоны, с поддержкой всех основных рекламных сетей.

Вы можете публиковать свои приложения под учетной записью разработчика как в iTunes, так и в Google Play, а также обновлять приложение или контент в режиме реального времени. Ещё ShoutEm предлагает регулярные автоматические обновления, чтобы приложение всегда опережало новые обновления для iOS и Android.

Цена: от \$59/месяц для Android до \$99/месяц для iOS и Android и \$179/месяц за дополнительные надстройки.

ЛЕКЦИЯ №2-№3. ВВЕДЕНИЕ, ИСТОРИЯ РАЗВИТИЯ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ.

Android - операционная система для мобильных устройств: смартфонов, планшетных компьютеров, КПК. В настоящее время именно Android является самой широко используемой операционной системой для мобильных устройств. Подтверждение этого факта можно найти в таблице, составленной по данным аналитической компании Gartner.

Таблица 1.1. Мировые продажи смартфонов конечным пользователям, распределение по ОС

Операционная система	Продано (тыс.ед.) III кв. 2013	Доля рынка (%) III кв. 2013	Продано (тыс.ед.) III кв. 2012	Доля рынка (%) III кв. 2012
Android	205022,7	81,9	124552,3	72,6
iOS	30330,0	12,1	24620,3	14,3
Microsoft	8912,3	3,6	3993,6	2,3
BlackBerry	4400,7	1,8	8946,8	5,2
Bada	633,3	0,3	4454,7	2,6
Symbian	457,5	0,2	4401,3	2,6
другие	475,2	0,2	683,7	0,4
Общее колво:	250231,7	100,0	171652,7	100,0

Источник: Gartner (ноябрь 2013)

Внимательное изучение таблицы позволяет увидеть подавляющую популярность смартфонов под управлением ОС Android в мире, доля таких устройств не первый год превышает половину от общего числа купленных смартфонов. Кроме всего прочего, эта популярность продолжает расти. Очевидно, что армия пользователей смартфонов под управлением Android будет искать дополнительные приложения для своих устройств, в связи с этим умение разрабатывать эти самые приложения может принести много пользы своему владельцу. Например, можно разрабатывать для себя полезные, интересные, занимательные (нужное подчеркнуть) приложения, а можно, разведав обстановку и осмотревшись, сделать разработку мобильных приложений своей профессиональной деятельностью, основной или дополнительной.

Данная лекция является первой для всего курса, призвана ввести читателя в курс дела. В первую очередь в ней рассматриваются вопросы становления и развития ОС Android. Для успешного программирования под Android необходимо понимать внутреннюю организацию и архитектуру этой платформы, а также полезно знать, какие инструменты и среды разработки можно использовать. Этим вопросам посвящена основная часть лекции. Кроме того, в лекции рассматриваются особенности запуска и отладки мобильных приложений.

Немного истории

Рассмотрим, как все начиналось. В 2003 году в Пало Альто, штат Калифорния Энди Рубин с единомышленниками (Рич Майнер, Ник Сирс и Крис Уайт) основали компанию Android Inc. Поначалу в компании занимались проектированием мобильных гаджетов, которые на основе геолокационных данных автоматически подстраивались под нужды пользователей.

В августе 2005 года Android Inc. стала дочерней компанией Google. Энди Рубин, Рич Майнер и Крис Уайт остались в Android Inc. и начали работать над операционной системой, базирующейся на ядре Linux. В Google задумали реализовать мощнейшую платформу, пригодную к использованию на тысячах различных моделей телефонов. В связи с этим был создан Open Handset Alliance (ОНА) - консорциум, состоящий из более 80 компаний, направляющий свои усилия на разработку открытых стандартов для мобильных устройств. В состав ОНА входят такие гиганты, как Google (организатор и идейный вдохновитель), HTC, Sony, Dell, Intel, Motorola, Qualcomm, Texas Instruments, Samsung Electronics, LG Electronics, T-Mobile, Sprint Corporation, NVIDIA и многие другие.

Первая версия Android была представлена 23 сентября 2008 года, версии было дано название Apple Pie (можно заметить созвучие с прямым конкурентом). Далее так повелось, что название каждой очередной версии представляет какой-либо десерт, при этом первые буквы наименований в порядке версий соответствуют буквам латинского алфавита по порядку. С развитием обновлений Android можно познакомиться в приведенной ниже таблице.

Версия, логотип, дата выход	Основные возможности
Android 1.0 Apple Pie	Первый стабильный релиз, основан на ядре Linux 2.6.25. Поддерживается: файловая система FAT32, стек интернет-протоколов TCP/IP; протоколы передачи данных: 802.11 b/g WiFi, Bluetooth 2.0 EDR, GPRS, EDGE, UMTS, HSDPA; фото и видеосъемка, однако недостаточно опций для настройки разрешения камеры, баланса белого и др.; сенсорные дисплеи и landscape режим отображения данных на экране, максимальная цветность дисплея - 16 бит (тип HVGA); виджеты и ярлыки на рабочем столе (Home Screen), сменные обои; регулярные телефонные функции, контроль вызова, конференц-связь, легкая интеграция с контактами; полноценный web-браузер на движке WebKit, HTML, XHTML; e-mail клиент, протоколы POP3, IMAP4, SMTP; медиа проигрыватель, позволяющий управлять, импортировать, проигрывать медиа контент в различных форматах. Базовые приложения: будильник; калькулятор; календарь; камера; контакты; сообщения (в том числе MMS); настройки; голосовой набор. Минимальные системные требования для запуска и работы: архитектура ARM, 128 MB RAM, 256 MB ROM.

Android 1.1 Banana Bread февраль 2009 (API level: 2)	Нововведения: Исправлены проблемы: с будильником; со спящим режимом; с вызовом дисплея набора номера; в IMAP ошибки запроса пароля и др. Изменения API. Добавлены подробности и отзывы к картам. Добавлена поддержка вложений из MMS. Локализации: Английская US (en_US), Немецкая (de).
Android 1.5 Cupcake апрель 2009 (API level: 3)	Нововведения: Поддержка экранной клавиатуры (); акселерометра; видеозапись и воспроизведение видео; приложение для работы с YouTube; стерео Bluetooth; функция копирования и вставки между приложениями (copy& paste). Локализации: добавились очень многие, в том числе и русская (ru_RU). Система: новое Linux ядро (версия 2.6.27); автоматическая проверка и восстановление файловой системы на SD card; новое приложение для просмотра СТК меню оператора (SIM Application Toolkit 1.0). Изменения в пользовательском интерфейсе (UI): изменено большинство UI-элементов, добавлены новые виджеты; определение режима (книжный или портретный) работы программы; анимированное переключение между окнами.
Android 1.6 Donut сентябрь 2009 (API level: 4)	Нововведения: Система: новое ядро Linux (версия 2.6.29); поддержка сотового стандарта CDMA; поддержка разрешений дисплеев: QVGA и WVGA; обновленный медиа-движок OpenCore 2; движок синтеза речи (многоязыковой); Gesture Builder поддержка возможности (для разработчиков) создавать, сохранять, загружать и распознавать жесты, прикреплять к определенным действиям. Пользовательские возможности: строка быстрого поиска (прямо с рабочего стола); история и закладки в браузере, контакты и поиск в интернете; возможность подключаться к видам VPN: L2TP/IPSEC pre-shared key based VPN, L2TP/IPSEC certificate based VPN, L2TP only VPN, PPTP only VPN; ускорение работы камеры; индикатор работы батареи позволяет увидеть сколько энергии потребляют работающие программы и сервисы. Обновленный Android Market.
Android 2.0, 2.0.1, 2.1 Eclair октябрь 2009 (API level: 5) (API level: 6) (API level: 7)	Нововведения в 2.0: поддержка работы нескольких почтовых аккаунтов одновременно, возможность использования совместных папок (входящие, исходящие) для всех аккаунтов; быстрый способ работы с контактами Quick Contact; поиск по всем сохраненным SMS и MMS сообщениям, удаление старых после заданного срока; возможности камеры: вспышка, цифровой зум, сценические режимы, баланс белого, цветовые эффекты, макрофокусировка; улучшенное расположение виртуальных клавиш клавиатуры, поддержка комбинированных нажатий клавиш (технология мультитач),

	усовершенствованная функция автодополнения; поддержка HTML5, версии Bluetooth 2.1, новых профилей OPP и PBAP. Нововведения 2.0.1: подрилиз версии 2.0, включающий в себя незначительные изменения в функционале и по большей части bugfix-ом версии 2.0. Нововведения 2.1: основным новшеством, представляющим интерес для конечного пользователя, стало добавление анимированных (живых) обоев, остальные изменения в Framework API, представляют интерес для разработчиков.
Android 2.2 Froyo май 2010 (API level: 8)	Нововведения: рост производительности примерно в 3-5 раз за счет использования Dalvik Virtual Machine Just-in-Time компилятора; возможности установки приложений на SD-карту, переноса приложений из внутренней памяти на карту и обратно; возможность использовать смартфон в качестве точки доступа к интернету, в качестве модема для других устройств; поддержка Adobe Flash; V8 javascript существенно повысил скорость работы штатного браузера.
Android 2.3, 2.3.3 Gingerbread декабрь 2010 (API level: 9) (API level: 10)	До весны 2013 года самая массовая версия на рынке. Нововведения: новое ядро Linux 2.6.35; поддержка открытых мультимедийных стандартов (VP8 и WebM), форматов AAC/AMR, звуковых эффектов и эквалайзера, фронтальной камеры (интеграция с VOIP(SIP)); обновленный GUI: уменьшение времени доступа к функциям, повышение общей энергоэффективности системы; улучшение стандартной клавиатуры системы: поддержка словарей, технологии мультитач, упрощенное выделение и копирование текста; поддержка технологии NFC; расширение возможностей работы с датчиками положения телефона.
Android 3.0-3.2 Honeycomb февраль 2011 (API level: 11) (API level: 12) (API level: 13)	Специальная версия для работы на планшетах (MID, tablets). Нововведения 3.0: новое ядро Linux 2.6.36; поддержка файловой системы ext4, файловой системы FUSE для MTP устройств; поддержка режима USB-хост для работы с клавиатурой, мышью и USB-хабами; поддержка MTP/PTP; виртуальная машина Dalvik: поддержка и оптимизация SMP, множество улучшений JIT, улучшенный сборщик мусора; совершенно новый интерфейс с полноценной оптимизацией под устройства с большими экранами; поддержка виртуальных рабочих столов, каждый из которых может иметь свой набор виджетов и ярлыков; улучшенные и переработанные базовые приложения: Browser, e-mail и др. Нововведения 3.1: поддержка работы кардридера; усовершенствован GUI: доработан менеджер задач, позволяющий переключаться

	между множеством различных приложений (в 3.0 только 5 программ одновременно); возможность менять размер виджетов, как по горизонтали, так и по вертикали. Нововведения 3.2: расширен спектр поддерживаемых планшетов; возможность автоматического масштабирования приложений для отображения на более крупных экранах.
Android 4.0, 4.0.3 Ice Cream Sandwich ноябрь 2011 (API level: 14) (API level: 15)	Нововведения: поддержка и смартфонов, и планшетов; поддержка новых процессорных архитектур, помимо ARM поддержка Intel x86 и MIPS; возможность разблокировки экрана: при помощи функции определения лица; жестами: перетащить замочек из центра экрана на иконку приложения и оно запустится; многозадачность: кнопка Recent Apps позволяет мгновенно переходить от одной задачи к другой с помощью списка в системной панели; новые элементы управления передачей данных через сеть: в приложении Настройки можно увидеть общее использование данных по каждому типу сети, объем данных, используемых каждым работающим приложением; доступность Android 4.0 для слепых и слабовидящих пользователей, браузер поддерживает экранного чтеца, который воспроизводит все видимое активное содержимое на экране; AndroidBeam - удобное средство обмена между двумя NFC-устройствами; Wi-Fi Direct и Bluetooth H DP, HFP: возможность прямого подключения к соответствующим устройствам.
Android 4.1-4.3 Jelly Bean июль 2012 (API level: 16) (API level: 17) (API level: 18)	Нововведения 4.1: увеличена скорость прорисовки интерфейса, улучшен поиск, добавлено несколько полезных сервисов; улучшена работа со словарями, возможно использовать голосовой ввод без подключения к интернету; специальные возможности: возможность управления смартфоном с помощью жестов и голосовых подсказок, подключения устройств ввода, поддерживающих шрифт Брайля; существенно доработана функция передачи данных Beam; переработан поиск (вместо ссылок ответ на запрос); голосовой поиск; Google Now: нужная информация в нужное время. Подробности: здесь Нововведения 4.2: реализована поддержка нескольких пользователей (планшеты); поддержка wireless display: возможность трансляции видео и изображений на внешний экран; возможность отображения полезной информации в режиме сна, при подключении к док-станции или на зарядке; улучшена панель уведомлений. Подробности: здесь Нововведения 4.3: ускорение работы системы; более точный набор на клавиатуре; скрытая возможность управления процессами программ (необходима активация); поддержка OpenGL/ES 3.0 (не на всех устройствах).

Android 4.4 Kit Kat октябрь 2013 (API level: 19)	Нововведения: многозадачность, оптимизация распределения ресурсов между приложениями; определитель номера работает не только с адресной книгой (например, Google maps); серьезная интеграция приложения Hangouts (отправка SMS, MMS, голосовые и видеовызовы); в состав вошел Quickoffice, интегрированный с Google Drive; поддержка принтеров, подключение через приложения поддерживающие печать (например, Google Cloud Print, HP ePrint); поддержка стандарта Wi-Fi Miracast, позволяющий вещать изображение на телевизор; возможность захвата экрана для записи видео.

Таблица 1.2. История обновлений ОС Android

2. Устройство платформы Android

Платформа Android объединяет операционную систему, построенную на основе ядра ОС Linux, промежуточное программное обеспечение и встроенные мобильные приложения. Разработка и развитие мобильной платформы Android выполняется в рамках проекта AOSP (Android Open Source Project) под управлением ОНА (Open Handset Alliance), руководит всем процессом поисковый гигант Google. Android поддерживает фоновое выполнение задач; предоставляет богатую библиотеку элементов пользовательского интерфейса; поддерживает 2D и 3D графику, используя OpenGL стандарт; поддерживает доступ к файловой системе и встроенной базе данных SQLite. С точки зрения архитектуры, система Android представляет собой полный программный стек, в котором можно выделить следующие уровни:

- Базовый уровень (Linux Kernel) - уровень абстракции между аппаратным уровнем и программным стеком;
- Набор библиотек и среда исполнения (Libraries & Android Runtime) обеспечивает важнейший базовый функционал для приложений, содержит виртуальную машину Dalvik и базовые библиотеки Java необходимые для запуска Android приложений;
- Уровень каркаса приложений (ApplicationFramework) обеспечивает разработчикам доступ к API, предоставляемым компонентами системы уровня библиотек;
- Уровень приложений (Applications) - набор предустановленных базовых приложений.

Наглядное изображение архитектуры на рисунке 1.

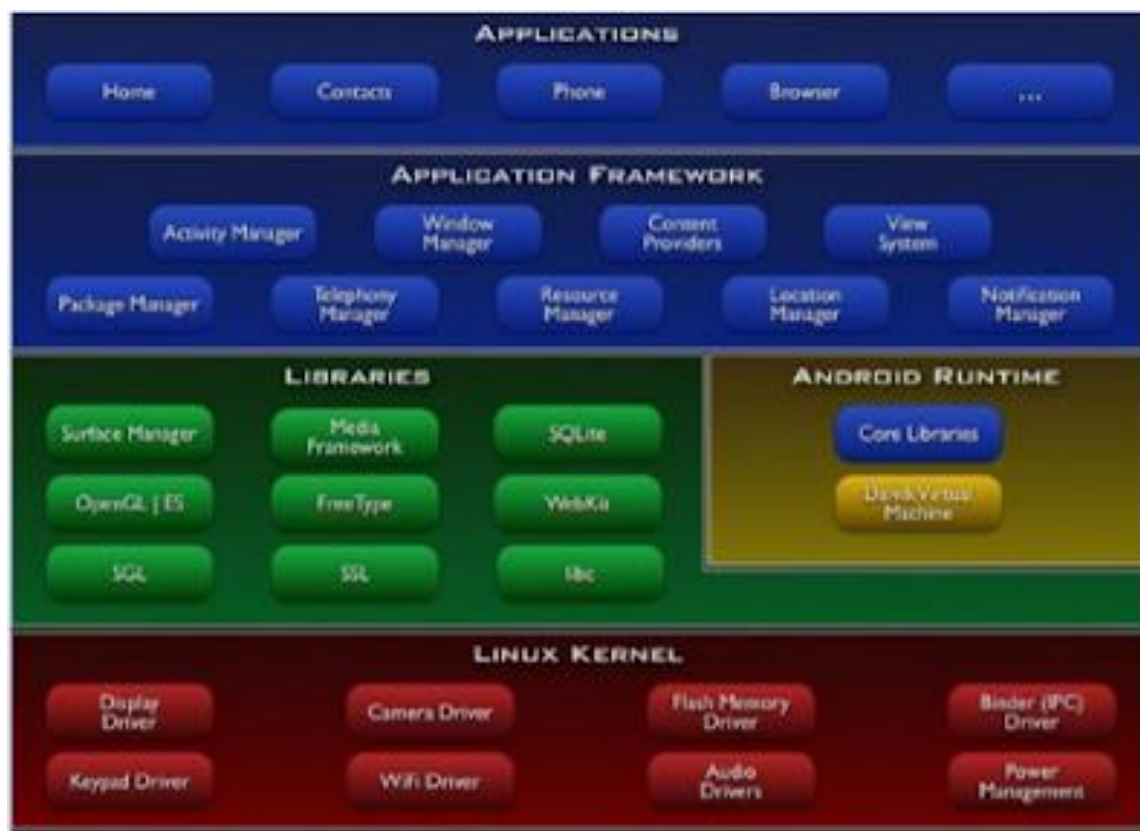


Рисунок 1. Архитектура Android

Рассмотрим компоненты платформы более подробно.

В основании компонентной иерархии лежит ядро ОС Linux 2.6(несколько урезанное), служит промежуточным уровнем между аппаратным и программным обеспечением, обеспечивает функционирование системы, предоставляет системные службы ядра: управление памятью, энергосистемой и процессами, обеспечение безопасности, работа с сетью и драйверами.

Уровнем выше располагается набор библиотек и среда исполнения. Библиотеки реализуют следующие функции:

- предоставляют реализованные алгоритмы для вышележащих уровней;
- обеспечивает поддержку файловых форматов;
- осуществляет кодирование и декодирование информации (например, мультимедийные кодеки);
- выполняет отрисовку графики и т.д.

Библиотеки реализованы на C/C++ и скомпилированы под конкретное аппаратное обеспечение устройства, вместе с которым они и поставляются производителем в предустановленном виде. Рассмотрим некоторые библиотеки:

Surface Manager – композитный менеджер окон. Поступающие команды отрисовки собираются в закадровый буфер, где они накапливаются, составляя некую композицию, а потом выводятся на экран. Это позволяет системе создавать интересные бесшовные эффекты, прозрачность окон и плавные переходы.

Media Framework - библиотеки, реализованные на базе PacketVideo OpenCORE. Используются для записи и воспроизведения аудио и видео контента, а также для вывода статических изображений. Поддерживаются форматы: MPEG4, H.264, MP3, AAC, AMR, JPG и PNG.

SQLite - легковесная и производительная реляционная СУБД, используется в Android в качестве основного движка для работы с базами данных.

3D библиотеки - используются для высокооптимизированной отрисовки 3D-графики, при возможности используют аппаратное ускорение. Библиотеки реализованы на основе API OpenGL|ES. OpenGL|ES (OpenGL for Embedded Systems) - подмножество графического программного интерфейса OpenGL, адаптированное для работы на встраиваемых системах.

FreeType - библиотека для работы с битовыми картами, для растеризации шрифтов и осуществления операций над ними.

LibWebCore- библиотеки браузерного движка WebKit, используемого также в известных браузерах Google Chrome и Apple Safari.

SGL (Skia Graphics Engine) - открытый движок для работы с 2D-графикой. Графическая библиотека является продуктом Google и часто используется в других программах. SSL - библиотеки для поддержки одноименного криптографического протокола.

Libc - стандартная библиотека языка C, а именно ее BSD реализация, настроенная для работы на устройствах на базе Linux.

Среда исполнения включает в себя библиотеки ядра, обеспечивающие большую часть низкоуровневой функциональности, доступной библиотекам ядра языка Java, и виртуальную машину Dalvik, позволяющую запускать приложения. Каждое приложение запускается в своем экземпляре виртуальной машины, тем самым обеспечивается изоляция работающих приложений от ОС и друг от друга. Для исполнения на виртуальной машине Dalvik Java-классы компилируются в исполняемые файлы с расширением .dex с помощью инструмента dx, входящего в состав Android SDK. DEX (Dalvik EXecutable) - формат исполняемых файлов для виртуальной машины Dalvik, оптимизированный для использования минимального объема памяти. При

использовании IDE Eclipse и плагина ADT(Android Development Tools) компиляция классов Java в формат .dex происходит автоматически.

Архитектура Android Runtime такова, что работа программ осуществляется строго в рамках окружения виртуальной машины, что позволяет защитить ядро ОС от возможного вреда со стороны других ее составляющих. Поэтому код с ошибками или вредоносное ПО не смогут испортить Android и устройство на его базе, когда сработают.

На еще более высоком уровне располагается каркас приложений (Application Framework), архитектура которого позволяет любому приложению использовать уже реализованные возможности других приложений, к которым разрешен доступ. В состав каркаса входят следующие компоненты:

- богатый и расширяемый набор представлений (Views), который может быть использован для создания визуальных компонентов приложений, например, списков, текстовых полей, таблиц, кнопок или даже встроенного web-браузера;
- контент-провайдеры (Content Providers), управляющие данными, которые одни приложения открывают для других, чтобы те могли их использовать для своей работы;
- менеджер ресурсов (Resource Manager), обеспечивающий доступ к ресурсам без функциональности (не несущим кода), например, к строковым данным, графике, файлам и другим;
- менеджер оповещений (NotificationManager), позволяющий приложениям отображать собственные уведомления для пользователя в строке состояния;
- менеджер действий (Activity Manager), управляющий жизненными циклами приложений, сохраняющий историю работы с действиями, предоставляющий систему навигации по действиям;
- менеджер местоположения (LocationManager), позволяющий приложениям периодически получать обновленные данные о текущем географическом положении устройства.

Application Framework предоставляет в распоряжение приложений в ОС Android вспомогательный функционал, благодаря чему реализуется принцип многократного использования компонентов приложений и ОС. Естественно, в рамках политики безопасности.

И, наконец, самый высокий, самый близкий к пользователю уровень приложений. Именно на этом уровне пользователь взаимодействует со своим устройством, управляемым ОС Android. Здесь представлен набор базовых приложений, который предустановлен на ОС Android. Например, браузер,

почтовый клиент, программа для отправки SMS, карты, календарь, менеджер контактов и др. Список интегрированных приложений может меняться в зависимости от модели устройства и версии Android. К этому уровню также относятся все пользовательские приложения.

Разработчик обычно взаимодействует с двумя верхними уровнями архитектуры Android для создания новых приложений. Библиотеки, система исполнения и ядро Linux скрыты за каркасом приложений.

Повторное использование компонентов других приложений приводит к идее задач в Android. Приложение может использовать компоненты другого Android приложения для решения задачи, например, если разрабатываемое приложение предполагает использование фотографий, оно может вызвать приложение, управляющее фотографиями и зарегистрированное в системе Android, выбрать с его помощью фотографию и работать с ней.

Для пополнения коллекции приложений своего мобильного устройства пользователь может воспользоваться приложением Google Play, которое позволяет покупать и устанавливать приложения с сервиса Google Play. Разработчики, в свою очередь, могут выкладывать свои приложения в этот сервис, Google Play отслеживает появление обновлений приложения, сообщает пользователям этого приложения об обновлении и предлагает установить его. Также Google Play предоставляет разработчикам доступ к услугам и библиотекам, например, доступ к использованию и отображению Google Maps.

Для установки приложения на устройствах с ОС Android создается файл с расширением *.apk (Android package), который содержит исполняемые файлы, а также вспомогательные компоненты, например, файлы с данными и файлы ресурсов. После установки на устройство каждое приложение "живет" в своем собственном изолированном экземпляре виртуальной машины Dalvik.

ЛЕКЦИЯ №4-№5. ОБЗОР СОВРЕМЕННЫХ МОБИЛЬНЫХ УСТРОЙСТВ.

Что представляет собой современный телефон? Нет, это не «чемоданчик» с «крутилкой» и не «кирпич» с большими кнопками, которые обладают функцией звонка. Современный мобильный телефон – это устройство, в котором сочетаются не только габариты и дизайн, но и технические характеристики, и операционная система.

Операционная система iOS

ios

iOS – одна из самых передовых операционных систем на рынке смартфонов и планшетов. Устанавливается на iPhone, iPod и iPad. Имеет множество функций, многозадачная, не требовательна к ресурсам. Но данная система платная, устройства компании Apple стоят не дёшево.

Также данная система имеет закрытый исходный код, а значит, пользователь не может самостоятельно вносить изменения в систему, а также практически все приложения для неё платны, ну если вы добросовестный пользователь и не ставите Jailbreak, чтобы не потерять гарантию. Хотя и последнее не гарантирует вам свободу действий.



Устройства Apple, интерфейс iOS.

Первая версия мобильной операционной системы была построена на том же ядре Unix, что и Mac OS X. Но с первых минут презентации стало ясно, что отличия будут колоссальными.



iPhone OS 1 — 29 июня 2007.

Каким бы инновационным ни был iPhone на момент презентации, его функциональность крайне ограничена. Мало какие знакомые сейчас функции были реализованы в первой версии операционки:

- Основной интерфейс;
- Мультитач жесты;
- iPod;
- Safari;
- Карты;
- Синхронизация с iTunes.

В обновлениях появились:

- Веб-приложения на домашнем экране;
- Изменение расположения иконок;
- Клавиатура с поддержкой мультитач;
- iTunes Music Store.



iPhone OS 2 — 11 июля 2008.

Во второй версии операционки Стив Джобс признал идею использовать только веб-приложения провальной. Появление App Store стало одним из самых важных изменений за время существования мобильной операционки;

- App Store;
- Mobile Me;
- Нативные приложения от сторонних разработчиков;
- Поддержка Exchange;
- Поиск в приложении Контакты.

В обновлениях появились:

- Плейлисты iTunes Genius;
- Google Street View;
- Скачивание подкастов.

Главное событие iOS – это запуск App Store и возможность выкладывать туда сторонние приложения для смартфона. Такие магазины приложений были уже не редкостью, но именно такого шага ждали фанаты Apple, потому что потенциал у iOS был огромный.

У основных конкурентов, такие магазины были или слабо интегрированы в само устройства, либо их не было совсем. Именно поэтому это обновление опять сделала iOS лидером рынка мобильных ОС. Теперь, доступ к магазину контента, был у человека в кармане.



Магазин приложений App Store.

С точки зрения бизнеса – это идеальная схема, облегчающая процесс продажи музыки и приложений. Прямо через телефон теперь стало возможно скачать и купить песню или игру, оплачивая это со своей карты.

iOS SDK. Со второй версии операционной системы появилось SDK (Software Development Kit) – это комплекс средств для разработки приложений. Теперь любой разработчик программ или игр, сможет писать свои приложения для iPhone и iPod. Это самое ожидаемое нововведение в iOS 2, которое ждали со времён презентации первой версии в 2007 году.

Сама операционная система остаётся ещё закрытой, приложения можно устанавливать только с App Store, много функций айфона нельзя привязать ещё к компьютеру.



iPhone OS 3 — 17 июня 2009.

В третью версию операционной системы было добавлено много функций, которых так сильно не хватало пользователям, например поиск по смартфону и копирование/вставка. Кроме того, в версии 3.2 появилась поддержка планшетов.

Возможности:

- Вырезать, копировать, вставить;
- Голосовое управление;
- ММС;
- Съемка видео;
- Автофокус;
- Поиск Spotlight;
- Push уведомления;
- Ландшафтная ориентация клавиатуры;
- Find my phone;
- Компас;
- Возможность войти в аккаунт YouTube;
- Bluetooth A2DP;
- Покупки внутри приложений;
- Родительский контроль;
- Прямые покупки в iTunes;
- HTML5 в Safari.

В обновлениях появились:

- Genius;
- Загрузка рингтонов;
- Поддержка голосового управления по Bluetooth;
- Поддержка разрешения экрана iPad;
- Приложения для iPad;
- Поддержка Bluetooth клавиатур;
- iBooks.

Spotlight – удобный поиск. Был реализован поиск файлов, приложений, документов внутри системы. Основная идея – это создание быстрого поиска, для этого был добавлен ещё один рабочий стол, на котором находилась форма для ввода текста. Теперь iOS догнал по этому вопросу основных конкурентов (BlackBerry, PalmOS, webOS и Windows Mobile), у которых это уже было хорошо реализовано.

Также, в третьей версии была добавлена поддержка MMS, что на середину 2009 года уже было не актуально, потому что это дополнение уже теряло популярности из-за того что стал более доступен мобильный интернет.

iOS 3.0 была намного удачней второй версии, потому что разработчики учли много пожеланий простых пользователей и постарались максимально убрать все недочёты.



iOS 4 — 21 июня 2010.

Летом 2010 iPhone OS была переименована в хорошо всем знакомую iOS. Наконец-то появилась многозадачность, пусть урезанная, но это стало очередным скачком в развитии:

- Многозадачность;
- Мапки на домашнем экране;
- FaceTime;
- Мруппировка сообщений в Mail;
- iAd;
- Установка любых картинок в качестве заставки;
- Возможность дарить приложения;
- Поиск в приложении Сообщения.

В обновлениях появились:

- Game Center;
- iTunes Ping;
- Многозадачность на iPad;
- Папки на iPad;
- AirPlay;
- AirPrint;
- iPhone в качестве точки доступа;

- iTunes Home Sharing;
- AirPlay для сторонних приложений.

FaceTime. Четвёртая версия iOS появилась в iPhone 4, поэтому так сильно была изменена фронтальная камера. Теперь кроме видео связи, пользователь может создавать видео конференции. Проблема в том, что конференция может быть только среди устройств Apple (iPad, MacBook).

Пользовательские папки. После того, как прошло три года с выхода первой версии, наконец-то была добавлена возможность создания папок на рабочем столе, теперь ярлыки можно объединять в папки и благодаря чему получалось очень компактное меню. Это функция была первой среди сенсорных смартфонов. В Android добавили такую возможность только в версии 4.0 Ice Cream Sandwich.

После выпуска четвёртой версии, многие пользователи заметили, что iPhone 4 неправильно отображает информацию, показывающую уровень приёма сигнала мобильной связи.

Многие пользователи стали проверять и ранние версии айфона и оказалось, что они тоже не правильно отображают уровень сигнала. Дело в том, что деления уровня сигнала начинают уменьшаться, только когда телефон находится в зоне очень слабого сигнала и то на пару делений, хотя должно уходить все 4 или 5. Вроде бы ничего особенного в этой неточности нету, но на западе разыгрался целый скандал и Apple пришлось быстро исправлять проблему с датчиком сигнала. В июле 2010 года вышла версия 4.0.1 (и 3.2.1 iOS для iPad), она и исправляла проблему с делениями сигнала.

iOS 4.1 появилась в сентябре, в ней расширились игровые возможности аппаратов, добавились Game Center. Добавили возможность съёмки фотографий в качестве HDR и была добавлена возможность загрузки виде роликов на YouTube.

Следующие обновление iOS 4.2.1 привнесло папки и мультизадачность в iPad, добавили установку этой версии на все мобильные устройства (правда часть функций не работало на старых устройствах — iPhone 3G и iPod Touch 2G). Apple внедрила технологию для воспроизведения потокового видео AirPlay, оно работала между всеми девайсами Apple с Apple TV.



iOS 5 — 12 октября 2011.

Пятая версия мобильной операционки смогла похвастаться виртуальным ассистентом Siri, фирменным сервисом сообщений и облаком:

- Siri;
- Центр уведомлений;
- iMessage;
- iCloud;
- Синхронизация с iTunes по Wi-Fi;
- Включение новых устройств без подключения к компьютеру;
- Киоск;
- Напоминания;
- Интеграция с Twitter;
- Пробки и альтернативные маршруты на картах;
- Поддержка RTF в почте;
- Базовое редактирование снимков;
- Функция Reader в Safari.

В обновлениях появились:

- Измененное приложение Камера для iPad;
- Увеличение лимита скачивания приложений по 3G до 50 МБ;
- Удаление фотографий из Фотопотока.

Это версия операционной системы появилась с презентацией iPhone 4S в октябре 2011 года. Новых функций и возможностей добавилось очень много, но самые главные коснулись аппаратной части.

Технология Siri. Эта функций появилась в iOS 5.0 и была доступна только для iPhone 4S, она предоставляла возможность голосового контроля над телефоном, можно даже задавать вопросы, телефон будет пытаться отвечать на них. Только одна проблема – если вы не знаете английский язык, то у вас ничего не получится.

Погода и другая информация. В этом направлении iOS отставала от Android и в пятой версии разработчики решили эту проблему, добавив вывод на рабочий экран сведения о погоде, местонахождении, температуре воздуха.

iMessage. Apple решили позаимствовать интересную технологию у BlackBerry, которая позволяет обмениваться короткими сообщениями между устройствами с одной операционной системой. При установке iOS 5 на iPod Touch или iPad, эта бесплатная услуга будет также работать.

iCloud. Один из самых интересных нововведений в новой версии прошивки. Apple старается расширить функционал, хранения данных на удалённых серверах. iCloud и выполняет эту роль, он заменил старую версию MobileMe. Документы, фотографии и другие файлы теперь можно скопировать на хранение в облачное хранилище данных. Данные могут синхронизироваться между всеми Apple устройствами одного владельца.



iOS 6 — 11 июня 2012.

Самым значительным изменением в актуальной версии операционной системы стали собственные карты Apple, кроме того программисты немного «прокачали» Siri и добавили Passbook в качестве утешения всем, кто ждал NFC.

Возможности новой ОС:

- Фирменные карты Apple;
- Новые возможности Siri;
- Интеграция Facebook;
- Passbook;
- Синхронизация вкладок через iCloud;
- Улучшения приложения Mail;
- Звонки FaceTime посредством мобильного интернета;
- Общий Фотопоток;
- Функция «Не беспокоить»;
- Новые опции ответа на звонок;
- Поиск по всем полям в приложении Контакты;
- Полноэкранный режим в браузере в ландшафтной ориентации;
- Новый интерфейс виртуальных магазинов;
- Приложение Часы для iPad;
- Удалено приложение YouTube;
- Панорамная съемка.

Обновления iOS 6 не добавили существенных изменений и в основном были направлены на исправление ошибок и повышение стабильности.



iOS 7 — июнь 2013.

В июне 2013 года на WWDC представлена новая версия мобильной операционной системы. Джони Айв стал главой подразделения Human Interface и после обновления приложения подкасты можно оценить грядущие изменения интерфейса.

Apple называет операционную систему «самым большим изменением iOS после выхода оригинального iPhone». Наиболее заметным изменением является кардинально переработанный интерфейс. В рекламном ролике, показанном во время основной презентации, Джонатан Айв описал обновление как «внесение порядка в запутанность», подчеркивая следующие особенности: изысканные шрифты, новые иконки, прозрачность и слои. Дизайн iOS 7 и OS X Mavericks (представленной в тот же день) содержат значительно меньше элементов скевоморфизма (таких как зеленый стол в Game Center, деревянная текстура в приложениях Киоск и iBooks и кожи в Календаре).

«Установив iOS 7, получаешь новый телефон, но хорошо знакомый в использовании» Крейг Федериги.

Кроме того, произошли существенные изменения функциональности, которых было маловато в шестой версии прошивки.

Во время презентации новой операционной системы старший вице-президент Apple Крейг Федериги подчеркнул десять основных изменений:

- Пункт управления. Подобно Центру уведомлений, Пункт управления (Control Center) доступен с помощью жеста вверх в нижней части экрана и обеспечивает доступ к таким параметрам как авиа режим, яркость, управление мультимедиа, AirPlay, AirDrop и быстрый доступ к некоторым приложениям: фонарик, компас, калькулятор и камера.

- Многозадачность. iOS 7 основана на вытесняющей многозадачности, введенной в iOS 4. Поддерживается фоновое обновление приложений.

- Safari. В Safari стало доступно поле «умного» поиска, впервые представленное в Safari 6 для OS X Mountain Lion. Другие изменения включают в себя бесконечное число вкладок, родительский контроль, улучшенные обмен ссылками в Twitter и Список для чтения.

- AirDrop. С помощью функции AirDrop (впервые представлена в OS X Lion), доступной для iPhone 5, iPad (4-го поколения), iPad mini, и iPod touch 5-го поколения, можно обмениваться файлами (фотографиями, видео, карточками контактов и т.д.) с другими пользователями iOS-устройств.

- Камера. Новый интерфейс камеры позволяет переходить между четырьмя различными режимами: видео, фото, квадратные и панорамные фото. Также появилась возможность наложить на снимки один из девяти различных фильтров.

- Фотографии. Приложение «Фото» в iOS 7 использует информацию каждой фотографии, чтобы сортировать их по дате, месту и году съёмки, а также поддерживает загрузку видео в фотопоток (представлен в iOS 5).

- Siri. Интерфейс Siri также подвергся редизайну. Для каждого представленного языка появилась опция выбора между мужским и женским голосом. Улучшена интеграция с Твиттером, Википедией и поисковой системой Bing. Появилась возможность управлять системными настройками с помощью голосовых команд.

- iOS в Машине. «iOS в Машине» использует Siri, интегрированную в отдельные модели автомобилей для управления навигацией, телефоном, музыкой и iMessage на экране автомобиля. Выход ожидается в 2014 году.

- App Store. Новые функции: поиск по возрастной группе, «Рядом со мной», а также реализовано автоматическое обновление приложений.

- Музыка и iTunes Radio. Наряду с изменениями интерфейса, в приложение Музыка теперь встроен iTunes Radio (сервис потокового воспроизведения музыки, наподобие Spotify). В США служба стала доступна после выхода бета-версии iOS 7. Запуск в Великобритании и Европе ожидается позже.

Другие изменения включают в себя FaceTime Audio, многостраничные папки, чёрный список контактов, а также новые возможности Find My iPhone, препятствующие использованию потерянного или украденного iPhone или iPad посторонними.

Операционная система Android



Android – операционная система для смартфонов и планшетов, а в последнее время и для фотоаппаратов, телевизоров, mini-pc и прочих гаджетов, разработанная компанией Google на базе ядра Unix, как бесплатная операционная система с открытым исходным кодом. Эта система занимает лидирующую позицию среди ОС для телефонов и является основным конкурентом iOS. Для неё, как и для iOS, разрабатывается множество приложений и игр, на данной платформе выпускаются как бюджетные телефоны от 1500 рублей, так и довольно продвинутые и современные аппараты. Лидером по производству смартфонов на этой платформе является компания Samsung, также на Android телефоны выпускают LG, HTC, Sony, Lenovo и многие другие компании.

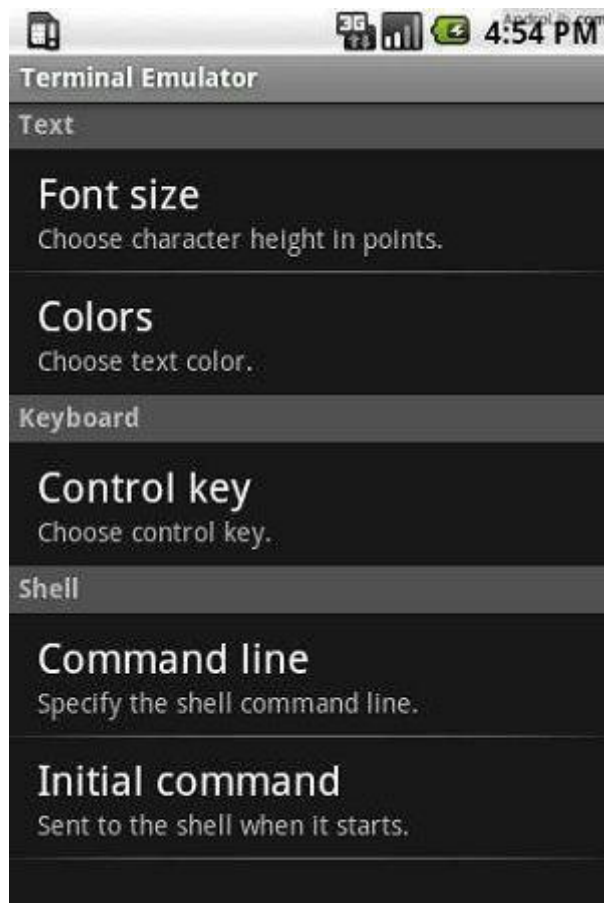


Интерфейс современного Android.
Android 1.0 – начало.



Первая версия мобильной операционки появилась в далеком 2008 году, и поначалу работала исключительно в руках разработчиков. В некоммерческой версии операционной системы были реализованы:

- Android Market;
- Поддержка камеры;
- Приложения Gmail и синхронизация;
- Синхронизация контактов с приложением People;
- Синхронизация календаря с приложением Calendar;
- Google Maps, Sync, Search и Talk;
- IM, CMC, MMC;
- Медиаплеер;
- Уведомления;
- Обои рабочего стола;
- Голосовой набор;
- Плеер YouTube;
- Wi-Fi и Bluetooth.



Android 1.0.

Android 1.5 — Cupcake.



30 апреля 2009 года была официально представлена первая потребительская версия «зеленого робота». В Android 1.5 появилось много изменений, которые сделали инженерный прототип хоть немного похожим на операционку для пользователей:

- Поддержка виджетов;
- Загрузка видео на YouTube;
- Загрузка фотографий в Picasa;
- Автоматическая смена ориентации экрана;
- Анимированные переходы между рабочими столами;

- Копирование и вставка в браузере;
- Запись и просмотр видео;
- Поддержка Bluetooth A2DP, AVRCP;
- Клавиатура с предикативным вводом.



Android 1.5.

Android 1.6 — Donut.



15 сентября 2009 вышло в свет обновление операционной системы. Android нуждался в доработках стабильности, производительности и безопасности. Это стало отличным поводом расширить функциональность:

- Быстрый поиск;

- Галерея, камера;
- Поддержка CDMA, EVDO, 802.1x, VPN;
- Индикатор аккумулятора;
- Преобразование речи в текст;
- Поддержка движка многоязычного произношения;
- Облегченный поиск приложений и скриншоты в Android Market;
- Поддержка разрешения WVGA;
- Новые API и инструмент для работы с жестами.



Android 1.6.

Android 2.1 — Eclair.



26 октября 2009 Google представил вторую версию мобильной операционной системы. В очередной раз было добавлено много необходимых функций, но операционка все еще была далека от совершенства:

- Поддержка нескольких аккаунтов;
- Поддержка Exchange и синхронизации почты с контактами;
- Bluetooth 2.1;

- Живые обои;
- Поиск по сохраненным СМС и ММС;
- Новые возможности камеры;
- Пошаговая навигация в картах Google;
- Поддержка различных размеров экрана и разрешений;
- Новые API;
- Поддержка HTML 5 в браузере.



Android 2.1.

Android 2.2 — Froyo.



20 мая 2010 появилось обновление операционной системы с многочисленными изменениями и нововведениями:

- Улучшенная поддержка Exchange;
- Движок Chrome V8 JS в браузере;
- Функция точки доступа;
- Отключение передачи данных по сотовой сети;
- Android to Cloud Device Messaging;
- Поддержка Flash;
- Поддержка высокого разрешения экрана;
- Поддержка многоязычной клавиатуры;
- Поддержка док-станций Bluetooth;
- Набор номера и передача контактов по Bluetooth;
- Фреймворки автомобильного и ночного режима.



Android 2.2.

Android 2.3 — Gingerbread.



6 декабря 2010 «зеленый робот» в очередной раз поумнел:

- Поддержка NFC;
- Поддержка новых размеров и разрешений экранов;
- Новый менеджер загрузок;
- Поддержка WebM/VP8, AAC и открытые API для нативного аудио;
- Клавиатура с мультитач;
- Улучшенная функция копирования и вставки;
- Видеочат в Google Talk;
- Поддержка SIP-телефонии;
- Новые аудиоэффекты.



Android 2.3.

Операционная система Windows Phone

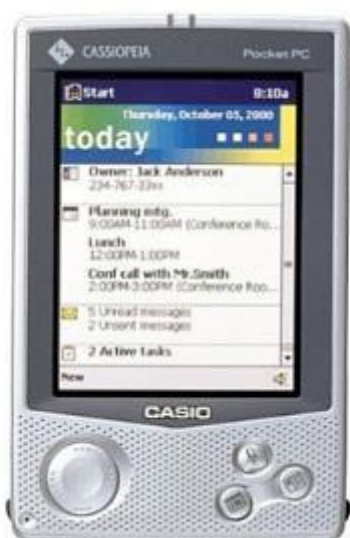


Данная система разработана компанией Microsoft для смены устаревшей мобильной операционной системы Windows Mobile, которая плохо подходила для сенсорных устройств. Операционная система спроектирована с нуля, поэтому у неё не очень много доступных приложений. Некоторым интерфейс данной операционной системы нравится, некоторым нет, как говорится, "О вкусах не спорят".



Интерфейс современного Windows Phone 8 (8.1).

История появления.



Windows SE.

В 1996 году появилась первая версия Windows CE, которая была урезанной версией Windows 95. Работы над мобильной и настольной операционкой велись совместно вплоть до 2000 года. Тогда их пути разошлись

и мобильная операционка стала по-настоящему мобильным продуктом. В июне 2000 года была представлена третья версия Windows CE с кодовым названием «Cedar». Она легла в основу операционных систем Pocket PC 2000 и Handheld PC 2000, чуть позже были выпущены Pocket PC 2002 и Smartphone 2002.

В 2003 году Pocket PC отправилась на свалку истории, и была представлена Windows Mobile 2003.



Windows Mobile 2003 — 23 июня 2003.

В 2003 году появилась первая мобильная операционная система из семейства Windows Mobile. До этого момента в мобильной операционке программисты реализовали основной набор софта для работы вне офиса, развлекательные приложения, например Windows Media Player 8 и модули беспроводных соединений.

- L2TP/IPsec VPN;
- Улучшенная поддержка Bluetooth;
- Приложение для просмотра картинок с функцией обрезания и пересылки файлов;
- Игра Jawbreaker;
- Опция отправки СМС в ответ на звонок в Phone Edition.



Windows Mobile 2003 SE — 24 марта 2004.

Обновление мобильной операционки было направлено на расширение модельного ряда устройств за счет поддержки новых разрешений экрана.

- Смена ориентации экрана;
- Разрешение экрана 640x480, 240x240, 480x480;
- Wi-Fi Protected Access;
- Просмотр страниц в одну колонку в Pocket Internet Explorer.



Windows Mobile 5 — 9–12 мая 2005.

На Developers Conference 2005 в Лас Вегасе была представлена новая версия мобильной Windows, в которой появилось огромное количество изменений и операционка избавилась от своего Pocket PC—прошлого в названиях приложений.

- Поддержка .NET Compact Framework 1.0 SP3;
- Среда для программ, основанных на .NET;
- Office Mobile;
- Windows Media Player 10 Mobile;
- Photo Caller ID;
- Улучшенная поддержка Bluetooth;
- ActiveSync 4.0;
- Интерфейс работы с GPS;
- Поддержка Persistent Storage;
- Microsoft Exchange Server;
- DirectPush.



Windows Mobile 6 — 12 февраля 2007.

Обновленная операционка была представлена в Барселоне на выставке 3GSM World Congress 2007, а в ее основу легла Windows CE 5.2. Windows Mobile 6 получила много новых функций, стала намного более производительной. Под управлением этой операционки появился HTC Touch:

- Первый смартфон, ориентированный на управление пальцем, а не пером;
- Поддержка IP-телефонии;
- Marketplace;
- Интеграция Windows Live;
- Шифрование карты памяти;
- Обновленный интерфейс;
- Windows Update;
- Exchange 2007;
- Microsoft Office OneNote;
- Поддержка HTML в Outlook Mobile;
- Новые разрешения экранов 320x320 и 800x480.



Windows Mobile 6.1 — 1 апреля 2008.

На выставке CTIA Wireless 2008 было представлено небольшое обновление операционной системы — Windows Mobile 6.1. В нем не было ничего особенного, но именно с устройств под управлением этой операционки началось распространение не только среди бизнес-сегмента, но и обычных пользователей.

Под управлением WM 6.1 вышел Samsung WiTu, получивший нестандартный на тот момент широкоформатный экран и ставший первым смартфоном Samsung с оболочкой TouchWiz:

- Масштаб и обзор страницы в Internet Explorer;
- Шифрование файлов на устройстве;
- Mobile Device Manager.



Windows Mobile 6.5 — 6 октября 2009 года.

В октябре 2009 состоялась долгожданная премьера Windows Mobile 6.5, которая получила существенное обновление функциональности. Одним из главных нововведений стала оптимизация интерфейса для управления пальцами. Под управлением WM 6.5 появился легендарный HTC HD2 — первый и единственный с поддержкой мультитач и до сих пор актуальный. Производителю удалось создать устройство, на котором энтузиастам удалось запустить практически все мобильные операционные системы, например, Android, Windows Phone 7, Ubuntu, MeeGo и даже Windows RT.

- Интерфейс для управления пальцами;
- Новый домашний экран Titanium;
- Новое меню;
- Internet Explorer Mobile 6;
- A-GPS.



Windows Mobile 6.5.3.

Долгая история Windows Mobile подошла к концу вместе с обновлением до версии WM 6.5.3. В ней было не много изменений и большинство из них были косметические, которые завершили переход от управления смартфонами стилусом к интерфейсу, ориентированном пальцем:

- Крупная иконка Пуск;
- Увеличенный нижний и уменьшенный верхний бар;
- Новые меню стандартных программ.



Windows Phone 7 — 15 февраля 2010.

В Барселоне на Mobile World Congress 2010 Стив Балмер анонсировал совершенно новую мобильную операционную систему — Windows Phone 7. Официально новую разработку представили 11 октября а уже 21 октября в продаже появились первые смартфоны. На самом деле бренд «Windows Phone» начали продвигать еще в Windows Mobile 6.5, но в 2010 году он начал работать в полную силу. Новая операционка полностью лишилась

совместимости с предыдущими версиями, получила кардинальное обновление интерфейса, полноценный магазин приложений и многое другое.

- Пользовательский интерфейс;
- Магазин приложений;
- Закрытая файловая система;
- Internet Explorer Mobile;
- Microsoft Office Mobile;
- Многозадачность;
- Синхронизация Zune.



Windows Phone 7.5 — февраль 2011.

На Mobile World Congress 2011 Стив Балмер представил первое крупное обновление Windows Phone. Программисты сделали более 500 улучшений и дополнений, которые положительно сказались на стабильности, быстродействии и удобстве операционки:

- Internet Explorer Mobile 9;
- Поддержка в браузере HTML5, Canvas, аппаратного ускорения;
- Поддержка фронтальной камеры;
- Поддержка 19 новых языков;
- Многозадачность в сторонних приложениях;
- Windows Live SkyDrive;
- Интеграция Twitter в People Hub.



Windows Phone 7.5 Tango — февраль 2012.

На Mobile World Congress 2012 мобильная операционка получило небольшое обновление, в котором уменьшились требования к «железу» для захвата рынков развивающихся стран недорогими смартфонами, например, Lumia 610:

- Поддержка процессоров с частотой 800 МГц;
- Поддержка устройств с 256 МБ оперативной памяти;
- Исправление ошибок.



Windows Phone 7.8 — конец 2012.

В конце 2012 года было объявлено об обновлении смартфонов до версии 7.8, которая получила часть функций новой, не совместимой со старыми устройствами, Windows Phone 8:

- Обои Bing на экране блокировки;
- Виртуальная кнопка спуска затвора в приложении Камера;
- Новые типы плиток;
- Новые цветовые схемы.



Windows Phone 8 — 29 октября 2012.

Windows Phone 8 — второе поколение Windows Phone. Операционка получила новую архитектуру Windows NT, что лишило владельцев старых смартфонов надежды на обновление и запуска приложений, созданных для 7.8:

- Живые плитки;
- Новый экран блокировки;
- Data Sense;
- Kid's Corner;
- Интеграция с SkyDrive;
- Internet Explorer 10;
- NFC и Wallet;
- Скриншоты.



Windows Phone 8.1 — 17 октября 2013.

В 2013 году вышла новая версия операционной системы Windows Phone 8.1, которая направлена на слияние настольной, планшетной и смартфонной «восьмерок». Началось полноценное объединение версий, приложения будут поставляться в качестве универсальных установщиков для всех устройств. Кроме того:

- Появился «Центр уведомлений»;
- Улучшена работа Data Sense;
- Поддержка USB On-The-Go
- Поддержка радио.

Заключение

Время подвести итоги и выделить отличительные черты каждой из рассмотренных операционных систем:

- iOS – закрытая система. Нет поддержки MicroSD-карт, зато есть магазин App Store. К плюсам можно отнести скорость работы (система способна работать даже на не самом мощном «железе») и множество приложений, в том числе написанных только под эту ОС. Устройства Apple изначально отличались дизайном, а их стоимость выше конкурентов.

- Android – открытая система. Очень много как устройств, на которые она установлена, так и производителей смартфонов (Samsung, HTC, Huawei, Oppo, LG, Lenovo, Meizu, Sony, Zopo, JiaYu, THL, Alcatel, Fly и прочие). Система с каждым обновлением становится все удобнее, в последних версиях появилась «плавность» работы, а по количеству приложений она лишь немного уступает iOS. В России стоимость аппарата на Android начинается с планки 1500 рублей. Это главный конкурент системы iOS.

- Windows Phone – закрытая система. В ее достоинства можно записать большое количество приложений, интересный интерфейс в виде «плиток», разнообразный модельный ряд, представленный смартфонами HTC, Nokia, LG, Samsung, Dell и прочими.

Каждая система хороша по-своему, но то, какую систему или устройство выбрать, решать уже конечному пользователю.

ЛЕКЦИЯ №6. ТЕХНОЛОГИИ РАЗРАБОТКИ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ НА ПЛАТФОРМЕ ANDROID.

Сейчас у всех есть сотовый телефон, смартфон или КПК. В каждом из них есть свои мобильные приложения. Они позволяют расширить возможности Вашего мобильного устройства. Мобильные приложения позволяют общаться с друзьями, выходить в интернет, смотреть погоду и многое другое. Существует множество технологий, которые используются для создания мобильных приложений и которые часто обновляются, а иногда и появляются новые. Но многие из них не дают возможность разрабатывать приложения для различных ОС, и это является существенным недостатком данных технологий.

В последнее время многие разработчики программного обеспечения (ПО) задумались о том, каким образом можно усовершенствовать технологии так, чтобы приложения были не только мощными, но и кросс-платформенными.

Целью данной статьи является выявление достоинств и недостатков платформы PhoneGap, с помощью которой можно разработать мощное приложение, которое будет поддерживаться на 6 платформах.

В статье также будет описано, как правильно настроить программную среду для разработки и как разработать простое приложение под ОС Android.

Обзор существующих технологий разработки мобильных приложений

В настоящее время существует хороший выбор языков программирования для разработки мобильных приложений. Это связано с тем, что для различных мобильных устройств приходится использовать различные языки программирования. Обычно это связано с тем, что мобильные устройства имеют различные ОС.

Ниже будут рассмотрены такие технологии как Java, Qt (основанный на библиотеках C++), Windows Phone SDK (написание на языке XAML), iPhone SDK (основной язык - Objective-C), Android SDK (основной язык - Java) и Symbian (основной язык - C++).

1. Java 2 Micro Edition (J2ME). В первую очередь J2ME это набор спецификаций и технологий, предназначенных для различных типов портативных устройств. Существуют два основных направления: Connected Device Configuration (CDC) и Connected Limited Device Configuration (CLDC). Направление определяет тип конфигурации центральных библиотек Java, а так же параметров виртуальной машины Java (в которой будут исполняться приложения). Логично предположить, что устройства CDC будут более «развитыми», в качестве примера можно привести коммуникаторы. К устройствам CLDC относятся обычные мобильные телефоны, аппаратно обладающие более скромными возможностями (ресурсами). Специальные режимы позволяют определять функциональность конфигураций для различных типов устройств. Режим Mobile Information Device Profile (MIDP) предназначен для CLDC портативных устройств с возможностью коммуницировать. Режим MIDP определяет функциональность - работу пользовательского интерфейса, сохранение настроек, работу в сети и модель приложения. CLDC и MIDP закладывают основу реализации J2ME [1].

Java-код интерпретируется непосредственно самим устройством при помощи так называемой Java Virtual Machine. Этот механизм делает возможным свободное распространение Java-приложений, так как они работают на всех устройствах с аналогичной Java-платформой [2].

Программирование Java-приложений и на сегодняшний день занимает большую часть, так как большинство мобильных устройств (в основном мобильные телефоны) в мире имеют уже предустановленную Java-машину.

2. Qt. Среда разработки Qt была приобретена Nokia в 2008 г. у норвежской Trolltech за 150 миллионов долларов. Qt в основном используется в качестве кросс-платформенной среды, которая позволяет использовать написанные с ее помощью приложения на различных устройствах и операционных системах, в том числе Windows, Mac OS X, Linux, Symbian, Android и других [3]. Начиная с версии Qt 4.0 появилась возможность программировать для мобильных устройств. С растущей пользовательской базой Qt, растёт потребность во встроенных, мобильных приложениях и UI-разработчиках.

Qt является одной из самых удачных библиотек для C++. Отладка приложений, разработанных для мобильных устройств, происходит с помощью эмулятора, который содержится в среде разработки. Таким образом, мы можем писать сложные приложения для мобильных устройств с использованием библиотек C++ и поддержкой кроссплатформенности.

В настоящее время последняя версия - Qt 5 бета. Финальный релиз планируется на 2012 год. Для работы Qt на мобильных устройствах необходима установка соответствующего Фреймворка.

3. Android SDK. Для разработки под Android можно использовать среду Eclipse с установленным плагином ADT. Разработка ведется на языке программирования Java. Есть возможность отладки с использованием эмулятора встроенного в ADT или непосредственно на мобильном устройстве с ОС Android.

Существует различные версии SDK, которые используются для написания кода для различных версий Android. В настоящее время большое распространения получили версии 2.2 и 2.3. Поддерживается почти полная обратная совместимость версий.

Кроме разработки на языке Java поддерживается возможность более низкоуровневая разработка с использованием Android NDK (Native Development Kit) на языке C/C++.

4. Symbian и C++. Для написания приложений под Symbian можно использовать язык программирования C++. В основном данный подход используется для Symbian OS v6.1, 7.0, 7.0s и 8.0 [6].

Разработка для Symbian OS (если говорить о C++) обычно ведется на ПК. Среда разработки - привычная многим программистам Visual Studio, это также могут быть IDE Metrowerks CodeWarrior Development Studio, Borland C++BuilderX Mobile Edition, Carbide.C++ (относительно новая IDE, созданная компанией Nokia на базе Eclipse), снабженная дополнительными инструментальными пакетами (SDK). Разработчику доступны практически все привычные возможности в отношении как создания ПО, так и отладки (трассировка, просмотр переменных, стека вызовов, структур классов и др.).

Отлаживаемая программа запускается в эмуляторе Symbian OS. Отметим, что эту подсистему правильнее было бы назвать симулятором, поскольку имитируются не аппаратные средства, а лишь программное окружение (соответствующие API операционной системы, реализованные поверх API Win32). При этом программные модули, которые загружаются в эмулятор, представляют собой исполнимые файлы для архитектуры x86 (не ARM, на базе которой построены смартфоны), соответствующее ПО для целевой платформы формируется после итоговой компиляции. Это предполагает определенную специфику (скажем, ранее была довольно распространена ситуация, когда программа, нормально функционировавшая в среде эмулятора, отказывалась работать на реальном устройстве), но сегодня эмулятор обеспечивает достаточно высокую степень сходства и проблемы возникают лишь при создании программ, нестандартно использующих API [7].

Выводы по результатам обзора. С появлением новых технологий, ранее использовавшиеся уходят в историю. Сейчас разработчиков, которые используют такие технологии как Symbian с использованием C++ и/или J2ME, становятся все меньше и лидирующую позицию занимают технологии, использующие различные SDK (Windows Phone SDK, iPhone SDK, Android SDK). Но недостаток всех существующих SDK в том, что разрабатываются нативные приложения, т.е. приложения, функционирующие под управлением только одной ОС.

ЛЕКЦИЯ №7. ТЕХНОЛОГИИ РАЗРАБОТКИ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ НА ПЛАТФОРМЕ IPHONE.

Профессия программиста — одна из самых престижных и востребованных, это факт. В отрасли разработки программного обеспечения есть свои ниши. Из них самая быстроразвивающаяся, а вместе с тем наиболее высокооплачиваемая — это разработка ПО для мобильных устройств. Как тебе известно, так уж сложилось, что, оставив конкурентов далеко позади, вперед вырвались три постоянных соперника. Фишка легла таким образом, что кто-то из этих трех безоговорочный лидер, кто-то всегда догоняет.

В сегменте мобильных технологий, без сомнения, лидирует Apple со своим смартфоном iPhone. На прошедшей в сентябре презентации компания из Купертино продемонстрировала уже 7-ю версию мобильного девайса. Вместе с ней Apple представила новую версию мобильной операционной системы iOS 10. Теперь это самая значимая операционка в мире, как и ее предшественницы. Из этого следует, что изучение разработки для iOS — самое выгодное вложение времени и средств.

«Яблочная» экосистема

Сегодня мы займемся разработкой простого мобильного приложения для iOS. Конечно, лучше всего для этого подходит родной Mac. Но если ты сторонник Windows или у тебя просто нет Мака, можно установить операционную систему macOS под виртуальную машину в Windows. На просторах Сети тебе не составит большого труда найти специально заточенную под PC версию macOS, в народе называемую Хакинтош. Смело ставь ее на виртуалку VMware, VirtualBox — на твой выбор. Сразу предупрежу, что на PC-клавиатуре нет части клавиш, которые есть на Mac-клаве, и многие их сочетания нажать в Хакинтоше невозможно. Это изрядно портит настроение. В остальном все описанные ниже операции можно успешно выполнить на Хакинтоше.

Необходимые знания

Чтобы программировать вообще и для iOS в частности, надо довольно много знать. Математика и логика, может, и не понадобятся вначале, но будут востребованы потом. Современные технологии избавили программиста от необходимости досконально знать архитектуру компьютера, однако понимать базовые механизмы, такие как системы счисления, их преобразование, скорость работы подпрограмм или эффективность алгоритмов (большое O), необходимо.

На более высоком уровне iOS-разработчику надо глубоко знать операционную систему macOS и собственно саму iOS. Также тебе обязательно надо освоить «яблочный» язык программирования. Знание Delphi, C++, C# или VB.NET тебе не сильно поможет. У Apple своя экосистема со своими языками: Objective-C и Swift. Конечно, знание C++ еще никому не навредило, программы и вставки на нем встречаются везде, даже у Apple. Но самый используемый по определению Objective-C, просто потому, что появился гораздо раньше (в середине восьмидесятых годов прошлого века), тогда как Swift лишь два года назад (в 2014-м). Apple возлагает большие надежды на новый язык и много инвестирует в него. Для поддержки старого софта используется Objective-C, а новый пишется на Swift. Поэтому тебе неплохо знать оба.

Чтобы не тратить время зря, все эти языки программирования можно изучить под чутким контролем преподавателей на сайте [GeekBrains](https://www.geekbrains.com/).

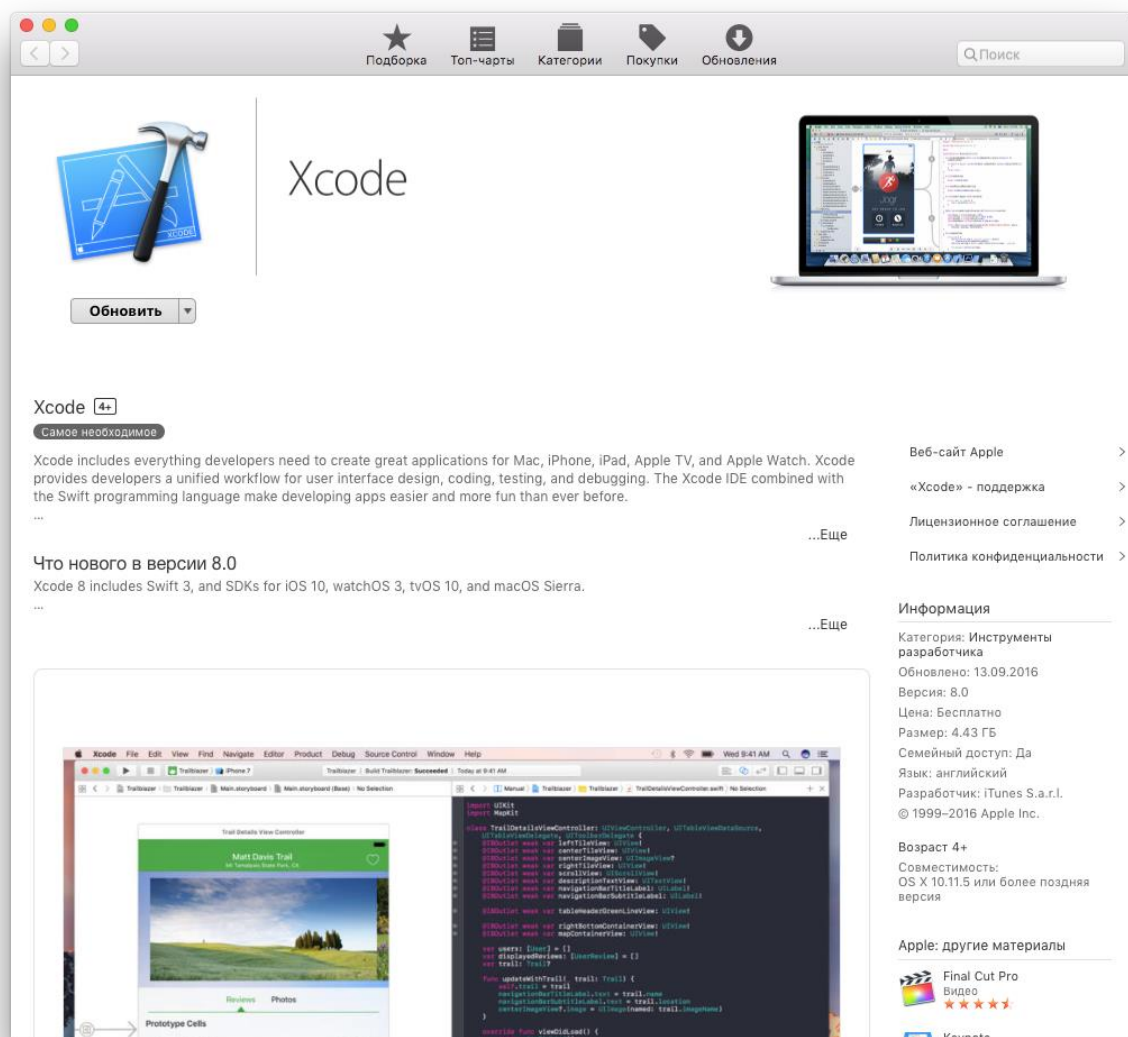
Инструменты для разработки

Как ты знаешь, главный инструмент разработчика под macOS и вместе с ней iOS — среда программирования Xcode. Она включает средства для создания приложений для Mac, iPhone, iPad, Apple TV, Apple Watch. Все платформы Apple. Xcode содержит средство построения интерфейса (Interface Builder), редактор кода, поддерживающего все современные средства работы с кодом. Кроме того, не выходя из Xcode, можно протестировать приложение; если оно разрабатывается для внешнего устройства, то его можно запустить в эмуляторе.

В систему включены эмуляторы всех устройств, новые версии которых можно скачать. В дополнение Xcode содержит графические инструменты для анализа производительности приложения, включая средства диагностирования загрузки CPU, использования накопителя (HDD, SSD и другие), нагрузки на графический адаптер (со стороны OpenGL).

13 сентября вышла новая, горячо ожидаемая версия среды разработки 8.0. Стабильные версии можно бесплатно скачать в App Store. Альфы и беты распространяются по подписке разработчика. Восьмая версия вобрала в себя: новую версию языка Swift 3, средство изучения Swift Playground для iPad, новую версию Interface Builder, которая стала быстрее работать и позволяет просматривать макет приложения на разных платформах, без необходимости развертывать на симуляторе само приложение.

Если у тебя старая версия Xcode, то рекомендуем обновить, поскольку мы будем использовать последнюю версию. Скачать ее можно из App Store:



Xcode 8 в Mac App Store

На Xcode 8 можно разрабатывать приложения для macOS Sierra, iOS 10, watchOS 3 и tvOS 10. Имеются все SDK. Новые версии операционок стали последовательно появляться аккуратно после презентации Apple, состоявшейся в начале сентября.

Язык Objective-C

Кратко вспомним основные сведения о языке. Objective-C — компилируемый объектно ориентированный язык программирования, используемый для написания приложений под «яблочные» системы. Представляет собой надмножество или, другими словами, надстройку над C, унаследовав от последнего синтаксис, типы данных, способы контроля за ходом выполнения программы и добавив к нему возможности объектно ориентированного программирования — описание классов и методов.

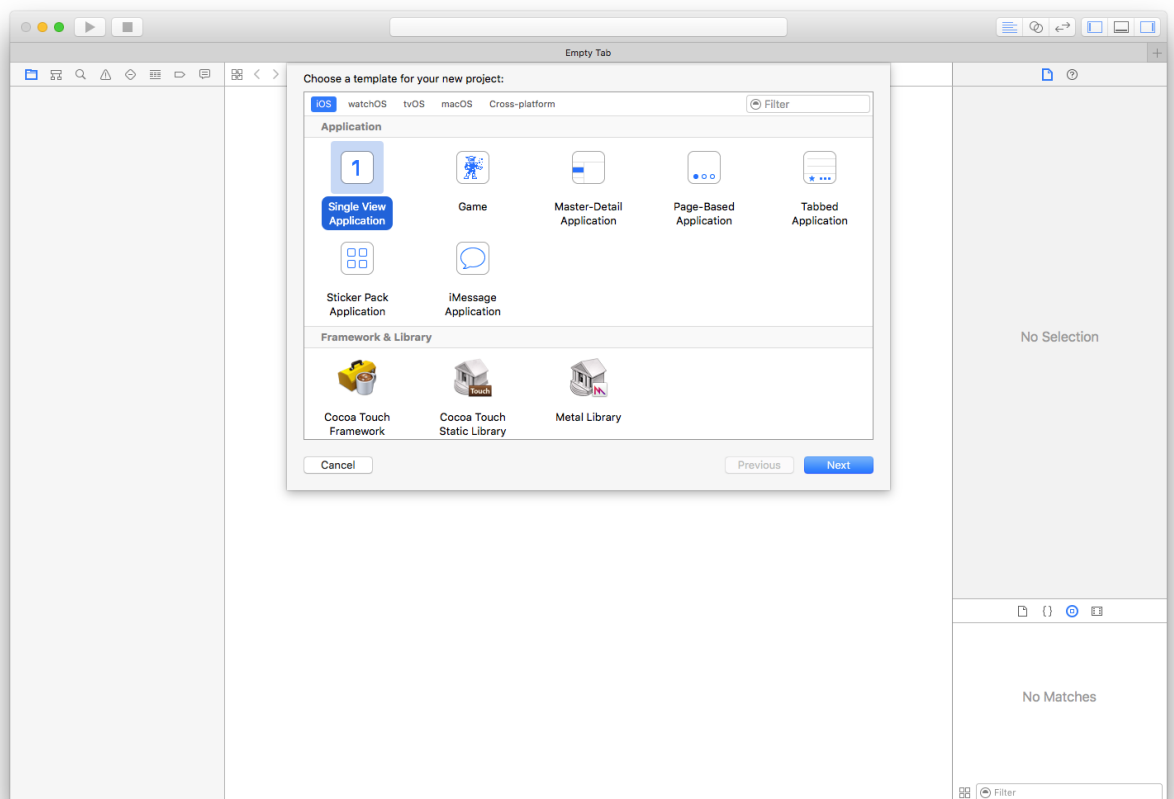
Как в C++, в Objective-C центральную роль играют объекты, с которыми связаны четыре парадигмы: абстракция, инкапсуляция, наследование, полиморфизм. Язык C++ является строго типизированным, тогда как Objective-C — слабо типизированный или обладает динамической системой типов данных. Подобно Smalltalk, в Objective-C объектам посылаются

сообщения. Это исключительная возможность динамических языков, так как тип объектов проверяется во время исполнения программы, а не во время компиляции.

Разработка приложения для iOS

В качестве семпла мы разработаем весьма простое приложение. Не будем отступать от священной традиции, поприветствуем мир. Однако с учетом того, что у нас графическое приложение, мы немного разнообразим его: добавим кнопку, нажимая которую будем отображать или скрывать надпись «Hello, World».

Запусти Xcode, в появившемся окне выбери Create new project или в главном меню File -> New -> Project. В следующем окне будет предложено выбрать целевую платформу и тип приложения.



Мастер создания приложений

В данном случае нас интересует платформа iOS. На выбор представлено семь типов приложений. Шесть из них — заготовки типичных приложений операционной системы iOS, по умолчанию содержащие разный набор компонентов. Седьмой тип — игра.

Типы приложений

Заготовка Single View Application предназначена для простого приложения с одним экраном. Заготовка включает компонент View Controller, который позволяет настроить вид приложения с помощью конструктора форм Interface Builder.

Master Detail Application создает приложение, где в табличном представлении отображается коллекция объектов. После выбора одного из них

показываются подробные сведения об этом объекте. Первый вид — мастер, второй — детализация.

С помощью Page-Based Application создаются приложения, имеющие несколько экранов, как страницы в книге. Следовательно, из этой заготовки создаются, например, читалки.

Tabbed application позволяет создавать приложения, где на каждый экран можно переключиться в любой момент, то есть у каждого экрана есть своя кнопка для его активации, на которой отображается заголовок. В качестве примера можно привести iTunes.

Game служит для создания заготовки игры. На выбор предлагаются четыре фреймворка для создания игры: SpriteKit, SceneKit, OpenGL ES, Metal. Вкратце рассмотрим их. SpriteKit представляет собой систему для двумерной визуализации и анимации текстурированных прямоугольников — спрайтов. При отображении кадров используется стандартный цикл, кадр выводится после того, как обработку пройдет все содержимое сцены. SceneKit — высокоуровневый фреймворк для визуализации трехмерной графики без помощи OpenGL. Он поддерживает загрузку, манипуляцию трехмерными объектами. В него включены: физический движок, генератор частиц и легкий способ скриптования. OpenGL ES — стандарт в области компьютерной графики. Позволяет визуализировать 2D- и 3D-сцены. Позволяет описать пайплайн для видеокарты: вершины проходят преобразование, собираются в примитивы, которые растеризуются в двумерное изображение и выводятся на экран. В пайплайн можно включить программируемые шейдеры. Metal — представляет собой низкоуровневый API, который позволит тебе выжать все мощности из твоего видеоадаптера. Отлаженные API вместе с предкомпилированными шейдерами и многопоточностью позволяют твоей игре подняться на новый уровень производительности и качества.

Sticker Pack Application — новый вид приложений, появившийся в iOS 10 и Xcode 8. Представляет собой набор простых или анимированных изображений, используемых в новом iMessage. Для своего создания не требует кодирования.

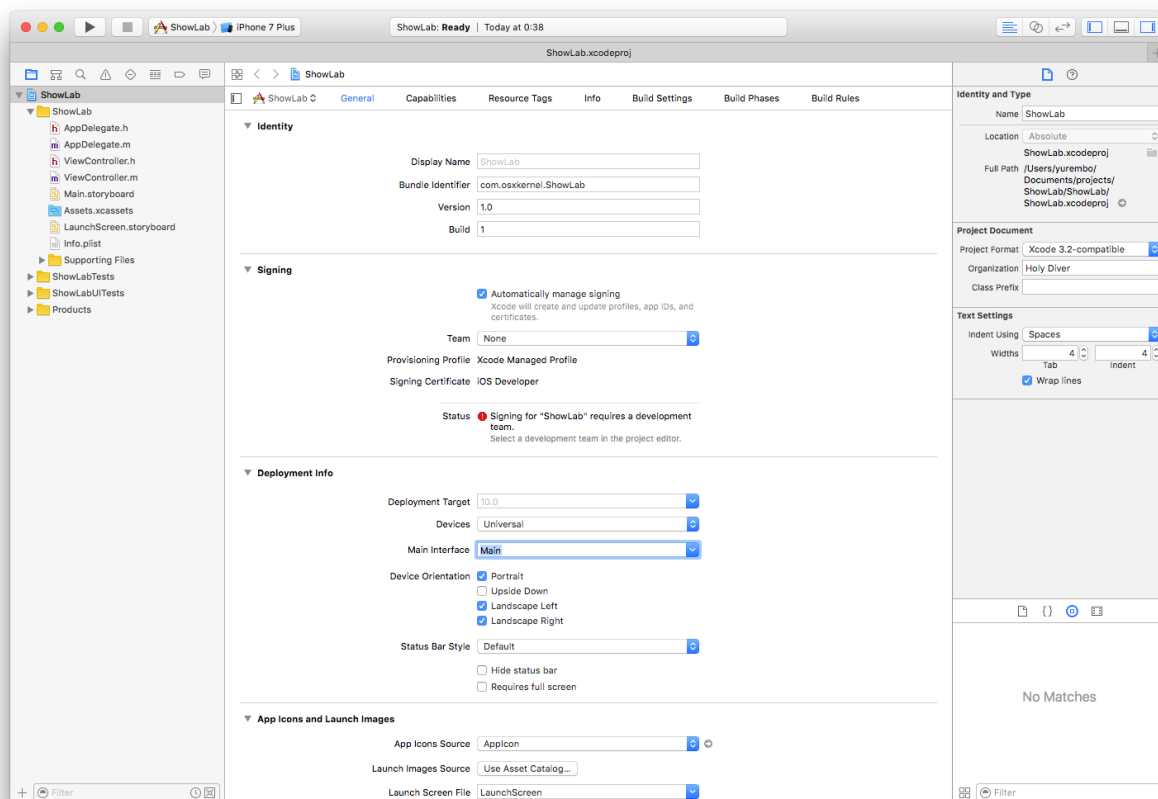
iMessage Application — новый вид приложений, появившийся в iOS 10 и Xcode 8. Позволяет создавать дополнения для iMessage, например для покупки и загрузки твоего пакета стикеров. Также можно, воспользовавшись API iMessage, создать свой аналог данного приложения, включая проигрывание аудио, видео, использование стикеров и другое.

Создание первого проекта

В качестве заготовки для нашего приложения выберем Single View Application. Поскольку мы не будем разрабатывать большую программу, нам хватит предоставляемой этой заготовкой средств. Нажимаем Next. На следующей странице мастера надо ввести имя проекта, к примеру ShowLab. В выпадающем списке Language оставим выбранный по умолчанию язык — Objective-C. Далее в выпадающем списке Devices оставим выбор Universal.

Здесь определяется, для какого устройства (iPhone или iPad) создается приложение. Пункт Universal означает для обоих. Снимем флажки Include Unit Tests и Include UI Tests, нам никакие тесты не нужны. Next. Выберем папку для сохранения проекта. Теперь жми кнопку Create.

В результате откроется окно со списком всех примененных к проекту параметров. В этом окне можно изменить установленные ранее в мастере параметры: ориентацию, целевой девайс и прочее.



Свойства проекта

Сначала нам надо создать интерфейс приложения. Для этого одним кликом выбери файл Main.storyboard в списке слева (если этот файл не видно, разверни содержимое папки ShowLab). Правее списка все окно займет Interface Builder. В центре отобразится макет устройства. В правом нижнем углу окна находится панель компонентов. Перетащи оттуда на макет компоненты Label и Button. Выше панели компонентов находится список свойств. Если у тебя он отсутствует, нажми кнопку Show the Attributes Inspector, находящуюся под заголовком окна в правой части интерфейса.

Выдели в макете компонент Label и настрой его свойство Text: в ниспадающем списке оставь выбор Plain, в строку ниже введи нужную надпись, в нашем случае «Hello, World». Если текст не помещается в границы надписи, измени их, перетаскивая маркеры на краях компонента. Чтобы централизовать его по горизонтали, перейди на страницу Size Inspector, нажав на кнопку Show the Size Inspector (справа от Show the Attributes Inspector). На этой странице из ниспадающего списка Arrange выбери пункт Center Horizontally in Container.

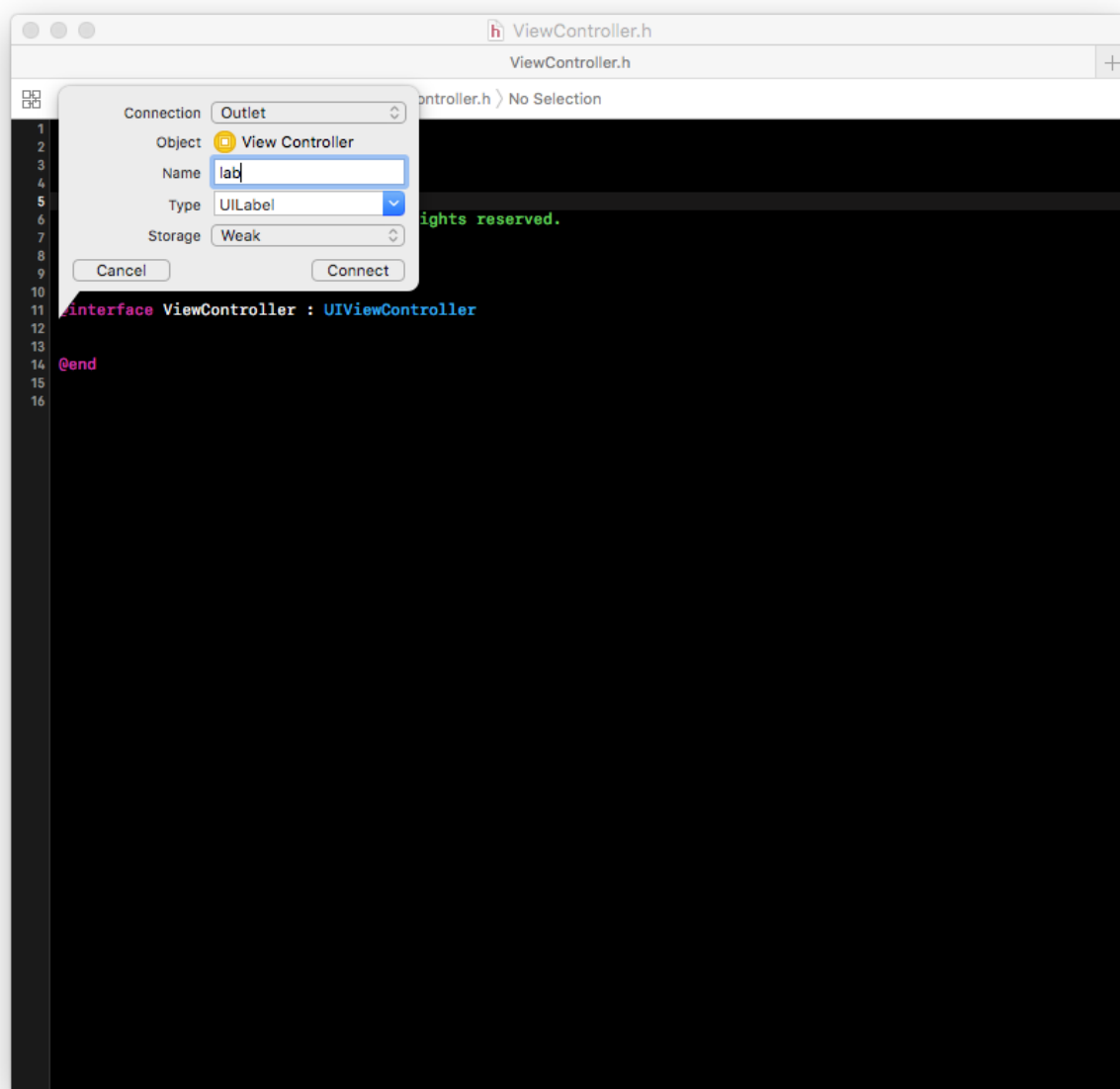
Теперь выбери компонент Button, измени его свойство Text на желаемую метку — Switch. Отцентрировать можешь так же, как описано выше.

Создаем связь между графическими элементами и кодом

В Visual Studio (или Delphi) объект в коде создается автоматически в тот момент, когда кладешь визуальный компонент на форму. В Xcode этого не происходит, но это не создает проблем.

Открой содержимое заголовочного файла ViewController.h в отдельном окне, дважды щелкнув на нем. В этом файле находится объявление расширения класса UIViewController, помечается ключевым словом @interface. Такая возможность была добавлена во вторую версию Objective-C. Теперь выполни такой трюк: перемести курсор мыши на компонент — текстовую метку, зажми клавишу Ctrl и левую кнопку мыши. Перемести курсор в окно с кодом (файл ViewController.h), за курсором потянется синяя линия. Отпусти мышь и клавишу внутри описания интерфейса ViewController.

Появится окно создания Outlet'a.



Создание Outlet'a

Это свойство объекта, которое ссылается на другой объект (в данном случае визуальный компонент). Тебе надо ввести имя объекта Outlet, по нему ты будешь обращаться к визуальному компоненту, пускай будет lab. Далее выбирается тип объекта, он выбран правильно: UILabel.

Еще ниже в списке Storage выбирается тип ссылки на объект: weak или strong. Если выбрать strong, то объект, на который указывает свойство, будет существовать до тех пор, пока свойство указывает на него, в таком случае он не сможет автоматически удалиться, когда перестанет использоваться. С другой стороны, когда действует слабая ссылка (weak), объект может самоуничтожиться. Итак, выбираем тип ссылки weak и жмем кнопку Connect. В итоге в код добавится такая строка:

```
@property (weak, nonatomic) IBOutlet UILabel *lab;
```

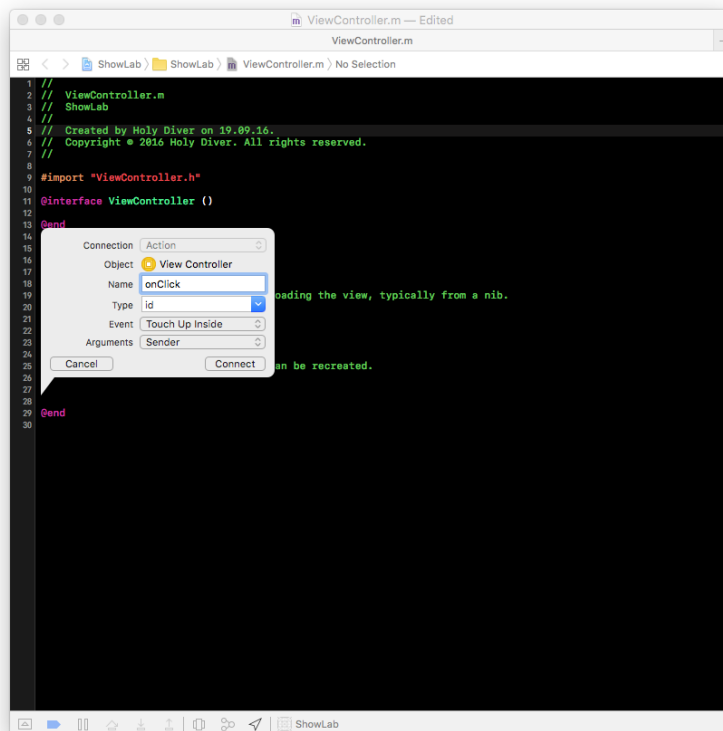
Убеждаемся, что Outlet — это свойство.

Теперь создадим Outlet для кнопки. Алгоритм остался прежним. Только для свойства Name надо ввести другое имя, например but. В код будет добавлена строка:

```
@property (weak, nonatomic) IBOutlet UIButton *but;
```

В результате имеем два указателя на визуальные компоненты: lab и but — соответственно, надпись и кнопку. Теперь, используя указатели, мы можем манипулировать компонентами в коде.

Затем надо создать обработчик события нажатия кнопки. Для этого в отдельном окне открой файл реализации ViewController.m. Точно таким же образом, как ты перетаскивал линию в заголовочный файл для создания аутлета, от кнопки перетаски линию в файл реализации и отпусти до закрывающей командной скобки — @end. Появится окно для создания события, подобное окну создания аутлета. Видишь разницу: в заголовочном файле создается ссылка на объект, в файле реализации — метод.



Создание обработчика события

Заполни поле Name, его значение представляет имя свойства — метода. Пусть будет onClick. Значение поля Type оставь по умолчанию — id. В языке Objective-C данный тип является предком для всех остальных. В ниспадающем списке Event выбрано по умолчанию событие Touch Up Inside, которое происходит в момент отпускания указателя (мыши, пальца...) над кнопкой, то есть заключительная стадия нажатия кнопки. Это то, что нам нужно. В списке Arguments оставим значение по умолчанию: Sender — это объект, отправивший данный сигнал, в нашем случае им всегда будет кнопка. Жмем кнопку Connect. В итоге будет добавлен следующий код:

```
- (IBAction)onClick:(id)sender {
}
```

Минус в начале означает закрытый метод (private). Ключевым словом IBAction помечаются события (методы) визуальных компонентов из Interface Builder.

Между командными скобками напишем выполняемый при нажатии кнопки код:

```
_lab.hidden = !_lab.hidden;
```

В этой строчке кода мы инвертируем значение свойства hidden. Оно имеет тип BOOL, который располагает двумя значениями: YES — истина и NO — ложь (несколько непривычно Windows-программерам, где true и false).

Обрати внимание на символ подчеркивания, стоящий перед именем объекта — надписи (lab). Без него компиляция завершится с ошибкой. Символ подчеркивания добавляется автоматически для объектов, содержащихся в других объектах; то есть в данном случае объект lab содержится в объекте ViewController. Раньше это было условное правило,

используемое для различия объектов, объявленных в качестве членов класса, и локальных объектов. А теперь это строгое правило, реализованное в компиляторе языка.

Сейчас можно скомпилировать приложение и запустить его на симуляторе. Мы выбрали симулятор для iPhone 7, входящий в Xcode 8. Кнопка компиляции и запуска представляет собой черный прямоугольник и находится на панели вверху интерфейса.

После компиляции приложения, запуска симулятора и загрузки нашего приложения на экране симулятора отобразится его интерфейс: надпись «Hello, World» и кнопка Switch. Если нажать последнюю, надпись исчезнет, при повторном нажатии она появится вновь.

Мы сделали очень простое приложение для понимания сути разработки под iOS.

Язык Swift

В 2014 году Apple представила новый язык программирования — Swift. Он быстро стал самым обсуждаемым и наиболее динамично развивающимся языком в мире. Это надежный, безопасный, интуитивно понятный язык для разработки приложений под операционные системы macOS, iOS, tvOS и watchOS для всего парка устройств от Apple. Swift — эффективный язык программирования. Его легче изучать, чем его предшественника — Objective-C, в то же время Swift позаимствовал из него самые удачные идеи. Вместе с тем разработчики сделали языки совместимыми, то есть в одной программе может быть код как на Swift, так и на Objective-C. В мире имеются миллиарды строк кода и сотни тысяч унаследованных программ, написанные на Objective-C, поэтому от его поддержки точно не будут отказываться.

При разработке Swift ставилась цель создать самый удобный язык системного программирования — быстрый, безопасный и выразительный.

Алгоритм на языке Swift работает в 2,6 раза быстрее, чем такой же алгоритм на Objective-C, и в 8,4 раза быстрее, чем на Python. Использование Swift делает процесс разработки программ более гибким и удобным.

Благодаря своему экспоненциальному развитию, 3 декабря 2015 года язык Swift был передан в сообщество открытых исходников. При этом Apple строго следит за его развитием, организовав комитет для его разработки. Теперь Swift доступен не только в системах Apple, но также в Linux. Все дополнительные инструменты для работы с языком, включая отладчик, стандартную библиотеку, менеджер пакет, тоже свободные и открытые.

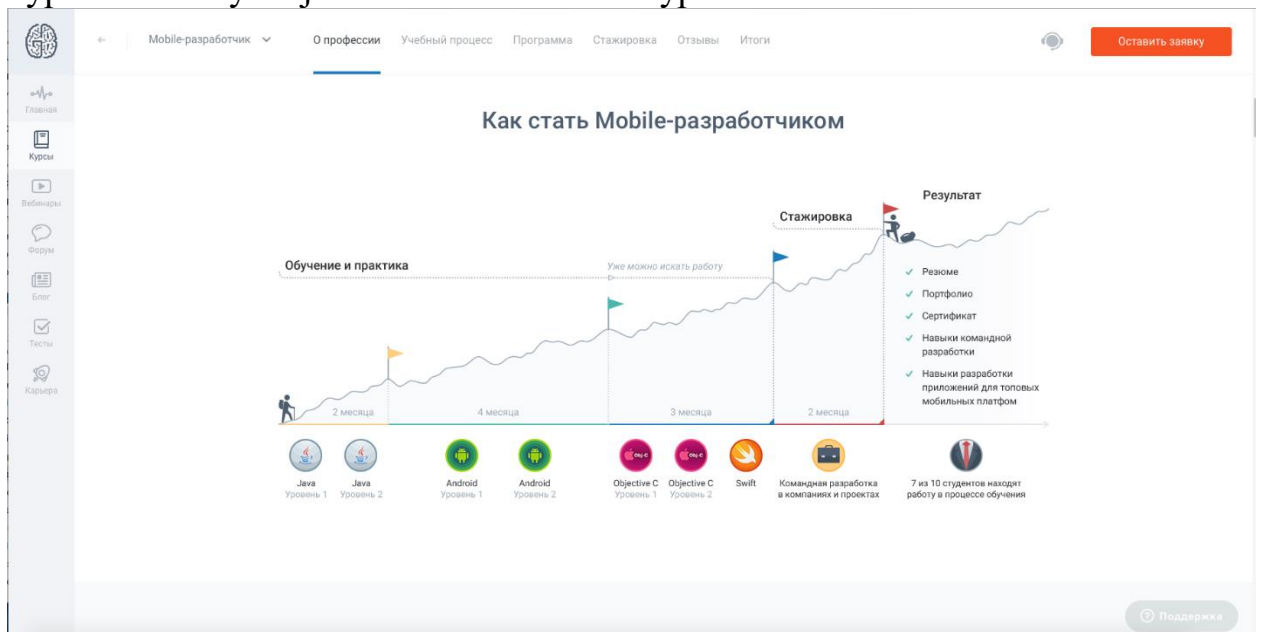
За этим языком будущее. App Store

Попросту говоря, App Store — это магазин контента для всего спектра Apple-девайсов. Здесь можно купить игры и приложения как от больших компаний, так и от независимых разработчиков. Каждый разработчик, в какой бы стране он ни жил, может продавать или распространять бесплатно свои программы на этой площадке цифровой дистрибуции. Чтобы начать это делать, надо зарегистрироваться как разработчик и купить лицензию для продажи своих приложений. Это плата за предоставляемый сервис и развитие магазина App Store.

Итоги

Мы разработали простое приложение для мобильной операционной системы iOS, используя для этого систему программирования Xcode вместе с языком Objective-C. Протестировали готовое приложение на симуляторе iPhone 7. Мы узнали, как построить пользовательский интерфейс с помощью Interface Builder, как связать графические элементы с кодом приложения, как создать обработчики событий. Познакомились с основами Objective-C. Кроме того, мы обратили внимание на перспективный язык Swift, используемый для разработки мощных и безопасных приложений для «яблочной» экосистемы.

Это достойный курс: пройдя его, ты точно не потратишь время зря. Он начинается с двух подробных курсов по языку Java, затем следует изучение базового программирования под мобильную операционную систему Android. Далее курс по базам данных в общем и языку SQL в частности. Затем углубленный курс по разработке профессиональных приложений для Android. После этого вектор изучения перейдет в сторону Apple-систем: два подробных курса по языку Objective-C и полезный курс по Swift.



ЛЕКЦИЯ №8. ТЕХНОЛОГИИ РАЗРАБОТКИ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ НА ПЛАТФОРМЕ WINDOWS PHONE.

Современный рынок мобильных устройств требует подготовки специалистов, способных быстро осваивать и создавать новые приложения под новые платформы. Среди них мобильная платформа Windows Phone 7 [1] в настоящее время считается наиболее перспективной. Разработана новая методика создания приложений (рис. 1),



Рис. 1. Структура методики создания мобильных приложений

включающая следующее:

- 1) подготовку окружения, в т. ч. установку средств разработки приложений – Visual Studio 2010, Service Pack 1 и Windows Phone SDK [3];
- 2) создание проекта;
- 3) анализ интерфейса и макета внешнего вида приложения;
- 4) анализ архитектуры проекта, который предполагает проектирование схемы классов будущего приложения;
- 5) разработку кода разметки интерфейса на языке разметки xaml;
- 6) разработку программной логики на языках C#, Java и Visual Basic) с использованием технологий .NET Framework 4, Silverlight и XNA;
- 7) отладку на симуляторе и реальном устройстве;
- 8) дальнейшее совершенствование проекта.

Методика демонстрирует основные принципы разработки приложений и учит использованию важных возможностей платформы.

Основная задача при разработке нового приложения заключается в создании пользовательского интерфейса на языке разметки xaml (eXtensible Application Markup Language) и логики на языках программирования. xaml –

это расширяемый язык разметки приложений, разработанный компанией Microsoft. Он основан на XML, содержит интуитивно понятные онтологические структуры, прост в освоении. Тесное взаимодействие объектов xaml и программной логики позволяет без труда работать с элементами управления в коде приложения. Разработанная методика включает три группы лабораторных работ и курсовую работу, в рамках которой студенты создают собственные приложения. Каждая из групп лабораторных работ объясняет и демонстрирует на практике следующие возможности: 1) варианты разметки (Grid, StackPanel, Canvas), основные элементы управления, контекст ввода, навигацию между страницами и передачу данных, элементы Pivot и Panorama, меню приложения; 2) задачи запуска и выбора, элемент управления WebBrowser, элемент управления Map и определение местоположения, работа с акселерометром, работа с HTTP, работа с изолированным хранилищем и локальной базой, привязка данных, основы Linq; 3) восстановление приложений из состояний Dormant и Tombstoned, работа с фоновыми службами, фоновая загрузка файлов, воспроизведение музыки, работа с будильником, живые плитки (Live Tiles), некоторые элементы управления расширения Microsoft.Phone.Controls.Toolkit. Каждая работа включает описание задачи и методику решения.

Структура описания включает:

- 1) формулировку задания;
- 2) определение осваиваемых тем;
- 3) описание типового решения;
- 4) возможные способы совершенствования.

В первом пункте определяется, какое приложение требуется разработать и какими функциями оно должно обладать. Второй пункт содержит список изучаемых тем, что позволяет студентам, предпочитающим самостоятельно во всем разбираться, узнать, какие конкретно вопросы им необходимо изучить. Третий пункт содержит полное описание методики разработки приложения. Описание включает пошаговые инструкции, примеры кода и снимки экрана. Создание каждой страницы разбивается на три этапа: разработку кода разметки страницы; подключение необходимых библиотек; разработку кода программы. Четвертый пункт описания лабораторной работы содержит список возможных совершенствований приложения. Для иллюстрации применения методики приведено описание курсовой работы «Тетрис». Задание. Используя технологию XNA, написать игру «Тетрис» для Windows Phone 7. Допускается использование технологии Silverlight + XNA, если основная логика игры будет написана на XNA Framework. Требования. В работе необходимо обеспечить: управление уровнями сложности (изменение скорости игры); подсчет очков и сохранение лучшего результата; реализацию меню игры (при использовании технологии Silverlight + XNA) и звуковых эффектов; вывод на экран набранных очков и уровня сложности; создание собственного тайла.

По технологии Silverlight + XNA необходимо создать новый проект XNA Game Studio 4.0 – Windows Phone Silverlight and XNA Application [4].

Поскольку работы подразумевают создание полноценных игровых приложений с большим числом строк кода, в статье далее приведен только порядок действий без детализации программной логики: «Создадим абстрактный класс Figure, который будет содержать основные методы для будущих классов фигур». Перейдем к работе с библиотеками XNA: «В классе игры определим глобальные переменные типа Texture2D, SpriteFont и SoundEffect (для этого нужно использовать пространство имен Microsoft.Xna.Framework.Audio). Внутри метода OnNavigatedTo() будем загружать изображения, шрифты и звуковые эффекты из ресурсов. Там же будем получать параметр от страницы MainPage.xaml с номером уровня сложности (скорости игры)». Далее обучающимся следует ознакомиться с одним из основных методов XNA, в котором происходят игровые действия – OnUpdate(). Далее детализируем обработку касаний экрана: «В методе OnUpdate() происходит обновление объектов, пересчет положений и обработка нажатий. Внутри данного метода будем перехватывать нажатия. При одиночном нажатии будем перемещать фигуру просто на одну клетку. При задержанном – будем осуществлять быстрое перемещение. В функции StartGame() будем определять, какие типы жестов считать сенсором экрана (для этого нужно использовать пространство имен Microsoft.Xna.Framework.Input.Touch)». Далее познакомим обучающихся с еще одним основным методом XNA, в котором происходит вывод изображений на экран – OnDraw(): «Метод OnDraw() отвечает за перерисовку игрового пространства. Он вызывается каждый игровой такт. Внутри него с помощью уже определенной переменной spriteBatch будем рисовать фон, кнопки, игровое поле, кирпичики на поле, фигуры и строки. Всю прорисовку необ ходимо заключать в функциональные «скобки» spriteBatch.Begin() и spriteBatch.End()». Данная часть курсовой обучает работе с методом OnDraw() и двумерной графикой: рисованию изображений и текстовых строк на экране телефона с использованием библиотек XNA. В результате описанных действий будет разработано игровое приложение, показанное на рис. 2.



Рис. 2. Игровое приложение «Тетрис»

В конце приводится список возможных совершенствований: совершенствование анимации; добавление фигур «пентамино» (или других). Совершенствование анимации подразумевает самостоятельное изучение библиотек эффектов XNA.

В связи с этим можно сделать вывод, что методика обеспечивает обучение разработке приложений для мобильных устройств. Мобильные технологии активно развиваются. По предварительной информации от компании Microsoft, новая платформа не будет иметь кардинальных отличий от своего предшественника, т. е. в основе разработки, как и ранее, будут лежать технологии Silverlight и XNA [5]. В результате предложенная методика потребует минимальных изменений. Планируется расширение курса, предполагающее добавление лабораторных работ по следующим темам: информационная безопасность; управление службами; взаимодействие с облачными сервисами; взаимодействие с социальными сетями.

ЛЕКЦИЯ №9. КРОССПЛАТФОРМЕННЫЕ ПРИЛОЖЕНИЯ, ИХ ОБЛАСТИ ПРИМЕНЕНИЯ.

Кроссплатформенное приложение — это однократно написанное приложение для развертывания сразу на нескольких платформах. Идея кроссплатформенных решений не в удобстве для пользователя, а в оптимизации процессов разработки мобильного приложения и уменьшения затрат на разработку приложений.

Кроссплатформенная разработка подразумевает использование специальных утилит (фреймворков). Наиболее популярными утилитами для создания кроссплатформенных приложений являются:

- Xamarin;

- Delphi FireMonkey;
- Appcelerator Titanium;
- Kony Platform;
- Adobe PhoneGap;
- IBM Worklight;
- Telerik Platform;
- Verivo Akula;

Кроссплатформенный подход в разработке мобильных приложений имеет следующие положительные моменты:

- Требуется меньше ресурсов для реализации приложения сразу под несколько платформ. Дизайнер делает только один набор графики;
- Меньше времени затрачивается на разработку;
- Упрощенный цикл обновления продукта. Если необходимо что-то добавить или исправить какую-то ошибку, это делается сразу для всех платформ;
- Использование единой логики приложения. Заложенная логика в работу приложения будет работать гарантированно одинаково для всех платформ;
- Если у Вас есть сайт, то кроссплатформенная разработка—это идеальный вариант решить задачу для Вашего бизнеса в кратчайшие сроки;

Каждый пользователь приложения должен быть уверен в том, что каждый элемент управления, каждая иконка будут иметь стандартный вид и положение на экране приложения. Для разных платформ эти стандарты будут разными. Для каждой платформы написан свой гайдлайн. Гайдлайн—это ценные указания от компаний-производителей платформ (iOS, Google.Play) в адрес разработчиков мобильных приложений, направленные на то, чтобы подогнать их дизайн и функциональность под стандарты. Таким образом, если приложение будет написано по гайдлайну одной из платформ, то пользователям второй это принесет дискомфорт.

Кроссплатформенные приложения отлично справляются с разработкой игр и тестовых заданий. Современные игры пишутся в подавляющем большинстве на кроссплатформенных технологиях, что очень сильно ускоряет разработку без ущерба для качества, т.к. в этом случае используются специальные графические фреймворки. Если какой то проект нужно сделать быстро для проведения каких-либо тестов, опросов, анкетирования и при этом ситуация требует параллельной работы проекта именно на нескольких платформах одновременно, в этом случае кроссплатформенная реализация может быть оптимальным решением.

Кроссплатформенные приложения пишутся сразу для нескольких платформ на одном языке, отличном от нативного. Как такой код может работать на разных устройствах? Тут тоже есть два подхода.

Первый заключается в том, что на этапе подготовки приложения к публикации он превращается в нативный для определённой платформы с помощью транспилера. Фактически один кроссплатформенный язык программирования «переводится» на другой.

Второй — в том, что к получившемуся коду добавляется определённая обёртка, которая, работая уже на устройстве, на лету транслирует вызовы из неродного кода к родным функциям системы.

Предполагается, что большая часть такого кода может переноситься между платформами — очевидно, что, например, логика совершения покупок, сохранения товара в корзину, просчёта маршрута для такси, написания сообщения в мессенджер не меняется в зависимости от того, Android у клиента или iOS. Нужно лишь доработать UI и UX для платформ, но сейчас, в определённых пределах, даже это можно объединить — например, меню-гамбургер активно используется как на Android, так и на iOS. Так что даже внесений исправления в интерфейс для того, чтобы приложение отвечало духу и букве нужной платформы — вопрос желания, необходимой скорости и качества разработки.

Преимущества:

стоимость и скорость разработки. Так как кода надо писать заметно меньше, то и стоимость работ снижается;

возможность использовать внутренние ресурсы компании. Как мы покажем дальше, разработку кроссплатформенных приложений зачастую можно осуществить силами уже существующих у вас программистов.

Недостатки:

неродной интерфейс или, как минимум, необходимость работы с интерфейсом каждой платформы отдельно. У каждой системы свои требования к дизайну элементов и иногда они взаимоисключающие. При разработке это необходимо учитывать;

проблемы в реализации сложных функций или возможные проблемы работы даже с простыми процедурами в силу ошибок самих фреймворков разработки. Кроссплатформенная среда лишь транслирует запросы к системным вызовам и интерфейсам в понимаемый ею, системой, формат, и потому на этом этапе возможны как сложности с пониманием, так и возникновение ошибок внутри самого фреймворка;

скорость работы. Так как кроссплатформенная среда является «надстройкой» над кодом (не всегда, но в определённых ситуациях), в ней возникают свои задержки и паузы в отработке действий пользователя и выводе на экран результатов. Это было особенно заметно несколько лет назад на смартфонах, более маломощных относительно сегодняшних, однако сейчас, с ростом производительности мобильных устройств, этим уже можно пренебречь.

Как видите, эти два метода практически являются зеркальным отражением друг друга — то, что плюсы у нативной разработки приложений, минусы у кроссплатформенной, и наоборот.

Популярные платформы и инструменты кроссплатформенной мобильной разработки

Как мы написали выше, есть два подхода — превращение кода в нативный на этапе сборки или добавление определённой обёртки, транслирующей вызовы к системе и от неё.

Cordova и PWA — два инструмента, работающие как раз в идеологии обёртки.

Cordova и HTML5

Одно из самых популярных направлений в кроссплатформенном программировании, которое часто по-народному называют PhoneGap. Фактически создаётся мобильный сайт, который «оборачивается» небольшим платформенным кодом, транслирующим вызовы от системы к приложению и обратно.

Все недостатки и достоинства тут выражены как нигде ярко. Вы можете использовать веб-разработчиков (HTML, CSS и JavaScript как основные технологии) и за месяц или даже пару недель сделать первую версию приложения за относительно небольшие деньги. Да, она будет подтормаживать в работе, возможно, в ней будет не совсем точная геолокация, но она будет работать на всех устройствах и позволит вам, как минимум, протестировать спрос со стороны клиентов на мобильных устройствах.

Для такого подхода создано огромное количество фреймворков, но все они делают фактически одно и то же. Различие между ними в том, что Cordova (PhoneGap) не задаёт ограничений и шаблонов на логику и UI для вашего HTML5-проекта, а фреймворки оперируют собственными готовыми UI-элементами, имитирующими мобильные платформы, и своей логикой разработки. В качестве примера такого подхода можно указать: Ionic Framework — обёртка; Framework7, Mobile Angular UI, Sencha Touch, Kendo UI — интерфейсные фреймворки.

PWA

Модная технология от Google — это те же самые веб-приложения, но за счёт использования определённых технологий (в первую очередь это так называемые Service Worker — работающие в фоновом режиме скрипты, и Web App Manifest — описание веб-приложения в понятном для мобильной системы виде) они без обёртки из PhoneGap могут работать как нативные. Они могут устанавливаться на домашний экран в обход магазина приложений, работать в офлайне, работать с пуш-уведомлениями, с нативными функциями.

Проблема в том, что не все платформы даже сейчас поддерживают эти «определённые технологии». В первую очередь это касается Apple, которой, видимо, очень не нравится возможность распространять приложения в обход App Store.

Учтя все недостатки HTML5-решений, многие компании создали инструменты, которые позволяют писать код на одном, не нативном, языке, а он потом транслируется в нативный. Так убивается два зайца одновременно: кодовая база получается одна, а приложения получаются максимально близки к нативному.

Xamarin

Платформа компании Microsoft. Используется стандартный для Enterprise-разработки язык программирования C#, кроссплатформенная

среда разработки — Visual Studio. На выходе — нативные приложения для iOS, Android и Windows. Правда, относительно большого размера.

React Native

Платформа от Facebook — приложения пишутся на JavaScript и с использованием CSS-подобных стилей. Интерфейс получается родной, а код интерпретируется уже на платформе, что придаёт ему нужную гибкость.

Будучи относительно молодой платформой, React Native пока очевидно (хоть и не катастрофически) страдает от недостатка средств разработки и документации.

Flutter

Естественно, не мог обойти тему кроссплатформенной разработки Android и iOS-приложений и такой гигант, как Google. Flutter, пока, правда, существующий только в бета-версии, исповедует отличный от React Native и Xamarin подход. Он не превращает исходный код в нативный, который выполняется платформой, а на самом деле рисует окно на экране смартфона и отрисовывает все элементы сам. В качестве языка используется «фирменный» Dart, который Google создал как усовершенствованную версию JavaScript.

У этого есть как преимущества (например, внешне идентичные интерфейсы), так и недостатки (например, перерисовка интерфейса требует определённых затрат памяти и процессорного времени).

Платформа быстро развивается и Google вкладывает в это много сил и средств. Но по сравнению с Flutter даже React Native кажется вполне устоявшейся и впечатляющей экосистемой.

Что выбрать

У вас уже наверняка пошла голова кругом, а понимания что выбрать, так и не появилось. Давайте представим простой список вопросов, который вам поможет:

должно хоть как-то работать на любом устройстве?
Выбирайте HTML как основу;

у вас достаточно средств, нет спешки и вы хотите самое качественное приложение? Вам прямой путь в нативную разработку;

у вас есть «встроенный» веб-разработчик или вы просто хотите быстро и просто попробовать мобильное приложение в деле? Тут можно рекомендовать Cordova/HTML или PWA;

у вас есть собственная CRM-система и поддерживающий её C#-разработчик? Берите Xamarin;

вы «хотите попробовать», но надо сделать всё красиво и модно? Смотрите в сторону React Native или Flutter.

Можно зайти и с другой стороны. Посмотрите на функциональность, которая вам потребуется в приложении, и исходите из этого:

простое приложение-визитка? Возьмите React Native или HTML5 и вы получите две платформы за минимальную цену;

у вас есть сайт с большой посещаемостью и вам нужно протестировать гипотезу присутствия в мобильном пространстве? HTML5;

сложные приложения с доступом к нужным функциям устройств? Нативная разработка, Xamarin, React Native.

Кроссплатформенная разработка — не панацея

При выборе нужно исходить из поставленных задач и существующих ресурсов. Кроссплатформенная разработка — хорошее и понятное направление, но со своими преимуществами и недостатками, которые нужно иметь в виду ещё до запуска проекта. Сделанное кроссплатформенное приложение очевидно лучше несделанного нативного. Вы можете быстро и дешево разработать его, загрузить в магазин и просто проверить спрос со стороны пользователей — ищет ли кто приложение от вас, устанавливает ли, какие функции использует. По результатам такого эксперимента можно будет решать судьбу мобильного направления в вашей компании и инвестиций в него.

ЛЕКЦИЯ №10-№11. НАТИВНЫЕ ПРИЛОЖЕНИЯ, ИХ ОБЛАСТИ ПРИМЕНЕНИЯ.

Под нативным мы подразумеваем мобильное приложение, которое создается для определенной платформы и непосредственно устанавливается на устройство пользователя (занимая определенный объем памяти). Такие приложения пользователь загружает через магазин приложений той или иной платформы, такой как Play Store для Google и Apple App Store для iOS.

С нативными приложениями компании могут изготовить приложение согласно индивидуальным запросам, чтобы затем пользователю было удобно им пользоваться, в дополнение к веб-сайту или другому каналу, которым он уже привык пользоваться. Эта целостность и является существенным преимуществом нативных приложений.

Некоторые другие важные преимущества нативных приложений:

- - Обозначение геолокации позволяет компаниям подстраивать свои программы лояльности или промоакции. Потребители могут получать уведомления, когда они находятся возле физических магазинов, или имеют возможность получить региональную скидку.
- - Данные действий (или бездействия) пользователя могут быть легко собраны и проанализированы, таким образом облегчая оценку эффективности всего приложения либо его отдельных функций.
- - Нативные приложения, как правило, работают и «чувствуются» лучше. Веб-приложения иногда создаются для имитации нативных, но они ограничиваются скоростью интернета и возможностями дизайна.

И возможные недостатки:

- - Нативные приложения зачастую более дорогие в разработке, особенно для компаний, которым нужны приложения на кроссплатформенных ОС
- - Нативные приложения должны быть одобрены каждым магазином приложений, а процесс привлечения внимания к нему пользователей может быть сложным (если это не приложение для внутреннего пользования в компании).

Чем натив отличается от гибрида?

Помимо натива, существует еще один тип проектирования. Он не такой ресурсозатратный и более универсальный, однако имеет определенные недостатки. Речь идет о таком альтернативном кроссплатформенном решении, как гибридные мобильные приложения, которые создаются сразу для обеих платформ.

7 фактов о гибридных mobile-продуктах:

1. пишутся сразу под две операционные системы
2. загружаются одновременно в маркеты AppStore и Google Play
3. multifункциональны и работоспособны
4. один способ написания - не требуется несколько языков
5. высокая скорость проектирования и визуальная схожесть продукта с нативом
6. некорректная работа при отсутствии подключения к интернету
7. минимализм в отношении визуализации и анимации - не подходит для игр.

При этом, нативные приложения, разработка которых необходима отдельно под IOS и Андроид, стоят гораздо дороже, чем цена единого кроссплатформенного решения. Желая внедрить мобайл в свой бизнес, стоит учитывать, что приложения-гибриды имеют среднюю скорость работы на фоне натива. Но, если необходимо значительно сэкономить бюджет и оперативно спроектировать два готовых бизнес-решения, стоит сделать выбор в сторону hybrid. Яркий пример успешного кроссплатформенного решения - Instagram, о популярности которого говорить не приходится.

Окончательный выбор можно сделать, понимая какие цели вы ставите и каких результатов ждете. Но, если хотите быть на гребне волны и стремитесь к мировым трендам - не упустите возможность воспользоваться преимуществами мобайла.

Разработка нативных мобильных приложений выделяется спецификой своей разработки. А также готово «похвастаться» рядом ключевых преимуществ:

- при правильной разработке можно добиться высокой скорости работы и достаточно производительность, что увеличивает спрос на использование со стороны пользователей;
- на высоком уровне безопасность использование подобных программных продуктов;
- со стороны разработчиков взимается достаточно высокая оплата за реализацию поставляемого проекта;
- предусматривается широкий спектр функциональности;
- можно использовать без подключения к сети и выхода в Интернет;
- для конечного пользователя получается удобное и комфортное в использование приложение, которое однозначно будет пользоваться высоким спросом.

Перед решением подготавливать в своих целях нативное мобильное приложение стоит обратить внимание на несколько негативных нюансов:

- низкий охват платформ;
- для разработки потребуется немало времени и финансов;
- косметические цели сопровождаются необходимостью регулярного проведения обновлений;
- несовместимость с другими операционными системами.
-

ЛЕКЦИЯ №12. ГИБРИДНЫЕ ПРИЛОЖЕНИЯ, ИХ ОБЛАСТИ ПРИМЕНЕНИЯ.

В последнее время в связи с ростом популярности концепции Web 2.0 все чаще можно встретить упоминание о гибридных приложениях. Под неофициальным термином «гибридное приложение» (mashup application) принято понимать интернет-приложение, интегрирующее данные из разных источников в Интернете и обеспечивающее пользователю возможность интерактивного взаимодействия с предоставляемой информацией.

На данный момент самыми распространенными являются картографические гибридные приложения. Они предоставляют пользователю информацию о географическом размещении каких-либо объектов. Такие приложения объединяют в себе информацию, полученную как из баз данных, так и с помощью картографических функций каких-либо сервисов (например, Google Maps). Таким образом, пользователь может получить информацию о размещении интересующего его объекта в графическом виде, а также интерактивно изменять требования к типу отображаемой информации, то есть пользователю будет предоставлена возможность работать только с теми данными, которые ему необходимы, и он будет освобожден от избыточной информации. Существуют разные типы гибридных приложений, о которых речь пойдет далее.

В данной статье мы рассмотрим технологии, используемые при создании гибридных приложений, проанализируем возможности применения семантических технологий для расширения их функциональности, а также предложим перспективные возможности использования гибридных приложений.

Типы гибридных приложений

Существующие в настоящий момент гибридные приложения можно разделить на следующие типы:

- картографические приложения;
- видео- и фотоприложения;
- поисковые и торговые приложения;
- новостные приложения.

Картографические гибридные приложения — это приложения, которые объединяют в себе информацию о географическом расположении какого-либо объекта с другой информацией, описывающей данный объект. Информацию о размещении объекта гораздо удобнее воспринимать, если она представлена графически — в виде карты.

Стартом для развития технологий создания такого рода приложений стало появление Google Map API (Application Programming Interface — программный интерфейс приложения), предоставляющего разработчикам возможность использования карт Google на своих веб-сайтах. Он позволил привязывать к карте практически любой объект.

Гибридные видео- и фотоприложения — это приложения, позволяющие объединить с какой-либо фотографией информацию о ее авторе, лицах, находящихся на ней, месте съемки и т.д. В настоящее время данный тип гибридных приложений широко используется в социальных сервисах.

Поисковые и торговые гибридные приложения — приложения, объединяющие в себе информацию об одном и том же объекте, но полученную из разных источников. Это дает пользователю возможность сравнивать информацию по товарам, если речь идет о торговых приложениях, и получать полный объем новостей об одном и том же событии.

Архитектура гибридных приложений

Гибридные приложения состоят из трех основных элементов:

браузер-клиента;

провайдер-контента;

mashup-сайта.

Клиентский браузер является средой взаимодействия пользователя с приложением, а также интерпретирует графическое содержимое. Через браузер происходит сбор информации о требованиях к содержанию веб-страницы, которую просматривает пользователь.

Провайдером контента является сторонний сервис, предоставляющий данные для гибридного приложения. Число провайдеров контента, предоставляющих доступ к данным через API, медленно растет, но все же большинство информации приходится получать при помощи специальных утилит, проводящих синтаксический анализ содержимого страницы.

Mashup-приложение является интеграционным стержнем, объединяющим информацию из разных источников. Существует два подхода к реализации данного веб-сервиса:

создание веб-приложения, размещенного и выполняющегося на стороне сервера. При этом могут быть использованы такие технологии, как ASP, Java, PHP и др.;

генерация содержимого полностью на стороне клиента. При этом применяются сценарии исполнения, реализованные, например, при помощи JavaScript.

Использование каждого из этих подходов в чистом виде не очень удобно. Например, при реализации серверного варианта пользователь столкнется с проблемой частого обращения к серверу для обновления информации на странице. Кроме того, при применении приложения, выполняющегося на стороне клиента, будет проблематично обрабатывать сложные запросы, адресованные сразу к нескольким источникам данных.

Как правило, на практике эти два подхода объединяются: сбор данных выполняется на сервере, а их компоновка и представление — в клиентском

приложении. Это позволяет не запрашивать страницу заново, а просто обновлять какую-то ее часть, что положительно отражается на скорости работы приложения.

Приложения такого типа в значительной мере ориентированы на пользователя и называются полнофункциональными веб-приложениями (Rich Internet Application, RIA). Они являются следующим поколением сервисов, именуемых сервисами Web 2.0.

Технологические средства разработки гибридных приложений

При разработке гибридных приложений применяются различные технологии.

Технология Asynchronous JavaScript and XML (AJAX) позволяет отделить взаимодействие пользователя с веб-сайтом от взаимодействия веб-браузера с сервером. В сообществе разработчиков AJAX считаются скорее моделью веб-приложений, а не технологией. AJAX основан на DHTML (Dynamic Hypertext Markup Language) и включает следующие технологии:

CSS (Cascaded Style Sheet) — используется для оформления стиля страниц;

API модели DOM (Document Object Model) — применяется для динамического отображения документов и взаимодействия браузера с сервером;

XML — используется для асинхронного обмена данными;

JavaScript — применяется для выполнения клиентских сценариев. На данный момент JavaScript поддерживается всеми браузерами.

Использование этих технологий дает возможность обмениваться незначительными объемами данных без полной перезагрузки страницы, что позволяет оперативно реагировать на действия пользователя.

Платформонезависимые веб-протоколы REST (Representational State Transfer) и SOAP (Service-Oriented Access Protocol) применяются для взаимодействия с удаленными веб-сервисами. Взаимодействие по протоколу SOAP происходит при помощи обмена XML-сообщениями. Программные интерфейсы веб-сервиса, с которым идет взаимодействие, описаны в формате WSDL (Web Services Description Language).

REST — протокол передачи состояния для интерпретации и представления. Основным понятием в нем является понятие ресурсов.

RDF (Resource Description Framework) предоставляет набор методик для описания данных.

Повышение интеллектуальности бизнеса с Web 2.0

Web 2.0 и архитектура, ориентированная на сервисы (Service-Oriented Architecture, SOA), становятся новым способом производства услуг. Эти технологии предлагают измененный подход к тому, как вводятся новшества, принимаются решения, ведется сотрудничество, привлекаются новые клиенты.

Данные технологии предполагают вовлечение средств бизнес-аналитики. Традиционно доступ к внутренней информации возможен разными ограниченными путями: через системы ERP (Enterprise Resource Planning) или

MIS (Management Information System). С использованием технологий Web 2.0 данные теперь доступны не только в рамках одной организации, но и через Интернет. В этом контексте термин BI (Business Intelligence) может быть интерпретирован как «совместный интеллект». Под данным термином понимается объединение нового типа представления данных с традиционным подходом предприятия к созданию этих данных, что позволяет использовать эти знания для достижения успеха на рынке.

Варианты применения технологии гибридных приложений

Использование гибридных приложений, осуществляющих интерактивное представление информации, поступающей из разных источников, вполне оправданно во многих областях.

Применение данного типа приложений возможно, к примеру, в системах управления цепочками поставок (Supply Chain Management, SCM), в сервисах предоставления данных и ведения рейтинга подобных сервисов, а также во многих других решениях.

Использование гибридных приложений в системах SCM

Возможности гибридных приложений можно применять для реализации системы уровня B2B (Business to Business). Основная идея данного предложения заключается в выполнении следующих этапов:

Сбор информации от поставщиков продукции.

Представление данных в единой форме.

Сравнение и анализ предоставляемых данных.

Параметрический отбор поставщиков.

Графически данный подход может быть представлен следующим образом (см. рисунок).

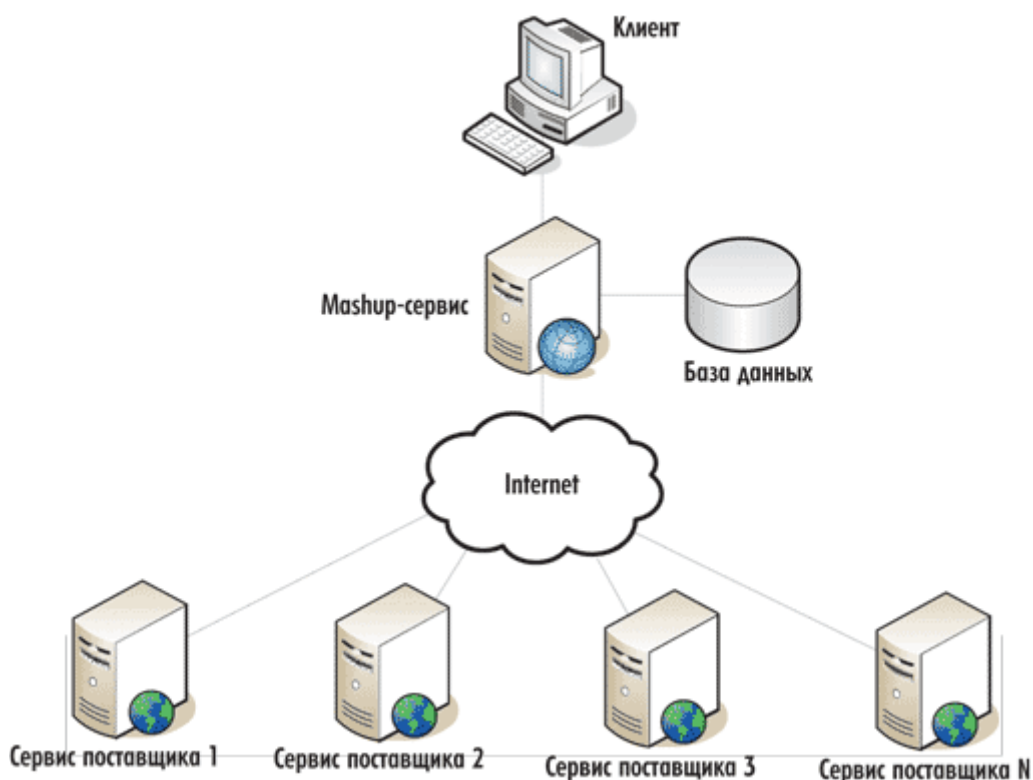


Схема получения информации для системы SCM с использованием гибридных приложений

Ставшая в последнее время популярной архитектура SOA позволяет инкапсулировать определенный функционал и бизнес-логику в сервисы. Предположим, что поставщик продукции реализует веб-сервис, осуществляющий предоставление информации по продукции (количество, стоимость, скорость доставки, стоимость доставки и др.). Предлагаемое гибридное приложение будет заниматься сбором этой информации и поддержанием ее в актуальном состоянии (то есть периодически обновлять данные, синхронизируя их с сервисом поставщика).

Для эффективного использования mashup-сервис необходимо снабдить функциями сравнения предлагаемой продукции (сырья) по определенным параметрам, заданным конечным пользователем, предоставить пользователю возможность изменять набор сервисов, информация которых ему необходима, а также реализовать кэширование данных для увеличения скорости предоставления информации. Далее функциональность можно наращивать, дополняя необходимыми методами обработки и представления данных.

Клиентом в данной ситуации может являться, например, менеджер по закупкам. Пользуясь предложенным гибридным приложением, он сможет в зависимости от ситуации выбрать того или иного поставщика, опираясь на функциональность данного приложения. Например, при высокой значимости фактора времени предпочтение будет отдаваться поставщикам с наименьшим временем доставки продукции. При наличии большего количества времени предпочтение будет отдаваться поставщикам, стоимость сырья и доставки у которых ниже.

Данные о сочетаниях параметров могут быть сохранены и приняты во внимание при следующем запросе информации. Таким образом, скорость обмена информацией и заказа сырья увеличится за счет быстрого доступа к данным о продукции в интерактивном режиме. Позитивным эффектом от использования такой архитектуры является существенное сокращение операционных издержек.

Данная система может быть адаптирована к схеме B2C (Business to Customer), если контент-провайдерами будут служить сервисы, предоставляющие услуги потребителям, например магазины.

Информационный веб-сервис с ведением рейтинга

Данный веб-сервис может быть разработан для предоставления информации, собранной с различных сервисов, в обобщенном виде, а также в режиме выбранных предпочтений.

Предположим, что в виде веб-сервисов реализованы услуги, предоставляемые агентствами недвижимости. Разработанный mashup-сервис осуществляет сбор информации по недвижимости с этих веб-сервисов и предоставляет возможность в интерактивном режиме просматривать и сравнивать данные, полученные с различных ресурсов.

Рейтинг сервисов можно производить по разным показателям: частота обращений, доступность (время нахождения сервиса в режиме доступности,

онлайн) и пр. Рейтингование нужно проводить для того, чтобы пользователь мог сравнить позиции разных сервисов по различным параметрам и выбрать тот, который ему больше подходит.

В дальнейшем можно расширить сферу предоставляемой информации и добавить новые сервисы. Бизнес-выгода от такого проекта при его успехе будет очевидна, например если взимается определенная плата за учетную запись в системе — для пользователей и определенная плата за нахождение сервиса в списке — для поставщиков услуг.

Заключение

По мере все более широкого внедрения SOA в компаниях можно говорить о том, что со временем сервисы станут более веб-ориентированными. Можно предположить, что вскоре внедрение SOA будет подразумевать обеспечение сервисов веб-интерфейсами для обмена и предоставления информации другим сервисам. Обеспечение веб-сервисов функциями, присущими приложениям Web 2.0, позволит бизнесу быстрее адаптироваться к меняющимся требованиям рынка и позволит более гибко управлять бизнес-процессами компании.

В данном контексте гибридные приложения займут хоть и не лидирующее положение, но прочно укрепят свои позиции как веб-приложения, позволяющие по запросу предоставлять информацию по услугам из различных источников, которыми будут являться сервисы сторонних компаний.

ЛЕКЦИЯ №13-№14. ОСНОВНЫЕ ЯЗЫКИ ДЛЯ РАЗРАБОТКИ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ.

Если Вы планируете или уже решили стать мобильным разработчиком. Первое с чем Вам придется столкнуться, это выбор платформы и языка на котором Вы планируете писать программы, но для начала, давайте узнаем на каких языках программирования пишут приложения.

Какие есть языки программирования и как их выбрать?

От того какую платформу разработки вы выберете, будет зависеть то, какой язык Вы будете изучать. Чуть ниже приведен список всех языков для iOS и Android, перечислены их преимущества и недостатки, для того, чтобы Вы могли оценить каждый из них, и выбрать наиболее подходящий для Вас.

Однако существуют и исключения. Существуют кроссплатформенные языки, которые позволяют писать код как для приложений на Android, так и iOS.

Как недостаток можно выделить то, что программы написанные на подобных языках будут значительно менее быстро работать и снизят продуктивность работы с системой, в отличии от программ написанных под определенную операционную систему.

На каком языке пишут приложения для Android?

В первую очередь ответим на вопрос о том, какие языки программирования для приложений Android подходят больше всего:

Java;
HTML5 / JavaScript;
C#;
Kotlin;
C / C++;
BASIC;
Python.

Разработчики Android на данный момент более востребованы в мире IT, в связи с тем что смартфоны на базе Android на сегодняшний день занимают две трети рынка мобильных устройств.

Обучение ниже перечисленным языкам позволит вам начать писать программный код для игр, приложений и программ. А для того чтобы вам было проще выбрать, мы составили перечень их преимуществ и недостатков.

Java

Любой, кто имеет дело с разработкой приложений на Android, ответит вам, что самый популярный язык программирования в их сфере — это Java. Данный язык является официальным языком разработки на Android, то есть он имеет наибольшую поддержку со стороны Google и большинство приложений в Google Play написаны именно на нем.

Самый простой способ начать разрабатывать приложения — скачать Android Studio.

Android Studio — это интегрированная среда разработки (IDE), разработанная компанией Google. В комплекте с этой IDE идет Android SDK (набор инструментов для упрощения и ускорения разработки приложений), что позволит вам, без лишних усилий, сразу же сесть и начать работать.

С помощью официальной документации от Google, вы быстро разберетесь, как работать с Android Studio. На том же сайте вы найдете огромное число библиотек для улучшения ваших приложений и подробные инструкции по использованию той или иной функции.

Сам язык Java был выпущен компанией Sun Microsystems в далеком 1995 году. Код Java можно выполнить на любом устройстве, так как сначала он транслируется в специальный байт-код, независимый от платформы, а затем этот байт-код выполняется виртуальной машиной JVM.

К сожалению, Java довольно сложен, поэтому я бы не рекомендовал его, в качестве “первого языка для изучения”. Java — это объектно-ориентированный язык, включающий в себя такие сложные темы, как конструкторы, NullPointerException, проверяемое исключение и т.д. Вам придется использовать очень много шаблонного кода (boilerplate code), который занимает ненужное место и отвлекает внимание, и все ради выполнения одной минимальной задачи. Также, разработка с помощью Java потребует от вас базового понимания таких понятий, как Gradle, манифест приложения и язык разметки XML.

Но все вышеперечисленное вовсе не означает, что Java — это плохой язык. Вообще нельзя называть, какой бы то ни было, язык плохим. Все недостатки Java существуют для нашего же с вами блага и способствуют

написанию чистого кода. Именно по этой причине многие люди так любят Java, а еще потому что он очень универсален и используется по всему миру. Согласно рейтингу PYPL (Popularity of Programming Languages), Java является самым востребованным языком программирования среди работодателей.

Android Studio, который с каждым годом становится все лучше и лучше, доставляет настоящее удовольствие при разработке приложений. Такие функции, как визуальный редактор и всплывающие подсказки делают процесс работы более плавным, а расширенные и мощные функции, регулярно добавляемые в IDE, предоставляют разработчикам доступ к таким вещам, как простое в использовании облачное хранилище и т.д.

Каков вердикт? Для тех, кто хочет испытать полное погружение в Android-разработку, Java — идеальный вариант для начала. Ну а для тех, кого отталкивает написание сложного и объемного кода, существует визуальный редактор и шаблонный код. Если вы новичок и хотите создать игру, либо просто хотите создать несколько проектов с нуля для портфолио и профессионального опыта — я советую вам начать с чего-нибудь попроще и вернуться к Java и Android Studio, когда вы будете более уверенно чувствовать себя в мире мобильной разработки.

Примечание: Java также можно использовать с Unity. Я расскажу о движке Unity в разделе C#. Просто знайте, что можно выучить Java, вместо C# и использовать с Unity, а затем перейти к разработке, с помощью Android Studio.

В случае мобильного программирования данный язык, поддерживается фреймворком Android Studio. Он является типом программного обеспечения, т.н. средой разработки. Ознакомиться с работой в которой вам предстоит в любом случае если вы решили программировать под Android. В этом вам так же поможет набор инструментов Android SDK.

Благодаря тому, что руководства, платные и бесплатные библиотеки Java найти не сложно, имея базовый набор знаний, создание программ на этом языке не составит особого труда.

Плюсы:

Основа всех языков мобильного программирования, обучение которому упростит последующее изучение языков;

Поддерживается Android Studio и набором инструментов Android SDK.

Минусы:

Сложный в освоении.

HTML5 / JavaScript

Чаще всего HTML, CSS и JavaScript используется для разработки веб-приложений. Однако есть и случаи исключения, благодаря специальным средам разработки таким как PhoneGap, Adobe Cordova, React Native, Ionic, Sencha программисты которые пишут код на HTML, CSS и JavaScript смогут создавать кроссплатформенные программы.

Плюсы:

Позволит создавать кроссплатформенные приложения;

Позволит создавать веб-приложения;

При должном наборе знаний, дает возможность верстать сайты.

Минусы:

Нативные приложения получатся менее качественными, т.к. изначально они не были рассчитаны на нативную разработку.

C#

C# — это более простой, более удобный объектно-ориентированный аналог языков C и C++, разработанный компанией Microsoft. Он сочетает в себе мощь C++, удобство Visual Basic, а также многие особенности синтаксиса Java. Как и в Java, в C# реализована функция сборки мусора (garbage collection), которая периодически освобождает память, удаляя объекты, не востребованные приложениями, поэтому вам не придется беспокоиться о таких вещах, как утечка и освобождение памяти.

Если вы хотите быстро и без особых усилий начать разработку приложений под Android — я рекомендую сочетание C# и Unity. Unity — это и игровой движок, и IDE. Unity — это бесплатный инструмент, который позволяет за считанные часы и несколько строк кода создать полноценную игру. Без преувеличения. Помимо легкости в использовании, Unity очень мощный движок, именно поэтому его используют многие игровые студии в Google Play Store. Кстати, забыл упомянуть про его кроссплатформенность. Помимо всего прочего, разработка приложений, с помощью C# и Unity, поможет вам лучше понять объектно-ориентированное программирование (поскольку “объекты” на самом деле являются объектами большую часть времени).

Каковы же минусы? Unity — идеальный вариант для разработки игр, но для создания стандартных Android-приложений, типа калькулятора, плохо подходит, особенно если вы собираетесь соответствовать материальному дизайну (material design) от Google. Если вы планируете стать профессиональным Android-разработчиком, вам лучше на время забыть про C# и Unity, так как это ограничит ваши возможности для трудоустройства, однако если ваша цель стать именно разработчиком игр — C# и Unity ваш путь в будущее.

Не заинтересованы в Unity? Тогда вам стоит рассмотреть Unreal Engine (графика в разы лучше, чем у Unity, но данный движок мало подходит для разработки мобильных игр) или же донельзя упрощенный движок GameMaker Studio.

C# также можно использовать, при помощи Xamarin через Visual Studio. Данный способ очень похож на традиционный способ разработки Android-приложений (при помощи Android Studio), но с добавлением функции кроссплатформенности (один и тот же исходный код для Android и iOS). C# и Xamarin — бессмысленный вариант для новичков в сфере разработки приложений под Android, но для небольшой компании, желающей создать приложение и для iOS, и для Android — это имеет смысл, поскольку большое количество информации в Интернете по тем или иным вопросам, а также сплоченное комьюнити поможет в этом нелегком деле.

Считается своего рода улучшенной версией Java с исправленными недостатками и более широким и удобным функционалом.

Отлично сочетается с моделированием в Unity 3D, делая ваши возможности в мобильной разработке практически безграничными позволяя создавать не только программы, но и игры.

Благодаря функции сборки мусора (garbage collection) появляется возможность удаления объектов не востребованных в приложении.

Плюсы:

Улучшенная версия Java;

В сочетании с Unity 3D позволяет создавать игры;

Минусы:

Для работы с ним необходимо обязательно знать Java.

Kotlin

Отличный язык, который так же как и C# в сочетании с Java становится еще лучше, основным же его преимуществом является полное отсутствие недостатков что позволяет создавать программы под Android практически беспрепятственно.

Компиляция в байт-код JVM, как и у Java, дает Вам возможность перенести написанный ранее код на Java в Kotlin и убрать шаблонный код «boilerplate code» что сделает синтаксис более читабельным.

Официально он был представлен компанией Google в 2017 году, позиционируя себя как следующий официальный язык написания приложения под Android после Java.

В связи с его относительно недавним появлением, на данный момент очень востребован т.к. разработчиков хорошо знакомых с Kotlin не так уже и много. Благодаря чему, найти работу при должном знании Kotlin Вам не составит труда.

Обучение ему займет у Вас не так много времени как изучение Java, и будет гораздо более простым, однако для его понимания знание Java опять таки является обязательным.

Плюсы:

Востребованный среди программистов;

Более прост в освоении чем Java;

Компиляция в JVM;

Совместим с Android Studio.

Минусы:

Обязательно знание Java.

C / C++

Предоставляя SDK и NDK среды разработки в Android Studio, Google дает возможность разработчикам на C и C++ создавать нативные мобильные приложения. Однако они не приспособлены к полноценной разработке приложений, а служат лишь для создания библиотеки, которую в дальнейшем с помощью Java можно будет подвязать к основному коду, что значительно сократит время написания кода.

Код написанный на C и C++ запускается через само мобильное устройство, что дает больше контроля такими элементами как сенсоры, жесты, память и прочие.

Сами по себе достаточно не просты в изучении и применении, в связи с тем, что изначально не были предназначены для разработки именно мобильных приложений.

Плюсы:

В связке с Java позволяет сократить код, что ускорит работоспособность программы.

Минусы:

Непростой в освоении;

Не позволяет полноценно создать приложение, а всего лишь подвязывать библиотеки к приложению.

BASIC

Отличный вариант для начинающих программистов в силу своей простоты. Однако в связи с тем что он обладает весьма ограниченным функционалом, сложные программы для устройств на базе Android на нем создать не получится. Писать на нем можно только в специальном фреймворке B4A, с Android Studio к сожалению, он не совместим.

Плюсы:

Прост в освоении.

Минусы:

Не совместим с Android Studio, Unity, и Xamarin;

Весьма ограниченный функционал;

Использование B4A не может быть бесплатным.

Python

Несмотря на то, что Android не поддерживает Python, его использование не представляется исключением для создания Android-приложений. Так как с помощью фреймворка Kivy можно скомпилировать код написанный на чистом Python в практически любой требуемый формат.

Наличие большого количества различных библиотек позволяет строить на нем нативные интерфейсы, что значительно улучшает функциональность приложений.

Плюсы:

Подходит как для нативных, так и для веб-приложений;

Позволяет строить нативные интерфейсы;

Легко читаемый синтаксис;

Прост в изучении.

Минусы:

Не официальный язык Android и не поддерживает его без фреймворка Kivy;

Не особо востребованный.

На каком языке пишут приложения для iOS?

На Android и iOS код пишется по-разному и соответственно на разных языках, но разница на самом деле не столь существенная. Давайте разберем список основных языков на которых пишут приложения для iOS:

Swift

C#

Objective-C

C++

Количество разработчиков, которые знают языки программирования iOS приложений значительно меньше чем тех кто умеет писать под Android, что делает их более востребованными на рынке труда.

Swift

Крис Латтнер, главный автор LLVM, вдохновившись Objective-C, Rust, Ruby, C#, Haskell и другими языками, в 2010 году начал разработку Swift, а в 2014-м уже выпустил первую версию.

Swift изначально заточен под операционные системы Apple, поэтому он производительнее и функциональнее, чем Objective-C, для работы с которым использовалась подключаемая runtime-библиотека. В 2015 году вышла версия 2.0, которая была еще быстрее, получила API для работы с ошибками и проверку доступности функций для целевых ОС. Актуальная версия на момент написания статьи — 4.2.

Синтаксис изменился, и теперь программы больше похожи на код C-подобных языков — с той лишь разницей, что после операций не ставится символ «;» (как тебе такое, ошибка компилятора?):

```
class Rect {  
var width: Int = 0  
var height: Int = 0  
}
```

Обратите внимание, что инициализатора нет, так же как и методов для получения или присваивания значения переменным. Чтобы создать объект класса Rect, достаточно написать такой код:

```
var rect1: Rect = Rect()  
rect1.width = 5  
rect1.height = 5  
print("Ширина: \(rect1.width); Высота: \(rect1.height)")
```

Обладает весьма удобным функционал, который делает разработку приложений для iOS простым и увлекательным процессом, среди которого можно отметить множественное возвращение значений, замыкания и дженерики. Остается наиболее востребованным языком разработки приложений для iOS даже в 2019 году т.к. большинство iOS разработчиков, предпочитают именно его.

Плюсы:

Упрощенная навигация;

Прост в изучении;

Легко читаемый синтаксис;

Взаимодействие с Objective-C;

Поддержка динамических библиотек.

Минусы:

Частые обновления требуют регулярного изучения нового материала;
«Мост» позволяющий взаимодействовать с Objective-C замедляет процесс сборки программы.

C#

С его помощью можно писать не только для Android, но и для iOS, что позволяет создавать кроссплатформенные приложения.

Плюсы:

В сочетании с Unity 3D позволяет создавать игры;

Минусы:

Для его изучения необходимо обязательно знать Java.

В связи с тем, что C# является кроссплатформенным, приложения на нем будут не столь оптимизированными, как приложения написанные на официальных языках Apple.

Objective-C

Objective-C

В 1980 году Брэд Кокс в своей компании Stepstone создал Objective-C — позже он стал основным языком разработки для устройств Apple. Objective-C — это компилируемый объектно-ориентированный язык программирования (ООП), основанный на C.

С помощью объектов решается проблема повторяемости кода. Но делается это не так топорно, как при структурном программировании, в котором увеличение объема и сложности программы снижает качество.

Основное отличие Objective-C от C++ (тоже поддерживает ООП и тоже был разработан в восьмидесятых годах) в том, что вызов метода в нем интерпретируется не как функция, а как сообщение — как в языке Smalltalk. Такие сообщения могут содержать что угодно, а отправить их можно куда потребуется.

Также объект может переслать часть полученного сообщения другому объекту (даже если он находится не на этом компьютере), что позволяет делегировать задачи — по такому же принципу работают нейросети.

Objective-C — компилируемый язык. Однако сообщения привязываются к функциям исключительно во время выполнения программы, что позволяет создавать динамичные приложения. Еще в языке поддерживаются протоколы и метайнформация.

Синтаксис здесь значительно отличается от синтаксиса C-подобных языков. Для сравнения:

вызов метода объекта в C#:

Object.Method(argument);

а это — отправка сообщения объекту на Objective-C:

[object method:argument];

Разница видна и в создании классов.

Objective-C:

```

#import "Rect.h" //Подключение библиотеки
@implementation Rect //Объявление класса
+ newRect //Инициализатор
{
Rect * rect = [[Rect alloc] init];
[rect setWidth: 1.0f];
[rect setHeight: 1.0f];
return rect;
}
//Методы, возвращающие значение переменной
- (float) width
{
return width;
}
- (float) height
{
return height;
}
//Методы, задающие значение
- (void) setWidth: (float) theWidth
{
width = theWidth;
}
- (void) setHeight: (float) theHeight
{
height = theHeight;
}
@end //Конец объявления

```

И пример на C#:

```

public class Rect //Объявление класса
{
private float width; //Создание переменных
private float height;
public Rect(int width, int height) //Инициализатор
{
this.width = width;
this.height = height;
}
public int Width { get; set; } //Методы для вывода и присваивания
значений переменным
public int Height { get; set; }
}

```

К синтаксису Objective-C может быть сложно привыкнуть, если раньше вы писали на чем-то другом.

Плюсы:

Большое количество документации, которая упрощает изучение;
Совместимость со Swift.

Минусы:

Устаревший, более сложный в освоении чем Swift, в связи с чем в ближайшие несколько лет перестанет использоваться;

Сложный синтаксис;

Низкая производительность в сравнении с Swift;

C++

C++ позволяет программировать под iOS. Обработка больших объемов данных позволяет решать сложные логические задачи, модули и писать сложные алгоритмы.

Однако полностью написать на C++ приложение не удастся, из-за чего требует совместной работы с Objective-C.

Плюсы:

Поддерживает объективно-ориентированное программирование, процедурное программирование и обобщенное.

Улучшает производительность при работе с Objective-C.

Минусы:

Не предназначен для полноценной разработки;

Непростой в изучении.

Заключение

Как вы уже поняли, для разработки приложений можно использовать практически любой язык программирования, каждый из которых имеет свои преимущества и недостатки. Время потраченное на разработку мобильных приложений на каждом языке также может колебаться в достаточно широком диапазоне.

В этой статье мы собрали самые популярные языки для разработки программ на Android и iOS. Рассказав о том, какие из них больше подходят начинающим программистам, а какие меньше.

По мнению нашей редакции, правильным решением будет начать освоение языков мобильного программирования с Java, научившись работать с которым вам будет проще разобраться в остальных, т.к. Java является их основой.

Однако, помимо перечисленных языков программирования, вы можете осуществлять создание приложений в кроссплатформенном конструкторе мобильных приложений. Навыки разработки для работы в котором не являются обязательными.

ЛЕКЦИЯ №15-№16. ИНСТРУМЕНТАРИЙ СРЕДЫ РАЗРАБОТКИ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ.

Прежде чем начать разрабатывать приложения под Android, рассмотрим существующие инструменты, подходящие для этих целей. Можно выделить необходимые инструменты, без которых разработка мобильных приложений под Android просто невозможна. С другой стороны, существует большое

количество вспомогательных систем, в какой-то мере упрощающих процесс разработки.

К обязательным инструментам относится AndroidSDK –набор средств программирования, который содержит инструменты, необходимые для создания, компиляции и сборки мобильного приложения. Рассмотрим кратко наиболее важные инструменты, входящие в состав Android SDK:

- SDK Manager - инструмент, позволяющий загрузить компоненты Android SDK. Показывает пакеты Android SDK и их статус: установлен (Installed), не установлен (Not Installed), доступны обновления (Update available).

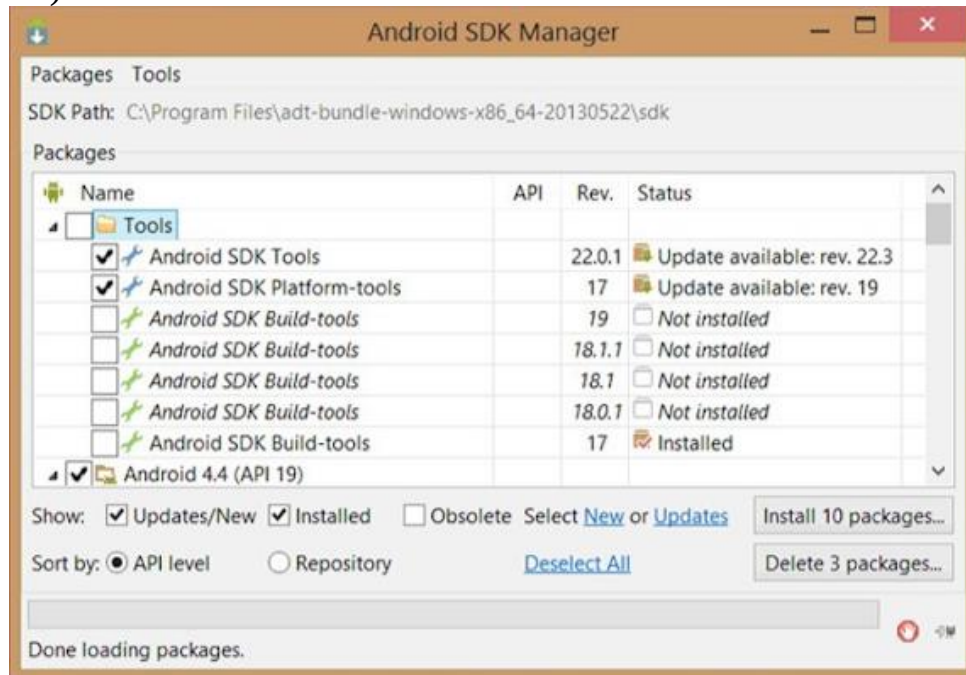


Рисунок 1. Android SDK Manager

Debug Monitor - самостоятельный инструмент, предоставляющий графический интерфейс к нескольким инструментам, предназначенным для анализа и отладки Android приложений:

- DDMS (Dalvik Debug Monitor Server) предоставляет услуги переброса портов, захват экрана устройства, информацию о потоках и динамической памяти устройства, вывод информации о действиях Android в реальном времени (logcat) и многое другое.

– HierarchyViewer позволяет отлаживать и оптимизировать пользовательский интерфейс Android приложения.

- Tracer for OpenGL ES - инструмент для анализа OpenGL|ES кода, используемого в мобильном приложении, позволяет захватывать команды OpenGL|ES и демонстрировать их по отдельным кадрам, что помогает понять как исполняются графические команды.

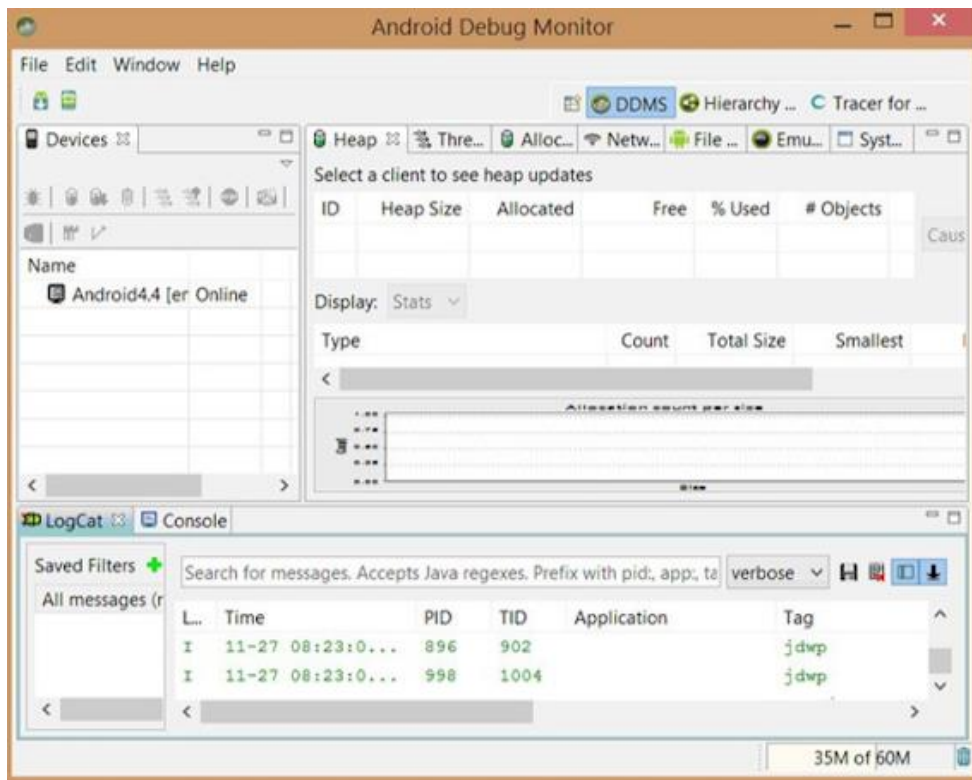


Рисунок 2. Окно инструмента Monitor

- Android Emulator (emulator) - виртуальное мобильное устройство, которое создается и работает на компьютере разработчика, используется для разработки и тестирования мобильных приложений без привлечения реальных устройств.
- AVD Manager - предоставляет графический интерфейс для создания виртуальных Android устройств (AVDs), предусмотренных Android Emulator, и управления ими. (В ЛР№1 подробно рассматривается создание и использование виртуального устройства).
- Android Debug Bridge (adb) - гибкий инструмент, позволяющий управлять состоянием эмулятора или реального Android устройства, подключенного к компьютеру. Также может использоваться для установки Android приложения(.apk файл) на реальное устройство.

Мы рассмотрели основные инструменты, входящие в состав Android SDK, разумеется, не все и недостаточно подробно. Для более серьезного изучения инструментов имеет смысл обратиться к сайту разработчиков здесь. Для разработки мобильных приложений под Android уверенного владения инструментами из SDK вполне достаточно. Если же возникают какие-то вопросы, дополнительные инструкции по созданию проектов, компиляции, запуску из командной строки содержатся в руководстве от Google здесь.

В современных условиях разработка ПО в большинстве случаев ведется с использованием интегрированных сред разработки (IDE). IDE имеют несомненные достоинства: процесс компиляции, сборки и запуска приложения обычно автоматизирован, в связи с чем для начинающего разработчика создать свое первое приложение труда не составляет. Но чтобы заниматься разработкой всерьез, необходимо потратить силы и время на изучение возможностей самой среды. Рассмотрим IDE, пригодные для

разработки под Android1. Для начала поговорим о двух средах разработки, которые рекомендует Google: Android IDE (ADT) и Android Studio.

Android IDE - среда разработки под Android, основанная на Eclipse. Предоставляет интегрированные инструменты для разработки, сборки и отладки мобильных приложений. В данном курсе Android IDE выбрана в качестве основной среды разработки. Возможности этой среды более подробно рассмотрены в первой лабораторной работе. Также там даны рекомендации по установке и настройке среды, созданию и запуску первого приложения как на эмуляторе, так и на реальном устройстве.

Android Studio-среда разработки под Android, основанная на IntelliJ IDEA. Подобно Android IDE, она предоставляет интегрированные инструменты для разработки и отладки. Дополнительно ко всем возможностям, ожидаемым от IntelliJ, в Android Studio реализованы:

- поддержка сборки приложения, основанной на Gradle;
- специфичный для Android рефакторинг и быстрое исправление дефектов;
- lint инструменты для поиска проблем с производительностью, с юзабилити, с совместимостью версий и других;
- возможности ProGuard (утилита для сокращения, оптимизации и обфускации кода) и подписи приложений;
- основанные на шаблонах мастера для создания общих Android конструкций и компонентов;
- WYSIWYG редактор, работающий на многих размерах экранов и разрешений, окно предварительного просмотра, показывающее запущенное приложение сразу на нескольких устройствах и в реальном времени;
- встроенная поддержка облачной платформы Google. Загрузить последнюю версию Android Studio, а также получить рекомендации по установке, настройке и началу работы можно здесь.

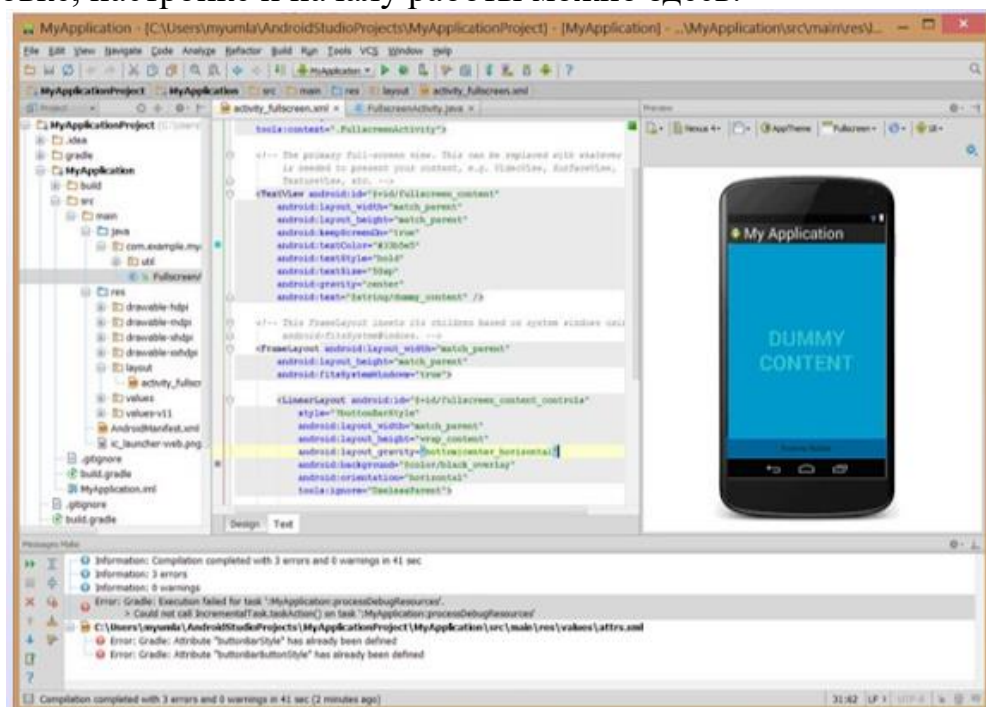


Рисунок 3. Среда разработки Android Studio

Перейдем к рассмотрению других инструментов, пригодных для разработки мобильных приложений под Android. Начнем с инструментов от Intel - Intel XDK и Intel Beacon Mountain.

Intel XDK позволяет легко разрабатывать кроссплатформенные мобильные приложения; включает в себя инструменты для создания, отладки и сборки ПО, а также эмулятор устройств; поддерживает разработку для Android, Apple iOS, Microsoft Windows8, Tizen; поддерживает языки разработки: HTML5 и JavaScript.

Последняя тема данного курса полностью посвящена изучению нового поколения инструментальных средств разработки мобильных HTML5 приложений и Intel XDK, предполагается разработка мобильного приложения с использованием этих инструментов.

Intel Beacon Mountain - среда разработки, позволяющая создавать приложения для устройств, работающих под управлением ОС Android. Предоставляет инструменты необходимые для проектирования, разработки, отладки и оптимизации приложений под Android. Освобождает разработчика от необходимости поддерживать систему разработки в актуальном состоянии, следит за обновлениями и добавляет их в среду разработки по мере появления. Поддерживает разработку для целевых платформ на основе процессоров Intel Atom и ARM. Beacon Mountain построена на основе Android IDE (Eclipse, Android ADT, Android SDK), для более серьезной разработки и оптимизации добавлены следующие инструменты Intel:

- Intel*HardwareAcceleratedExecutionManager(Intel*HAXM) - аппаратно поддерживаемый процессор виртуализации, использующий технологию виртуализации Intel* (Intel* VT) для ускорения работы эмулятора в среде разработки.

- Intel* Graphics Performance Analyzers (Intel* GPA) System Analyzer поддерживает мобильные устройства с процессором Intel Atom под управлением ОС Android. Позволяет разработчикам оптимизировать загрузенность системы при использовании процедур OpenGL, предоставляя возможность получать множество системных метрик в реальном времени, отображающих загрузенность CPU, GPU и OpenGL ES API. Разработчик может запустить несколько графических экспериментов для выявления узких мест в обработке графики.

- Intel* Integrated Performance Primitives (Intel*IPP) Preview - библиотека оптимизированной обработки данных и изображений, поддерживающая мобильные устройства с платформой Intel под управлением ОС Android. Preview версия является частью полной версии Intel IPP, которая тоже поддерживает ОС Android.

- Intel* Threading Building Blocks (Intel* TBB) - широко используемая, признанная библиотека шаблонов C++ для создания масштабируемых приложений и увеличения производительности. Поддерживает мобильные устройства с платформой Intel под управлением Android. Проверенные алгоритмы позволяют разработчикам эффективно распараллелить C++

мобильные приложения, что повышает производительность при снижении энергетических затрат.



Рисунок 4. Страница поддержки Intel* Beacon Mountain

Нельзя обойти вниманием инструментарий Marmalade SDK.

Marmalade SDK - кроссплатформенное SDK от Ideaworks3D Limited.

Представляет собой набор библиотек, образцов, инструментов и документации, необходимых для разработки, тестирования и развертывания приложений для мобильных устройств. Используется, в основном, для разработки игр. Многие получившие признание игры, такие как Cut the Rope и Plants vs. Zombies, были разработаны с использованием этого программного средства. К сожалению, Marmalade SDK представляет собой проприетарное программное обеспечение (самая дешевая лицензия \$15 в месяц) и не может быть рекомендована в данном учебном курсе, но читатель может самостоятельно попробовать бесплатную 30-дневную версию, доступную по ссылке здесь.

Нельзя не сказать об отечественных разработках. Например, компания 1С идет в ногу со временем, версия платформы 1С 8.3 позволяет разрабатывать мобильные приложения. Программный продукт "1С: Предприятие 8. Расширение для карманных компьютеров" обеспечивает возможность работы с данными информационных баз 1С: Предприятия 8 на мобильных устройствах (карманных компьютерах, коммуникаторах, терминалах сбора данных), а также на персональных компьютерах (в том числе ноутбуках), не имеющих прямого доступа к информационным базам 1С: Предприятия 8.

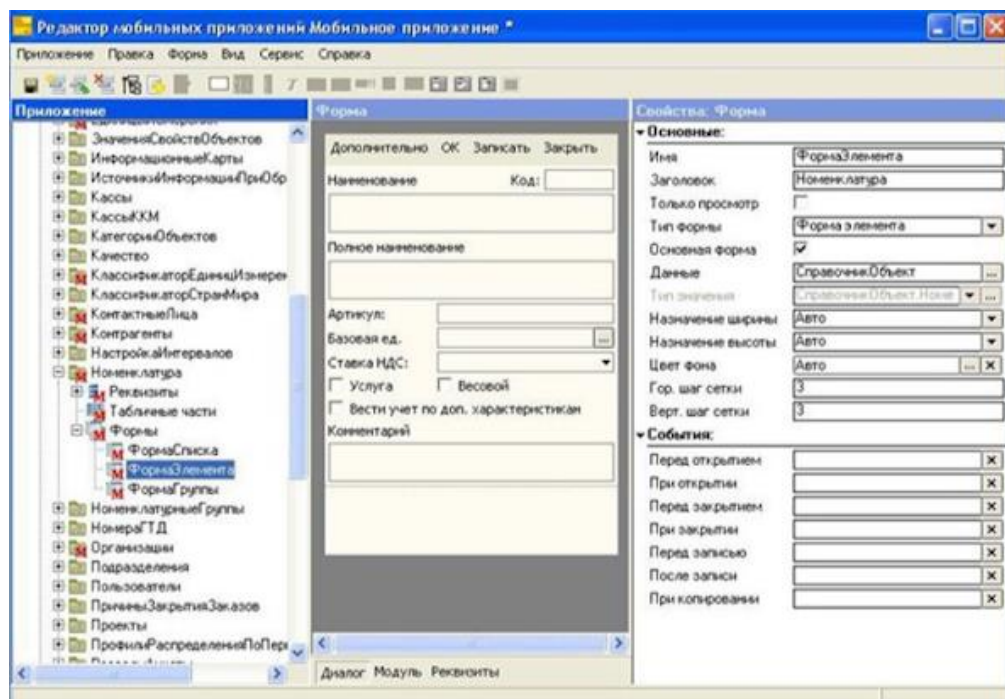


Рисунок 5. Редактор мобильных приложений 1С

Разумеется, данный программный продукт имеет очень узкую сферу применения, однако в некоторых случаях может являться наиболее удачным решением.

Эмуляторы

Эмуляция. Стандартный эмулятор Android

Эмуляция (англ. emulation) в вычислительной технике - комплекс программных, аппаратных средств или их сочетание, предназначенное для копирования (или эмулирования) функций одной вычислительной системы (гостя) на другой, отличной от первой, вычислительной системе (хосте) таким образом, чтобы эмулированное поведение как можно ближе соответствовало поведению оригинальной системы (гостя). Целью является максимально точное воспроизведение поведения в отличие от разных форм компьютерного моделирования, в которых имитируется поведение некоторой абстрактной модели Википедия.

Эмулятор - виртуальное мобильное устройство, которое запускается на компьютере. При помощи эмулятора можно разрабатывать и тестировать приложения без использования реальных устройств. На рисунке 1 приведен пример запущенного стандартного эмулятора. Подробно работа с эмуляторами рассмотрена в лабораторной работе.

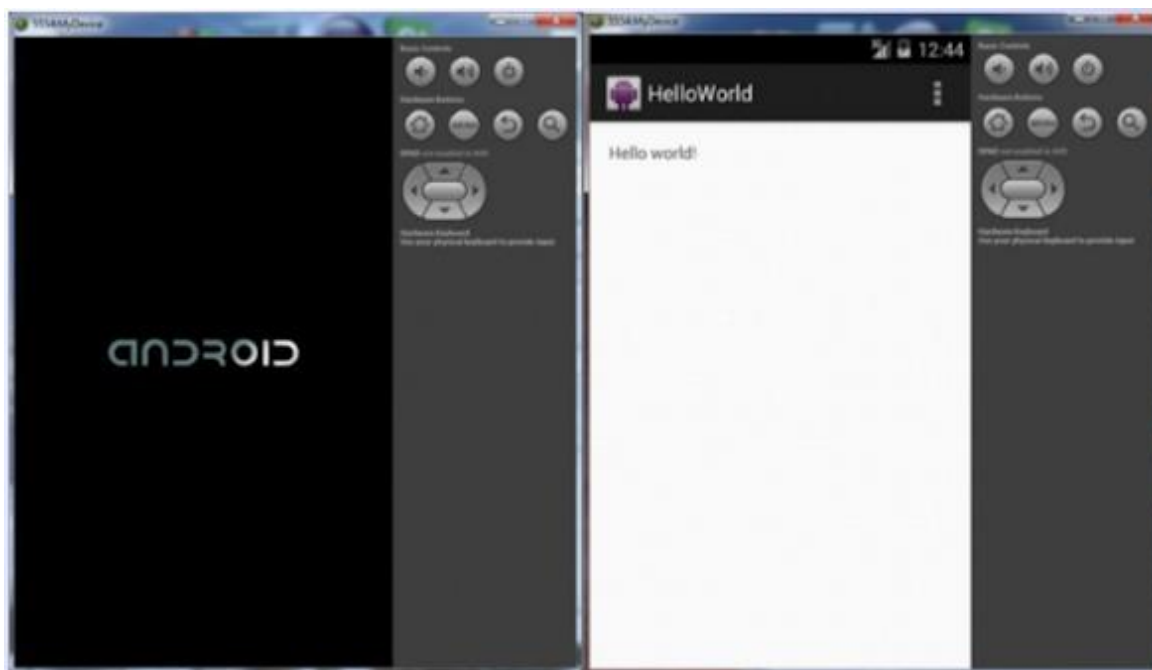


Рисунок 1. Эмулятор Android SDK в процессе запуска и приложение Hello, world!

К достоинствам использования эмуляторов можно отнести простоту их использования и нулевую стоимость. Разработчику не нужно покупать огромное количество устройств с различными характеристиками, чтобы проверить работоспособность приложения на различных смартфонах. Достаточно создать несколько эмуляторов с требуемыми характеристиками и запустить на них приложение. К сожалению, эмуляторы имеют и ряд недостатков:

- Требуют много системных ресурсов.
- Из-за различий в архитектуре процессоров компьютера и смартфона медленно запускаются. Современные персональные компьютеры построены на архитектурах x86 и x64, а большинство процессоров смартфонов на Android - ARM. Процесс эмуляции одной архитектуры на другой чрезвычайно сложен и происходит довольно медленно.
- В некоторых случаях стандартного эмулятора недостаточно. Речь идет о возможностях смартфонов, которыми обычные компьютеры не обладают (например, наличие датчика gps или акселерометра). В таких случаях полноценную отладку можно провести только с использованием реального устройства.

Альтернативные эмуляторы

Стандартный эмулятор, поставляемый вместе с Android SDK, не устраивает многих. Существуют проекты, поддерживающие разработку и развитие альтернативных эмуляторов. В качестве примера можно привести Genymotion (см. рис. 2) - быстрый эмулятор Android (по мнению его разработчиков). Он содержит предварительно настроенные образы Android (x86 с аппаратным ускорением OpenGL). Genymotion доступен для Linux, Windows и Mac OS X и требует для своей работы VirtualBox. Иными словами, Genymotion представляет собой виртуальную машину с установленной ОС

Android, которую пользователь запускает так же, как и другие виртуальные машины. Проблема высокого потребления системных ресурсов, конечно, не исчезает, однако скорость запуска существенно увеличивается. В настоящее время активно развивается.



Рисунок 2. Альтернативный эмулятор Genymotion

ЛЕКЦИЯ №17. СТРУКТУРА ТИПИЧНОГО МОБИЛЬНОГО ПРИЛОЖЕНИЯ.

В этой лекции показывается как установить SDK Android на Ubuntu 12.04 (на Windows аналогично), а также покажу как написать первое приложение используя IntelliJ IDEA 12.0.2

Android SDK представляет собой среду разработки для операционной системы Android mobile, который позволяет создавать приложения для устройств Android.

Шаг 1.

Переходим по ссылке developer.android.com и скачиваем отсюда пакет `adt-bundle-linux-x86.zip` (в зависимости от 32bit/64bit).

Внимание!

Обязательно должен быть установлен JDK 6 – 7, скачать можно [тут](#).

После того как вы скачали `adt-bundle-linux-x86.zip` распакуем его.

```
tar -xvzf adt-bundle-linux-x86.zip
```

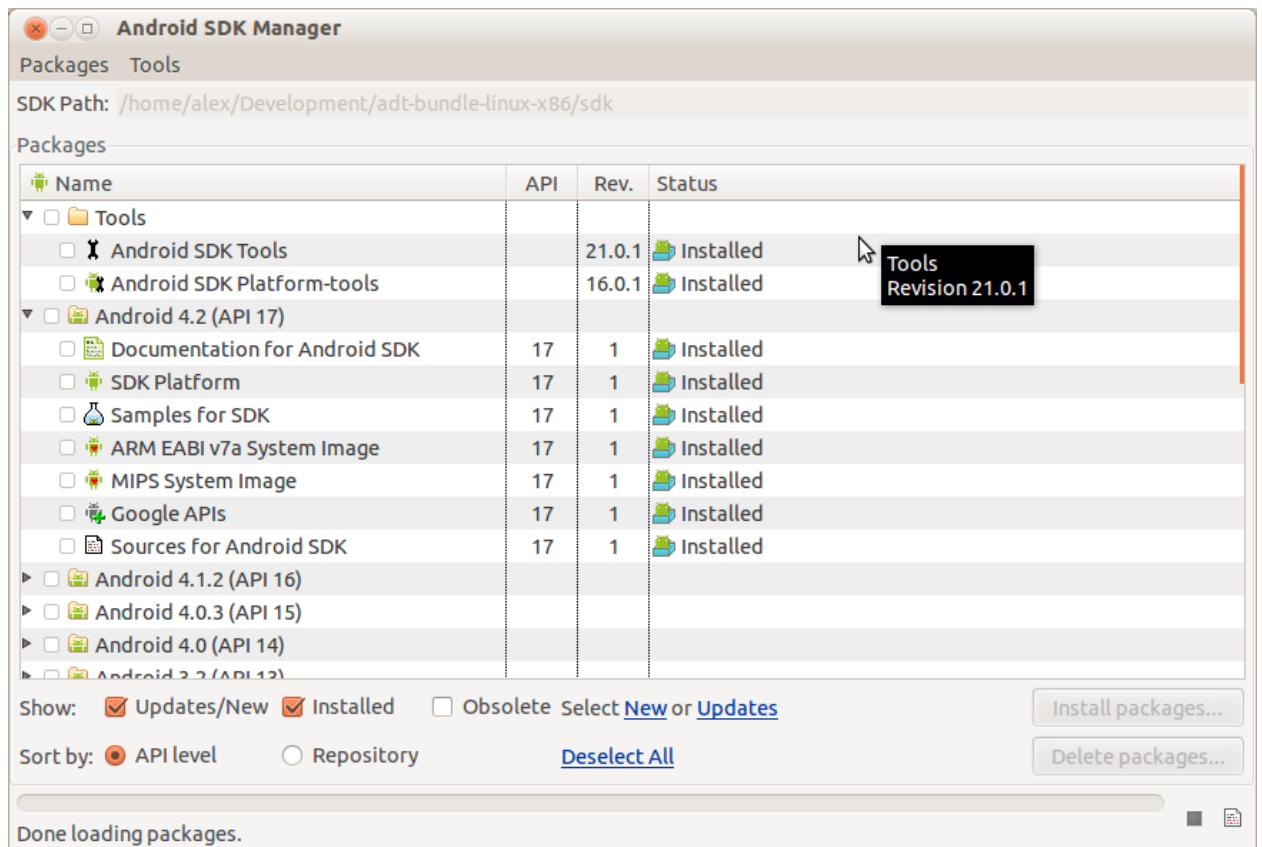
После извлечения пакета, выполните следующую команду, чтобы перейти в каталог инструментов.

```
cd ~/Development/adt-bundle-linux-x86/sdk/tools/
```

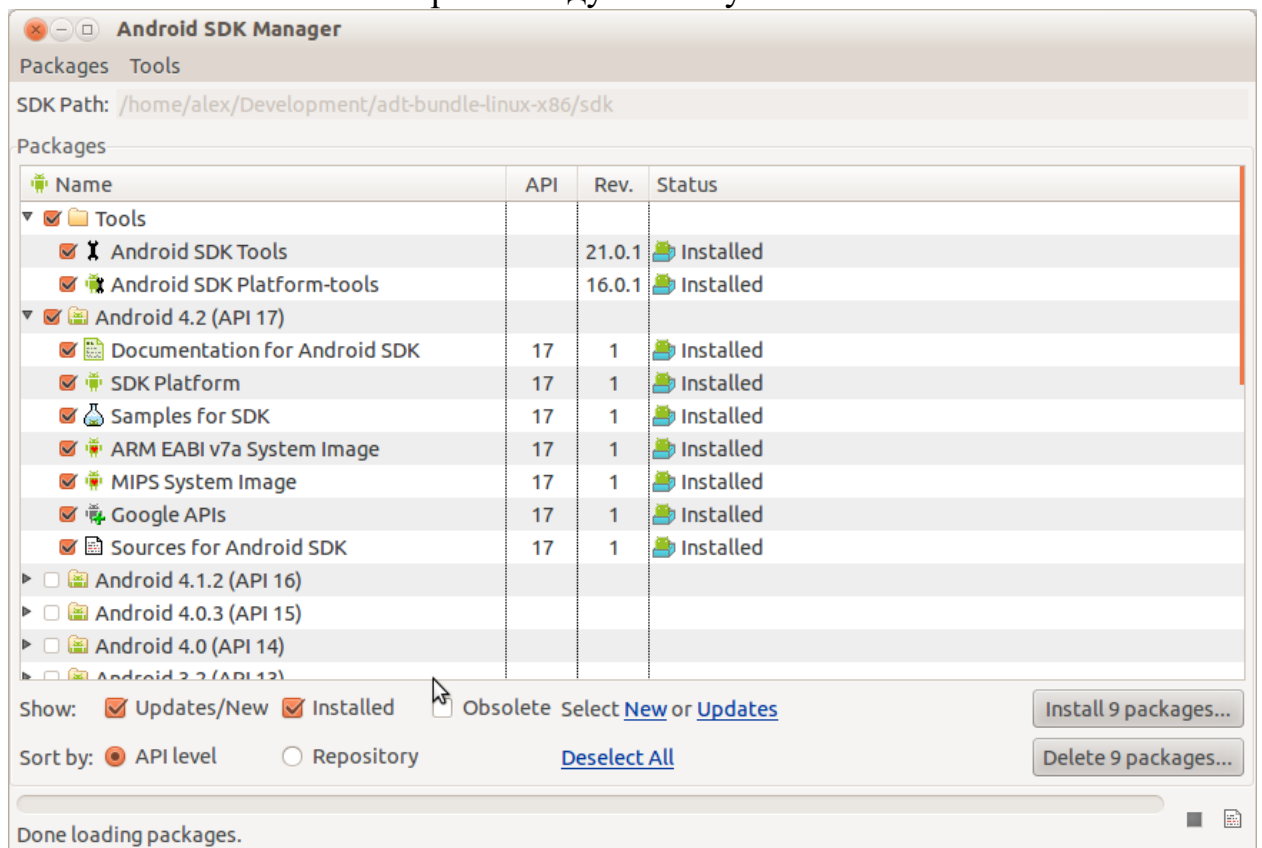
И, наконец, выполнить команду, чтобы начать установку.

```
./android
```

После вы увидите окно установки и обновления Android.



В появившемся окне выбираем следующие пункты:



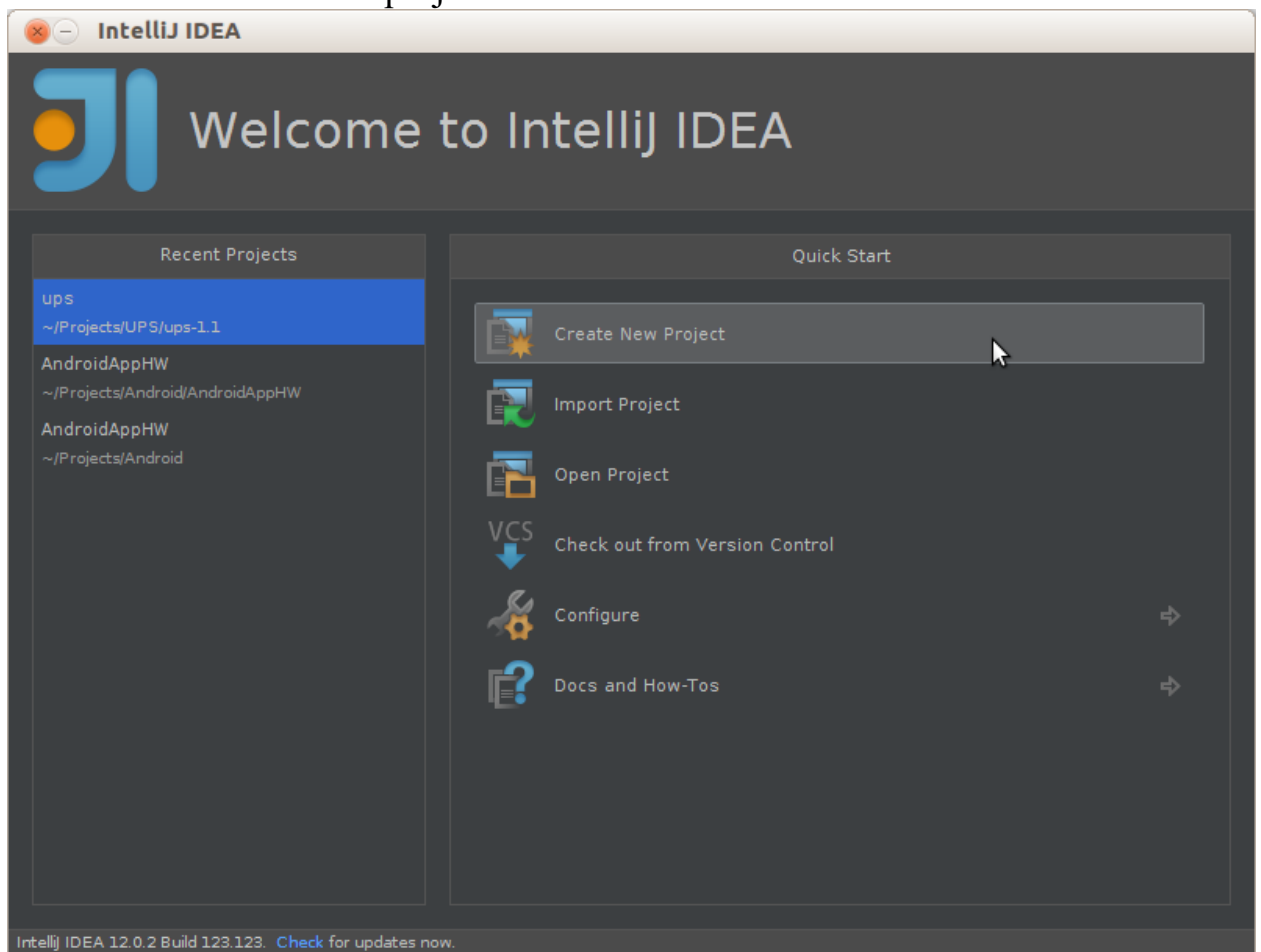
И нажимаем "Install 9 packages..."

Установка JDK Android успешно выполнена :)

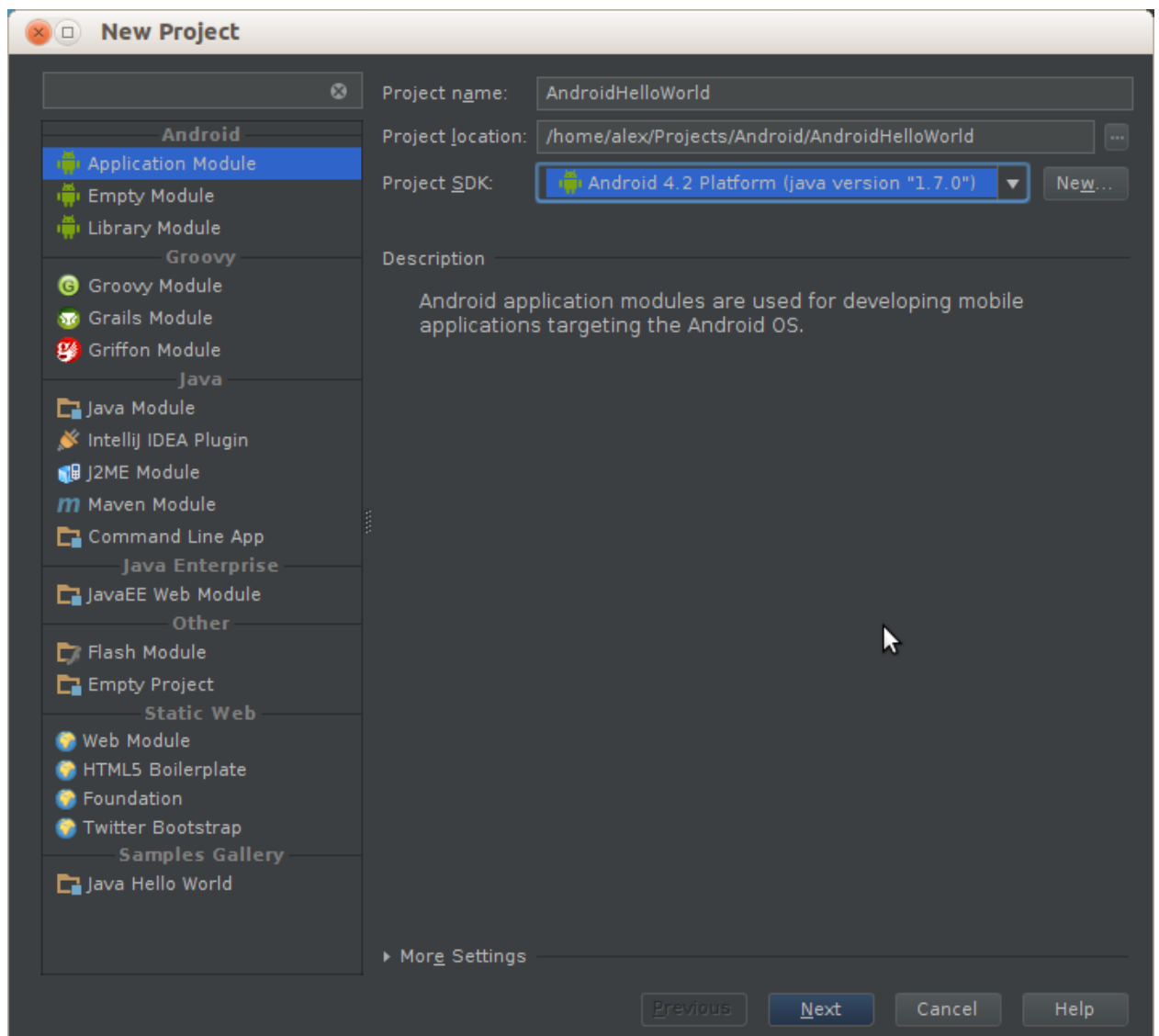
Переходим к Шагу 2.

Шаг 2.

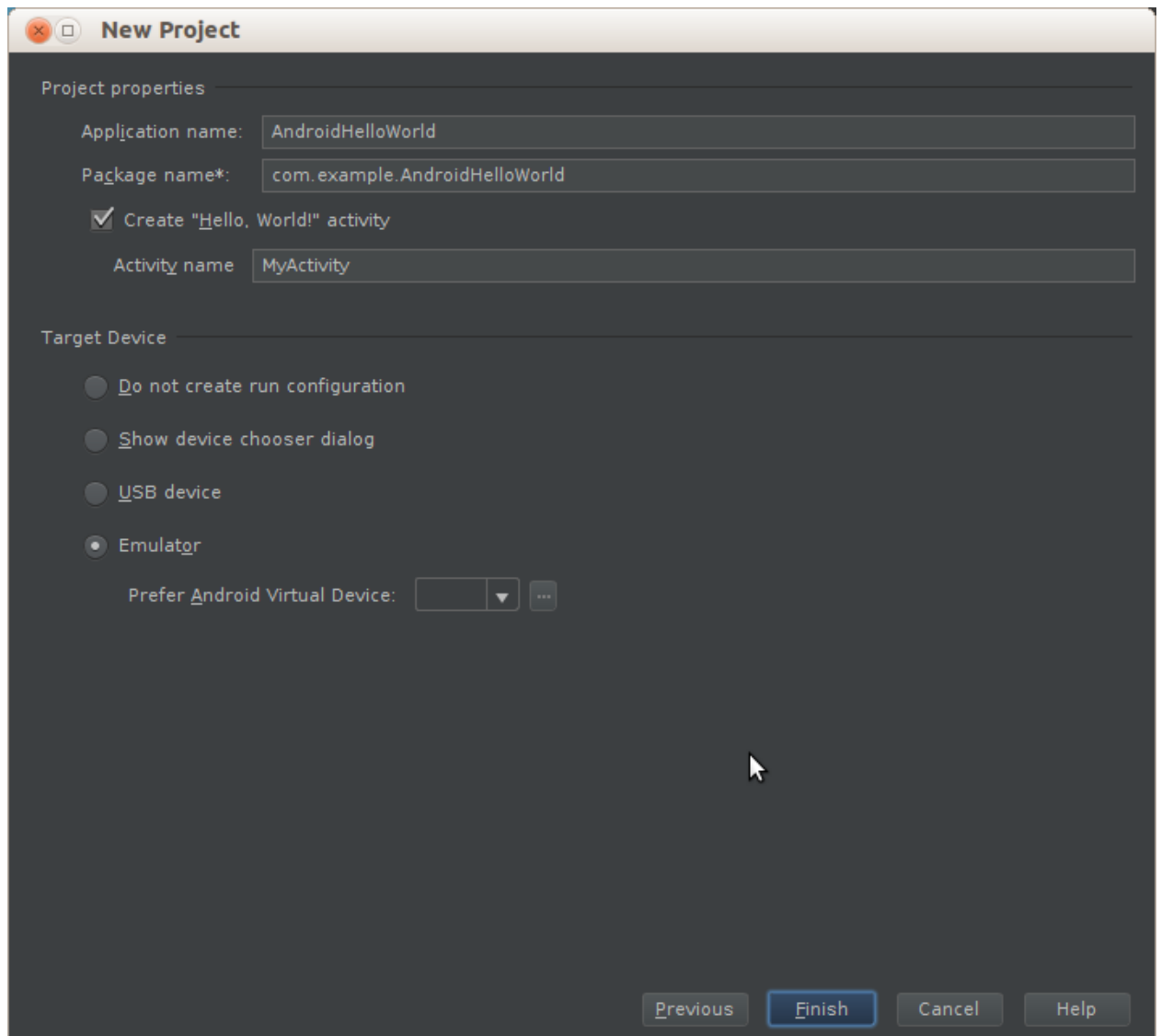
Запускаем IntelliJ Idea 12.
Нажимаем “Create new project”



Дальше выбираем “Android -> Application Module” вводим “Project name: <name>” указываем путь, где будет лежать ваш проект и указываем “Project SDK” в списке его может не быть, как его установить смотрите ниже, и нажимаем кнопку “Next”.



Дальше Ставим чек на Create “Hello, World” activity и указываем чеком что будет использоваться Emulator, нажимаем Finish.

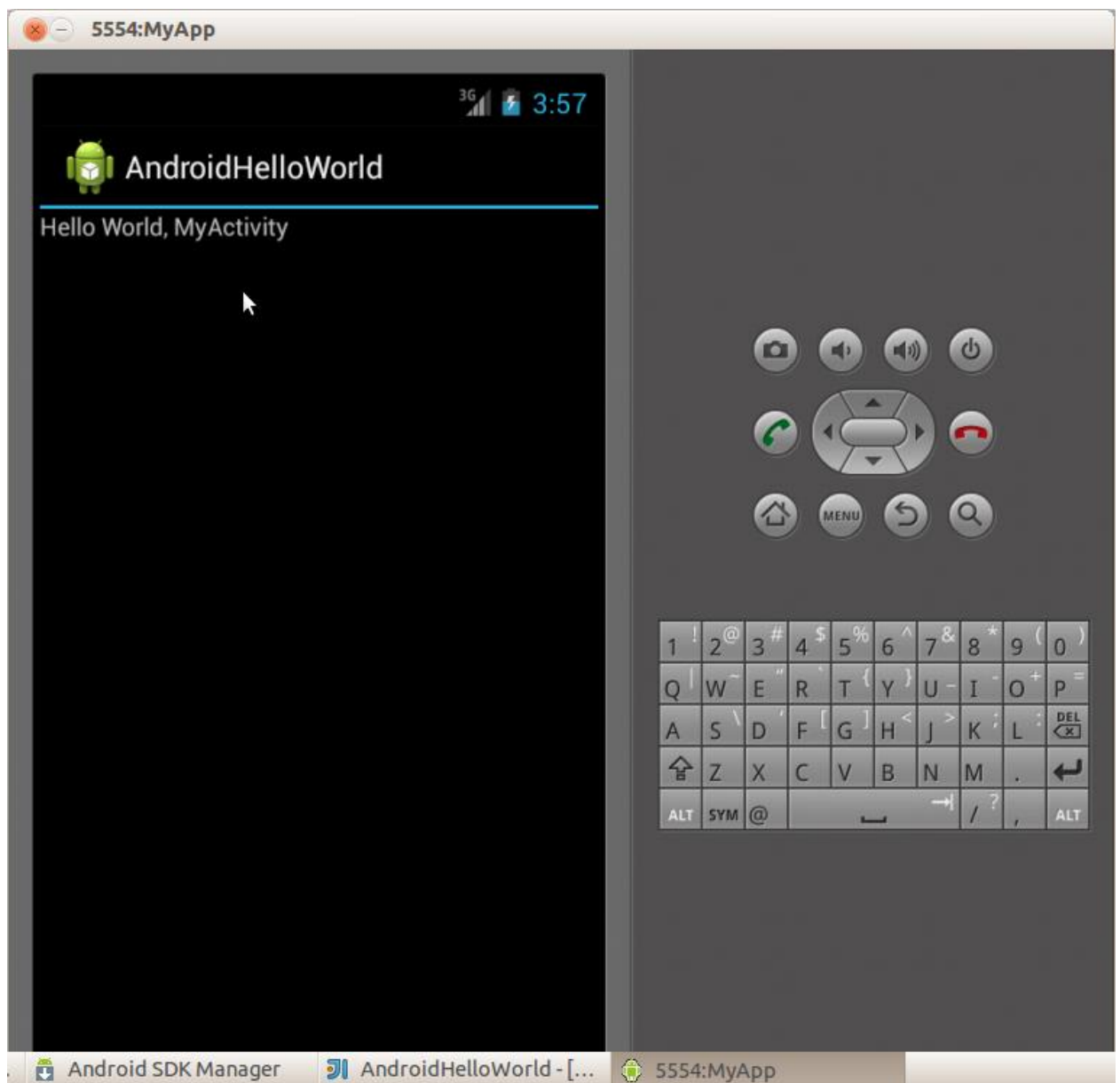


В результате всего этого у вас создастся проект, где будет находится файл: package com.example.AndroidHelloWorld;

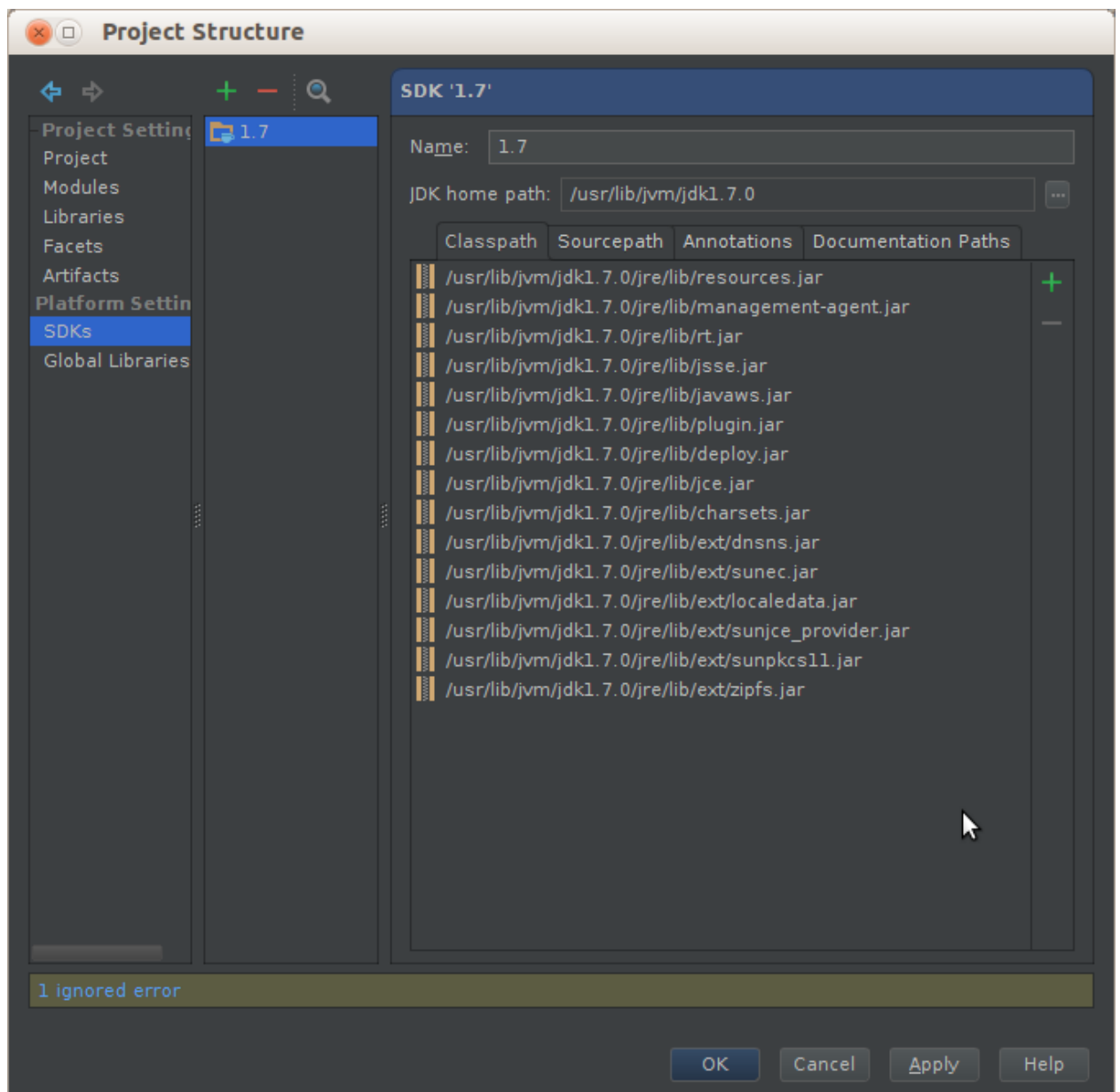
```
import android.app.Activity;
import android.os.Bundle;
```

```
public class MyActivity extends Activity {
    /**
     * Called when the activity is first created.
     */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

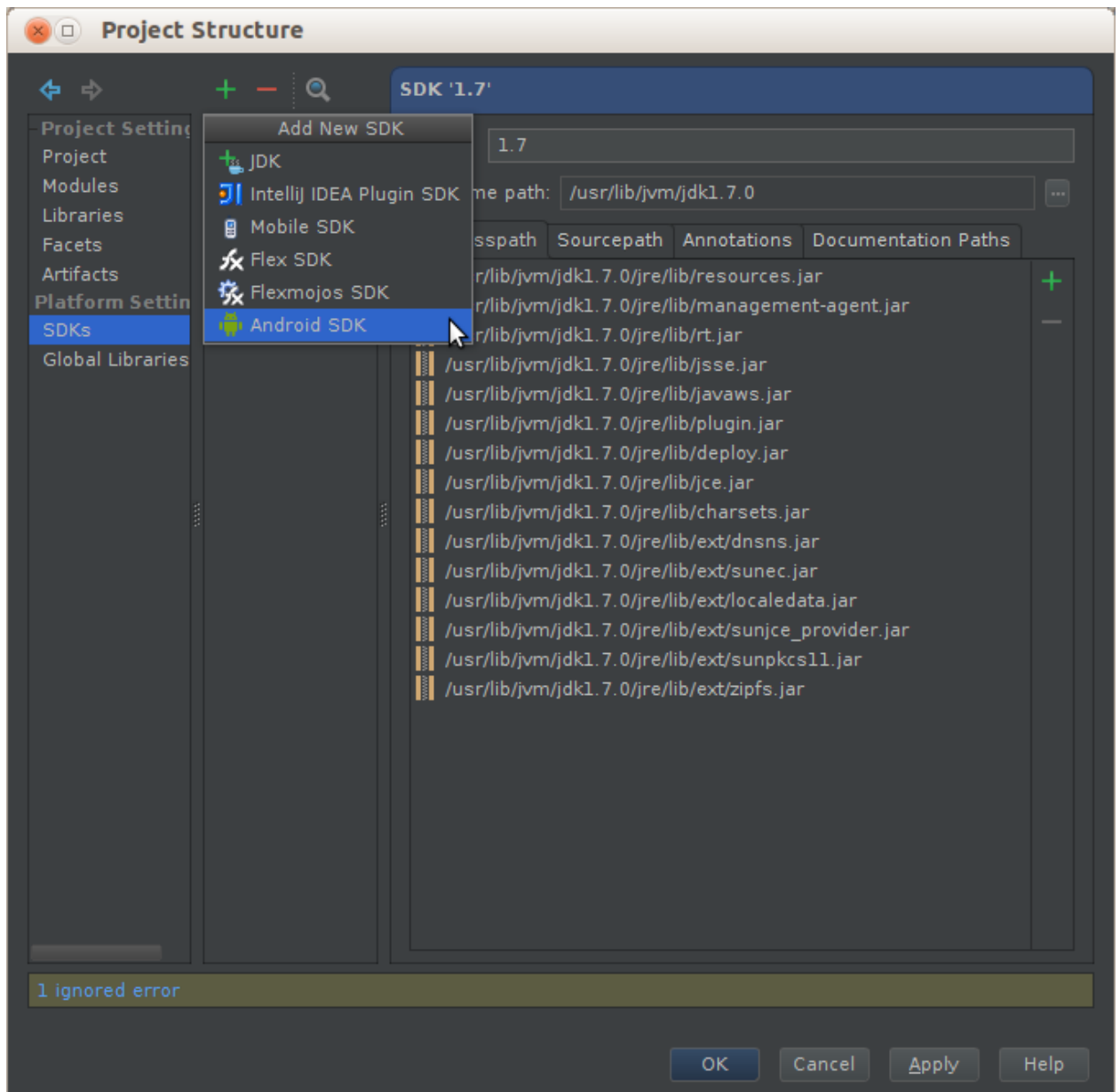
Запускаем его и получаем следующее:



Теперь рассмотрим случай, когда SDK Android не настроено в вашей IntelliJ IDEA 12.



В окне “Project Structure” нажимаем плюс (Add -> Android SDK)

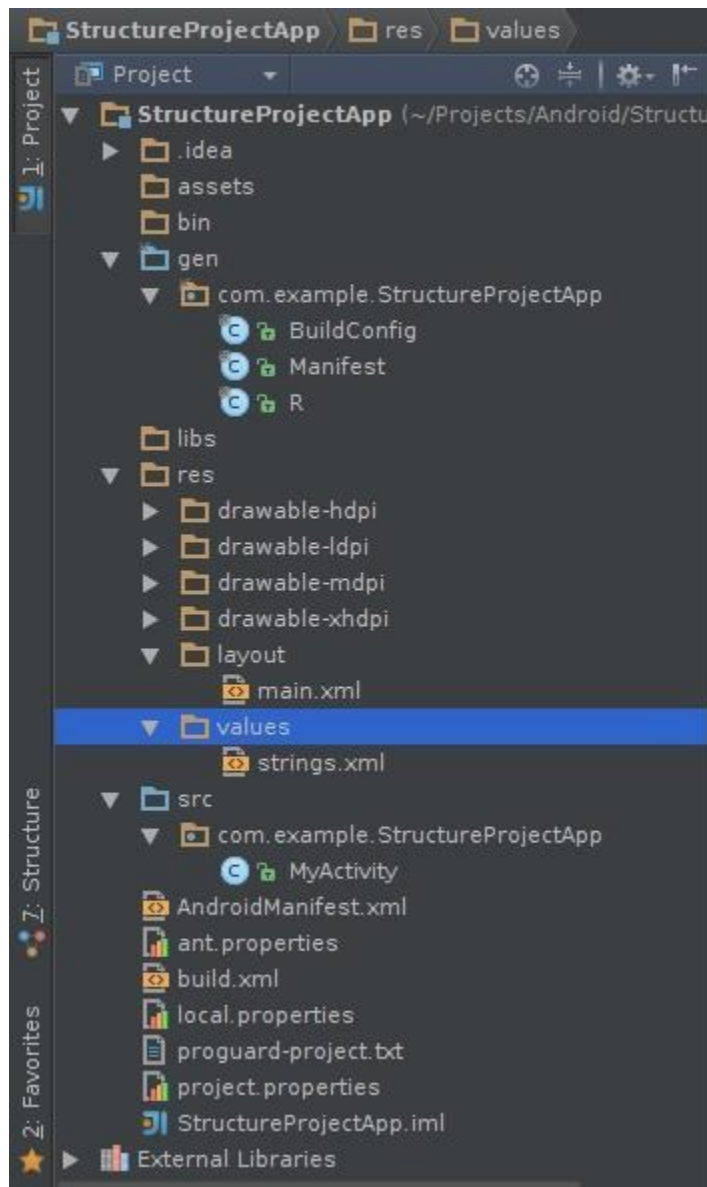


Указываем путь “~/Development/adt-bundle-linux-x86/sdk” и нажимаем “Ok”.

Для более лучшего понимания Android application я рекомендую ознакомиться со структурой проекта и его папками и файлами.

Сейчас мы познакомимся по ближе с приложением на Android и разберать его структуру, а начнем знакомство мы с создания проекта в IntelliJ IDEA 12.

И в только что созданном проекте посмотрим во вкладку “Project”:



src – содержит исходный код приложения.

com.example.StructureProjectApp – это package, наименование которого, указывается при создании проекта.

Main.java – файл исходного кода, в котором описан класс главной Activity.

gen – содержит java файлы, создаваемые автоматически во время разработки приложения. Файлы, находящиеся в папке gen не должны изменяться в ручную!

BuildConfig.java – содержит константу DEBUG, значение константы устанавливается автоматически в зависимости от того, является ли версия текущей сборки приложения, конечной или промежуточной.

Константа позволяет запускать код только в режиме отладки. Файл BuildConfig.java был добавлен после выхода ADT revision 17.

```
public final class BuildConfig {
    public final static boolean DEBUG = true;
}
```


R.java – хранит уникальные идентификаторы для всех ресурсов приложения (ID).

Имена этих ID – констант совпадают с именами файлов ресурсов. Рассмотрим содержимое XML-файла строковых ресурсов res/values/strings.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">StructureProjectApp</string>
</resources>
```

main.xml – файл описывает компоновку элементов экрана.

assets – папка, позволяющая организовать совершенно произвольную структуру вложенных папок и файлов. Содержимое папки assets не считается ресурсами.

bin – каталог сборки приложения. При компиляции приложения Google Android получается сборка, размещаемая в этом каталоге.

res – содержит структуру папок ресурсов приложения. Наименование каждой папки определяет тип ресурсов, которые она должна содержать.

Например:

res/drawable-ldpi предназначается для файлов изображений, которые будут применяться на экранах низкой плотности.

res/values – для строковых ресурсов, цветов и размеров.

res/layout – для файлов компоновки экрана.

В Google Android множество видов ресурсов. Каждый из которых, я буду разбирать по мере создания нашего первого приложения.

AndroidManifest – файл, который указывает точку входа в приложение.

Служит для задания общей информации о приложении. Определения атрибутов приложения. Запроса прав доступа к определенным критическим операциям и другой информации. Данный файл принято называть файлом описания, или файлом манифеста.

protect.properties – содержит параметры проекта, например версию API. Файл создается автоматически и не может быть удален.

ЛЕКЦИЯ №18-№19. ЭЛЕМЕНТЫ УПРАВЛЕНИЯ И КОНТЕЙНЕРЫ.

Управление мобильным приложением осуществляется через специальные наборы элементов управления (контролов) на экране устройства. Элементы управления отображаются пользователю в пользовательском интерфейсе, а настраиваются в режиме администрирования "Первой Формы". Мобильное приложение 1F поддерживает следующие элементы управления:

- нижнее меню (меню вкладок) — набор кнопок внизу экрана, каждая кнопка соответствует вкладке (режиму просмотра). Максимально возможное количество вкладок - 6 штук. Нижнее меню присутствует на экране всегда. По умолчанию не содержит ни одной вкладки.

- боковое (левое) меню — открывается жестом swipe (скользнуть пальцем по экрану слева направо) или нажатием на кнопку  в верхнем левом углу экрана. Это иерархический список элементов, который формируется из

двух частей: сверху главное меню, под ним меню Избранное. По умолчанию главное меню содержит пункт "Настройки".

- дашборд (рабочий стол) — набор плиток, доступен из бокового\нижнего меню в пунктах с названием "Главная". Рабочий стол может отображать разные данные — например, для широких плиток Категорий и Задач он может загружать сокращенные тексты задач.

- верхнее (выпадающее) меню — отображается в разделе "Лента", открывается нажатием на название раздела.

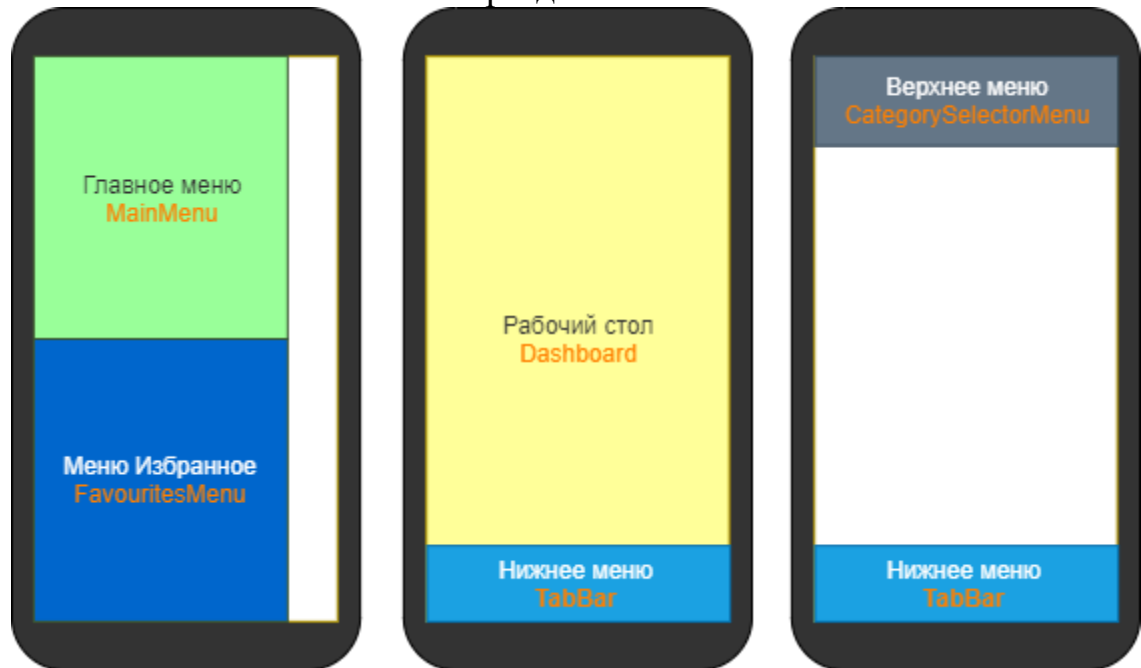


Схема мобильного интерфейса. Контейнеры.

Принципы настройки мобильного интерфейса

Для создания и настройки элементов мобильного интерфейса используются контейнеры, шаблоны и блоки.

Контейнер — это упорядоченный список элементов интерфейса. Каждый экран приложения состоит из контейнеров, одного или нескольких (за исключением ненастраиваемых экранов загрузки и авторизации). Система поддерживает 5 видов контейнеров: рабочий стол (Dashboard), нижнее меню (TabBar), главное меню (MainMenu), меню Избранное (FavouritesMenu), верхнее меню (CategorySelectorMenu).

Состав контейнеров на экране может различаться для разных учетных записей. Например, для некоторых пользователей может отображаться нижнее меню, а для других нет.

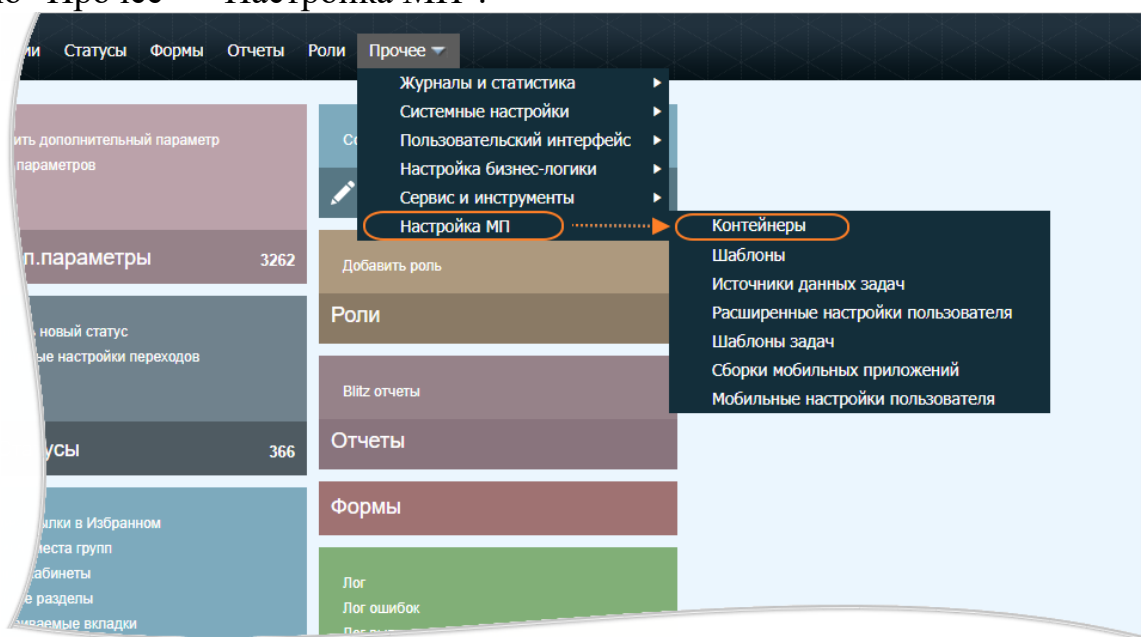
Элементы контейнера называются блоками. Блок — это "атом", наименьшая единица мобильного интерфейса. Блок может быть кнопкой, плиткой рабочего стола, пунктом меню и другим базовым элементом, из которых состоит интерфейс экрана приложения. Действие, которое приложение будет осуществлять при взаимодействии с блоком, определяется типом блока и типом его данных. Блоки могут быть вложенными (включать в себя другие блоки) — это блоки-папки. Примеры блоков: кнопка вкладки "Главная" в нижнем меню, ДП в карточке задачи.

Внешний вид и поведение контейнера и блока определяется шаблоном. Любой контейнер и любой блок могут иметь один или несколько шаблонов. Шаблоны имеют наборы параметров, таких как цвет, размер, масштабируемость и др.

Все ID и названия контейнеров, шаблонов, полей и блоков являются регистрозависимыми: TabBar и tabBar — это разные ID.

Если у пользователя в мобильном приложении есть несколько аккаунтов (учетных записей), при переключении между ними некоторые контейнеры перестраиваются (например, нижнее меню свое для каждого аккаунта), а некоторые контейнеры объединяются (например, в боковом левом меню отображаются данные для всех аккаунтов, они следуют друг за другом).

Чтобы перейти к настройке контейнеров, выберите одноименный пункт в меню "Прочее" - "Настройка МП".



Переход к настройке контейнеров.

Откроется таблица со списком существующих контейнеров.

Контейнеры

Добавить

Перетащите сюда заголовок столбца для группировки

	Id	Шаблон	Группы	Порядок	Активен		
	iosLeftMenu	menu		1			
	iosToolbar	toolbar		2			
	FavouritesMenu	menu	UserGroup	4			
	Dashboard	dashboard		5			
	TabBar	toolbar		6			
	MainMenu	menu		7			
	iosMainMenu	menu		8			

Список контейнеров.

Создание контейнера

Чтобы создать новый контейнер, нажмите кнопку **Добавить**. Откроется окно для добавления контейнера. Идентификатор контейнера (поле Id) должен начинаться с латинской буквы. Выберите шаблон, по которому будет создаваться контейнер, а также укажите группы пользователей, которые смогут его использовать.

The screenshot shows a window titled 'Контейнеры' (Containers) with a '+ Добавить' (Add) button. Below it, there are three input fields: 'Id:' (empty), 'Шаблон:' (contactsFolder), and 'Кому:' (Добавить сотрудника). A 'Добавить' (Add) button is at the bottom right.

Окно создания нового контейнера.

Если доступ к контейнеру предоставлен только определенным группам, то в правой колонке списка контейнеров будут отображаться названия этих групп:

	iosToolbar	toolbar		2	☒		
	FavouritesMenu	menu	UserGroup	4	☒		
	Dashboard	dashboard		5	☒		

Группа, имеющая доступ к контейнеру.

ID контейнера не является его уникальным идентификатором. Напротив, для некоторых контейнеров ID должны быть строго определенными - например, для нижнего меню ID должен быть TabBar, для главного рабочего стола - Dashboard. И если для разных групп пользователей настроены разные контейнеры одного типа, они будут иметь одинаковые ID.

Все ID и названия контейнеров, шаблонов, полей и блоков являются регистрозависимыми: TabBar и tabBar — это разные ID.

Настройка контейнеров описана [здесь](#).

Редактирование основных настроек контейнера

Для редактирования основных настроек контейнера нажмите иконку в начале нужной строки таблицы. Строка перейдет в режим редактирования.

The screenshot shows a row for the 'Dashboard' container in edit mode. It has a 'Dashboard' label, a 'dashboard' template, a 'Кому:' field with 'Клиенты' selected, a value of '21', and a checked checkbox.

Редактирование основных параметров контейнера.

Вы можете изменить Id контейнера, выбрать другой шаблон, указать группы, которым доступен этот шаблон, и изменить признак активности контейнера.

После внесения изменений нажмите иконку в начале строки чтобы сохранить внесенные изменения или иконку чтобы отменить изменения.

Изменить порядковые номера контейнеров можно, перетаскивая мышью строки в таблице.

- если такого контейнера нет, то контейнер с наименьшим значением в колонке "Порядок", для которого в колонке "Группы" не указано ничего;
- если и такого контейнера нет, то будет отображаться произвольный контейнер с таким Id.

Чтобы избежать случая, когда пользователю отображается произвольный контейнер, рекомендуется настраивать контейнер по умолчанию — такой, где в колонке "Группы" не указано ничего, и у нее должен быть наименьший порядковый номер среди всех одинаковых ID.

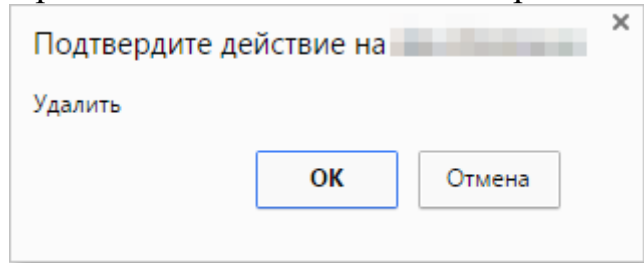
Копирование контейнера

Чтобы создать новый контейнер на основе существующего, используйте иконку копирования  в конце соответствующей строки таблицы.

Удаление контейнера

Для удаления контейнера нажмите на иконку  в соответствующей строке таблицы.

При этом появится окно подтверждения.



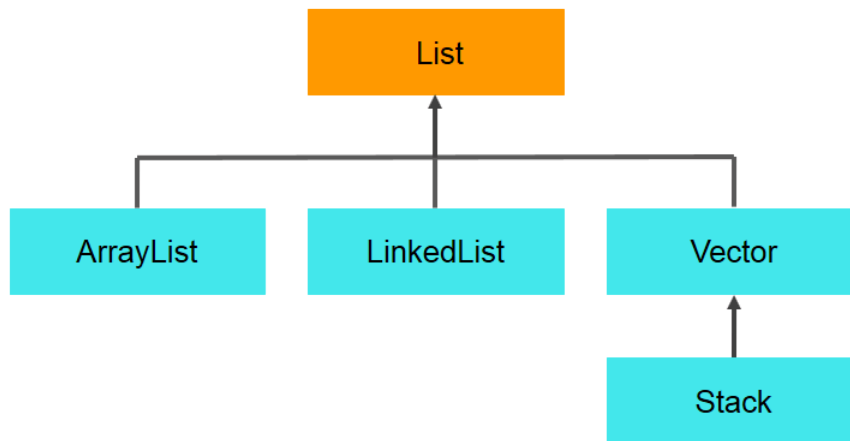
Окно удаления объекта.

Для подтверждения удаления нажмите ОК. Если действие было сделано ошибочно, нажмите Отмена.

ЛЕКЦИЯ №20-№21. РАБОТА СО СПИСКАМИ.

Виды списков

Списков существует несколько. List - это только название интерфейса, который объединяет несколько реализаций:



vertex vertex-academy.com

На практике мы чаще всего используем:

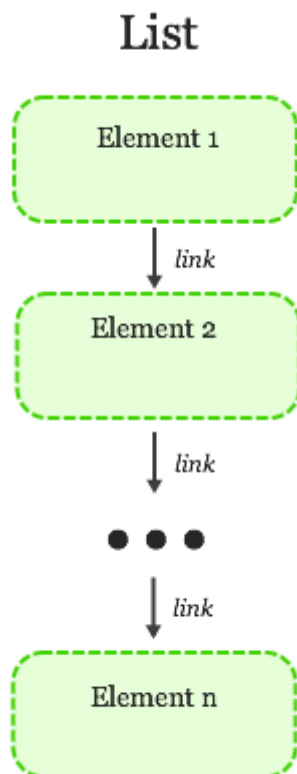
ArrayList (мы также рассматривали в статье ["Что такое ArrayList"](#))

LinkedList

На Vector можете не смотреть ☐

Как хранятся данные

Так что же все-таки такое List? В памяти его можно представить так:



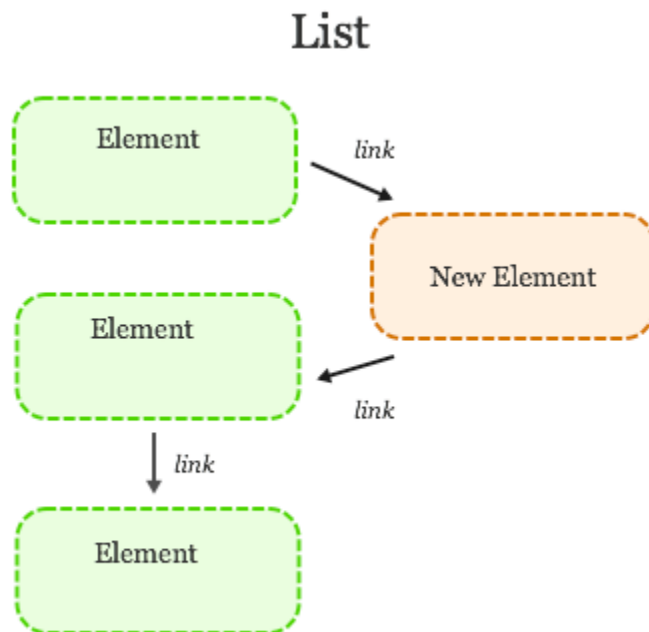
То есть список, в классическом смысле - это тип данных, в котором каждый элемент содержит какой-то контент (например, String, Integer или вообще какой-нибудь объект), а также ссылку на следующий элемент списка.

Зачем?

Основное преимущество списка по сравнению с массивом - его можно легко расширять.

Если у нас есть массив на 3 элемента, а нам вдруг понадобилось положить туда 10 элементов, нам просто придется создавать новый массив на 10 элементов. Только так и никак иначе! ;(

Но со списками все по-другому. Мы можем свободно добавлять элементы в начало, середину или конец списка. Все потому, что для массива нам нужен один "сплошной" участок памяти, и чтобы его расширить, придется искать другой "целый" кусок памяти. Но с List проще - мы просто меняем ссылки:



Для создания простых списков применяется интерфейс List, который расширяет функциональность интерфейса Collection. Некоторые наиболее часто используемые методы интерфейса List:

`void add(int index, E obj)`: добавляет в список по индексу index объект obj

`boolean addAll(int index, Collection<? extends E> col)`: добавляет в список по индексу index все элементы коллекции col. Если в результате добавления список был изменен, то возвращается true, иначе возвращается false

`E get(int index)`: возвращает объект из списка по индексу index

`int indexOf(Object obj)`: возвращает индекс первого вхождения объекта obj в список. Если объект не найден, то возвращается -1

`int lastIndexOf(Object obj)`: возвращает индекс последнего вхождения объекта obj в список. Если объект не найден, то возвращается -1

`ListIterator<E> listIterator ()`: возвращает объект ListIterator для обхода элементов списка

`static <E> List<E> of(элементы)`: создает из набора элементов объект List

`E remove(int index)`: удаляет объект из списка по индексу index, возвращая при этом удаленный объект

E set(int index, E obj): присваивает значение объекта obj элементу, который находится по индексу index

void sort(Comparator<? super E> comp): сортирует список с помощью компаратора comp

List<E> subList(int start, int end): получает набор элементов, которые находятся в списке между индексами start и end

По умолчанию в Java есть встроенная реализация этого интерфейса - класс ArrayList. Класс ArrayList представляет обобщенную коллекцию, которая наследует свою функциональность от класса AbstractList и применяет интерфейс List. Проще говоря, ArrayList представляет простой список, аналогичный массиву, за тем исключением, что количество элементов в нем не фиксировано.

ArrayList имеет следующие конструкторы:

ArrayList(): создает пустой список

ArrayList(Collection <? extends E> col): создает список, в который добавляются все элементы коллекции col.

ArrayList (int capacity): создает список, который имеет начальную емкость capacity

Емкость в ArrayList представляет размер массива, который будет использоваться для хранения объектов. При добавлении элементов фактически происходит перераспределение памяти - создание нового массива и копирование в него элементов из старого массива. Изначальное задание емкости ArrayList позволяет снизить подобные перераспределения памяти, тем самым повышая производительность.

Используем класс ArrayList и некоторые его методы в программе:

```
import java.util.ArrayList;
```

```
public class Program{
```

```
    public static void main(String[] args) {
```

```
        ArrayList<String> people = new ArrayList<String>();
```

```
        // добавим в список ряд элементов
```

```
        people.add("Tom");
```

```
        people.add("Alice");
```

```
        people.add("Kate");
```

```
        people.add("Sam");
```

```
        people.add(1, "Bob"); // добавляем элемент по индексу 1
```

```
        System.out.println(people.get(1)); // получаем 2-й объект
```

```
        people.set(1, "Robert"); // установка нового значения для 2-го объекта
```

```
        System.out.printf("ArrayList has %d elements \n",  
        people.size());
```

```

    for(String person : people){

        System.out.println(person);
    }
    // проверяем наличие элемента
    if(people.contains("Tom")){

        System.out.println("ArrayList contains Tom");
    }

    // удалим несколько объектов
    // удаление конкретного элемента
    people.remove("Robert");
    // удаление по индексу
    people.remove(0);

    Object[] peopleArray = people.toArray();
    for(Object person : peopleArray){

        System.out.println(person);
    }
}

```

Консольный вывод программы:

Bob

ArrayList has 5 elements

Tom

Robert

Alice

Kate

Sam

ArrayList contains Tom

Alice

Kate

Sam

Здесь объект ArrayList типизируется классом String, поэтому список будет хранить только строки. Поскольку класс ArrayList применяет интерфейс Collection<E>, то мы можем использовать методы данного интерфейса для управления объектами в списке.

Для добавления вызывается метод add. С его помощью мы можем добавлять объект в конец списка: people.add("Tom"). Также мы можем добавить объект на определенное место в списке, например, добавим объект на второе место (то есть по индексу 1, так как нумерация начинается с нуля): people.add(1, "Bob")

Метод size() позволяет узнать количество объектов в коллекции.

Проверку на наличие элемента в коллекции производится с помощью метода `contains`. А удаление с помощью метода `remove`. И так же, как и с добавлением, мы можем удалить либо конкретный элемент `people.remove("Tom");`, либо элемент по индексу `people.remove(0);` - удаление первого элемента.

Получить определенный элемент по индексу мы можем с помощью метода `get()`: `String person = people.get(1);`, а установить элемент по индексу с помощью метода `set`: `people.set(1, "Robert");`

С помощью метода `toArray()` мы можем преобразовать список в массив объектов.

И поскольку класс `ArrayList` реализует интерфейс `Iterable`, то мы можем пробежаться по списку в цикле `for`-`each`: `for(String person : people)`.

Хотя мы можем свободно добавлять в объект `ArrayList` дополнительные объекты, в отличие от массива, однако в реальности `ArrayList` использует для хранения объектов опять же массив. По умолчанию данный массив предназначен для 10 объектов. Если в процессе программы добавляется гораздо больше, то создается новый массив, который может вместить в себя все количество. Подобные перераспределения памяти уменьшают производительность. Поэтому если мы точно знаем, что у нас список не будет содержать больше определенного количества элементов, например, 25, то мы можем сразу же явным образом установить это количество, либо в конструкторе: `ArrayList<String> people = new ArrayList<String>(25);`, либо с помощью метода `ensureCapacity`: `people.ensureCapacity(25);`

Отличие `ArrayList` и `LinkedList`

Не вдаваясь в подробности, запомните, что:

`LinkedList` лучше использовать, если Вы знаете, что будете часто добавлять и удалять элементы из списка.

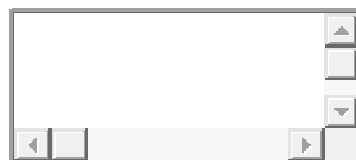
`ArrayList` лучше использовать, если Вам главное иметь быстрый доступ к элементам списка.

Синтаксис



```
1 ArrayList<String> myArrayList = new ArrayList<String>();
```

или



```
1 LinkedList<String> myLinkedList = new  
  LinkedList<String>();
```

Обратите внимание, что в скобках мы задаем тип данных, который будет храниться в списке:

Тип
↓

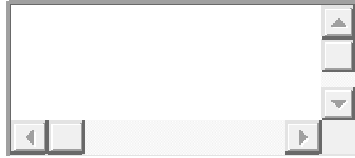
```
ArrayList<String> myArrayList = new ArrayList<String>();
```

В наших примерах это String, но Вы можете задать любой другой.

Кстати, далее мы будем приводить примеры на ArrayList.

Добавление элемента

Как добавлять элемент в список? Для этого используется метод add():



```
public class Test {
```

```
    public static void main(String[] args) {
```

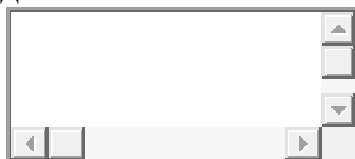
```
        ArrayList<String>    myArrayList    =    new
ArrayList<String>();
```

```
        LinkedList<String>    myLinkedList    =    new
LinkedList<String>();
```

```
        myArrayList.add("We add an element to ArrayList!");
        myLinkedList.add("We add an element to
LinkedList!");
    }
}
```

Кстати, мы можем добавлять в список не только элемент класса, который мы указали, но и всех его классов-наследников.

Например, представим, что у нас есть класс Animal, у которого есть наследник Cat:

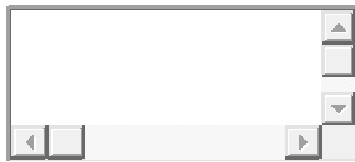


```
class Animal {
    // ...
}
```



```
class Cat extends Animal {
    // ...
}
```

Тогда, если указать как принимаемый тип Animal, а положить внутрь Cat - ничего плохого не произойдет:



```
public class Test {
```

```
    public static void main(String[] args) {
```

```
        ArrayList<Animal> myArrayList = new  
ArrayList<Animal>();
```

```
        Cat myCat = new Cat();  
        myArrayList.add(myCat);
```

```
    }
```

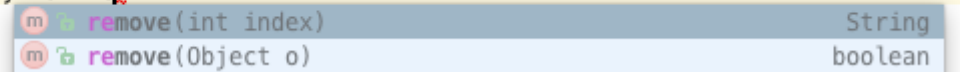
```
}
```

Удаление элемента

Удалить элемент можно двумя способами - по индексу и по ссылке:

```
myArrayList.add("Hey!");
```

```
myArrayList.re
```



Например, представим, что мы добавляем в наш ArrayList строку "Hey":



```
public class Test {
```

```
    public static void main(String[] args) {
```

```
        ArrayList<String> myArrayList = new  
ArrayList<String>();
```

```
        myArrayList.add("Hey!");
```

```
    }
```

```
}
```

Чтобы удалить ее, мы можем обратиться по индексу (индекс первого элемента - ноль):

```
myArrayList.remove(0);
```

или написать "Hey" как аргумент:

```
myArrayList.remove("Hey!");
```

И так, и так наш элемент будет удален.

Проход по списку

Итак, давайте завершим наш урок проходом по списку. А то мы все добавляли, удаляли, а наглядно посмотреть работает оно или нет не могли ☐

Давайте рассмотрим самый простой способ - пройтись по списку с помощью цикла for:



```
public class Test {
```

```
    public static void main(String[] args) {
```

```
        ArrayList<String>    myArrayList    =    new  
ArrayList<String>();
```

```
        myArrayList.add("Hey!");
```

```
        for (int i = 0; i < myArrayList.size(); i++)  
            System.out.println(myArrayList.get(i));
```

```
    }
```

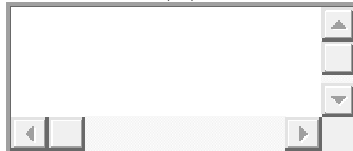
```
}
```

Здесь у нас в списке лежит только один элемент - строка "Hey". Давайте запустим:

```
Hey!
```

```
Process finished with exit code 0
```

Отлично! Добавим еще что-нибудь:



```
public class Test {
```

```
    public static void main(String[] args) {
```

```
        ArrayList<String>    myArrayList    =    new  
ArrayList<String>();
```

```
        myArrayList.add("Hey!");  
        myArrayList.add("This");  
        myArrayList.add("is");  
        myArrayList.add("my");  
        myArrayList.add("first");  
        myArrayList.add("list!");
```

```

        for (int i = 0; i < myArrayList.size(); i++)
            System.out.println(myArrayList.get(i));
    }
}

```

Запускаем:

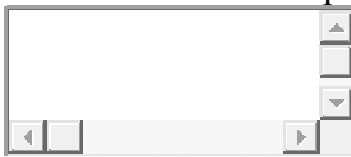
```

Hey!
This
is
my
first
list!

```

Process finished with exit code 0

Замечательно - теперь удалим что-нибудь:



```

public class Test {

```

```

    public static void main(String[] args) {

```

```

        ArrayList<String>    myArrayList    =    new
ArrayList<String>();

```

```

        myArrayList.add("Hey!");
        myArrayList.add("This");
        myArrayList.add("is");
        myArrayList.add("my");
        myArrayList.add("first");
        myArrayList.add("list!");

```

```

        myArrayList.remove(0);
        for (int i = 0; i < myArrayList.size(); i++)
            System.out.println(myArrayList.get(i));

```

```

    }
}

```

Получим:

```

This
is
my
first
list!

```

Process finished with exit code 0

Хорошо, теперь Вы знаете как работать с List-ми.

ЛЕКЦИЯ №22-№23. ОБРАБОТКА СОБЫТИЙ.

Клиентские программы на языке JavaScript основаны на модели программирования, когда выполнение программы управляется событиями. При таком стиле программирования веб-браузер генерирует событие, когда с документом или некоторым его элементом что-то происходит. Например, веб-браузер генерирует событие, когда завершает загрузку документа, когда пользователь наводит указатель мыши на гиперссылку или нажимает клавишу на клавиатуре.

Если JavaScript-приложение интересуется определенным типом события для определенного элемента документа, оно может зарегистрировать одну или более функций, которая будет вызываться при возникновении этого события. Имейте в виду, что это не является уникальной особенностью веб-программирования: все приложения с графическим интерфейсом пользователя действуют именно таким образом - они постоянно ожидают, пока что-то произойдет (т.е. ждут появления событий), и откликаются на происходящее.

Тип события - это строка, определяющая тип действия, вызвавшего событие. Тип «mousemove», например, означает, что пользователь переместил указатель мыши. Тип «keydown» означает, что была нажата клавиша на клавиатуре. А тип «load» означает, что завершилась загрузка документа (или какого-то другого ресурса) из сети. Поскольку тип события - это просто строка, его иногда называют именем события.

Цель события - это объект, в котором возникло событие или с которым это событие связано. Когда говорят о событии, обычно упоминают тип и цель события. Например, событие «load» объекта Window или событие «click» элемента <button>. Самыми типичными целями событий в клиентских приложениях на языке JavaScript являются объекты Window, Document и Element, но некоторые типы событий могут происходить и в других типах объектов.

Обработчик события - это функция, которая обрабатывает, или откликается на событие. Приложения должны регистрировать свои функции обработчиков событий в веб-браузере, указав тип события и цель. Когда в указанном целевом объекте возникнет событие указанного типа, браузер вызовет обработчик. Когда обработчики событий вызываются для какого-то объекта, мы иногда говорим, что браузер «возбудил» или «сгенерировал» событие.

Объект события - это объект, связанный с определенным событием и содержащий информацию об этом событии. Объекты событий передаются функции обработчика события в виде аргумента (кроме IE8 и более ранних версий, где объект события доступен только в виде глобальной переменной event). Все объекты событий имеют свойство type, определяющее тип события, и свойство target, определяющее цель события.

Для каждого типа события в связанном объекте события определяется набор свойств. Например, объект, связанный с событиями от мыши, включает координаты указателя мыши, а объект, связанный с событиями от клавиатуры,

содержит информацию о нажатой клавише и о нажатых клавишах-модификаторах. Для многих типов событий определяются только стандартные свойства, такие как `type` и `target`, и не передается никакой дополнительной полезной информации. Для таких типов событий важно само наличие происшествия события, и никакая другая информация не имеет значения.

Распространение события - это процесс, в ходе которого браузер решает, в каких объектах следует вызвать обработчики событий. В случае событий, предназначенных для единственного объекта (таких как событие «load» объекта Window), надобность в их распространении отсутствует. Однако, когда некоторое событие возникает в элементе документа, оно распространяется, или «всплывает», вверх по дереву документа.

Если пользователь щелкнет мышью на гиперссылке, событие «mousemove» сначала будет возбуждено в элементе `<a>`, определяющем эту ссылку. Затем оно будет доставлено вместиющим элементам: возможно, элементу `<p>`, элементу `<div>` и самому объекту Document. Иногда удобнее бывает зарегистрировать единственный обработчик события в объекте Document или в другом контейнерном элементе, чем выполнять регистрацию во всех интересующих нас элементах.

Дав определения некоторым терминам, можно перейти к изучению вопросов, связанных с событиями и их обработкой.

Регистрация обработчиков событий

Существует два основных способа регистрации обработчиков событий. Первый, появившийся на раннем этапе развития Всемирной паутины, заключается в установке свойства объекта или элемента документа, являющегося целью события. Второй способ, более новый и более универсальный, заключается в передаче обработчика методу объекта или элемента.

Дело осложняется тем, что каждый прием имеет две версии. Свойство обработчика события можно установить в программном коде на языке JavaScript или в элементе документа, определив соответствующий атрибут непосредственно в разметке HTML. Регистрация обработчиков вызовом метода может быть выполнена стандартным методом с именем `addEventListener()`, который поддерживается всеми браузерами, кроме IE версии 8 и ниже, и другим методом, с именем `attachEvent()`, поддерживаемым всеми версиями IE до IE9.

Установка свойств обработчиков событий

Самый простой способ зарегистрировать обработчик события заключается в том, чтобы присвоить свойству целевого объекта события желаемую функцию обработчика. По соглашению свойства обработчиков событий имеют имена, состоящие из слова «on», за которым следует имя события: `onclick`, `onchange`, `onload`, `onmouseover` и т.д. Обратите внимание, что эти имена свойств чувствительны к регистру и в них используются только строчные символы, даже когда имя типа события состоит из нескольких слов (например «readystatechange»). Ниже приводятся два примера регистрации обработчиков событий:

```

// Присвоить функцию свойству onload объекта Window.
// Функция - обработчик события: она вызывается, когда документ будет
загружен
window.onload = function() {
// Отыскать элемент <form>
var elt = document.getElementById("shipping_address");

// Зарегистрировать обработчик события, который будет вызываться
// непосредственно перед отправкой формы
elt.onsubmit = function() { return validate(this); }
}

```

Такой способ регистрации обработчиков событий поддерживается во всех браузерах для всех часто используемых типов событий. Вообще говоря, все прикладные интерфейсы, получившие широкую поддержку, которые определяют свои события, позволяют регистрировать обработчики установкой свойств обработчиков событий.

Недостаток использования свойств обработчиков событий состоит в том, что они проектировались в предположении, что цели событий будут иметь не более одного обработчика для каждого типа событий. При создании библиотеки для использования в произвольных документах для регистрации обработчиков лучше использовать прием (такой как вызов метода `addEventListener()`), не изменяющий и не затирующий ранее зарегистрированные обработчики.

Установка атрибутов обработчиков событий

Свойства обработчиков событий в элементах документа можно также устанавливать, определяя значения атрибутов в соответствующих HTML-тегах. В этом случае значение атрибута должно быть строкой программного кода на языке JavaScript. Этот программный код должен быть не полным объявлением функции обработчика события, а только ее телом. То есть реализация обработчика события в разметке HTML не должна заключаться в фигурные скобки и предваряться ключевым словом `function`. Например:

```
<button onclick="alert('Спасибо');">Щелкните здесь</button>
```

Если значение HTML-атрибута обработчика события состоит из нескольких JavaScript-инструкций, они должны отделяться точками с запятой либо значение атрибута должно располагаться в нескольких строках.

Некоторые типы событий предназначены для браузера в целом, а не для какого-то конкретного элемента документа. Обработчики таких событий в языке JavaScript регистрируются в объекте `Window`. В разметке HTML они должны помещаться в тег `<body>`, но браузер регистрирует их в объекте `Window`. Ниже приводится полный список таких обработчиков событий, определяемых проектом спецификации HTML5:

<code>onafterprint</code>	<code>onfocus</code>	<code>ononline</code>	<code>onresize</code>
<code>onbeforeprint</code>	<code>onhashchange</code>	<code>onpagehide</code>	<code>onstorage</code>
<code>onbeforeunload</code>	<code>onload</code>	<code>onpageshow</code>	<code>onundo</code>
<code>onblur</code>	<code>onmessage</code>	<code>onpopstate</code>	<code>onunload</code>

onerror

onoffline

onredo

При разработке клиентских сценариев обычно принято отделять разметку HTML от программного кода на языке JavaScript. Программисты, следующие этому правилу, избегают (или, по крайней мере, стараются избегать) использовать HTML-атрибуты обработчиков событий, чтобы не смешивать программный код на языке JavaScript и разметку HTML.

addEventListener()

В стандартной модели событий, поддерживаемой всеми браузерами, кроме IE версии 8 и ниже, целью события может быть любой объект - включая объекты Window и Document и все объекты Elements элементов документа - определяющий метод с именем addEventListener(), с помощью которого можно регистрировать обработчики событий для этой цели.

Метод addEventListener() принимает три аргумента. Первый - тип события, для которого регистрируется обработчик. Тип (или имя) события должен быть строкой и не должен включать префикс «on», используемый при установке свойств обработчиков событий. Вторым аргументом методу addEventListener() передается функция, которая должна вызываться при возникновении события указанного типа. В последнем аргументе методу addEventListener() передается логическое значение. Обычно в этом аргументе передается значение false. Если передать в нем значение true, функция будет зарегистрирована как перехватывающий обработчик и будет вызываться в другой фазе распространения события.

Спецификация со временем может измениться так, что будет допустимо опускать третий аргумент вместо того, чтобы явно передавать в нем значение false, но на момент написания этих строк отсутствие третьего аргумента в некоторых текущих браузерах приводила к ошибке.

Следующий фрагмент регистрирует два обработчика события «click» в элементе <button>. Обратите внимание на различия двух используемых приемов:

```
<button id="mybutton">Щелкните меня!</button>
<script>
```

```
var b = document.getElementById("mybutton");
b.onclick = function() {
  alert("Спасибо, что щелкнули на мне!");
};
```

```
b.addEventListener("click", function() {alert('Еще раз спасибо!')}, false);
</script>
```

Вызов метода addEventListener() со строкой «click» в первом аргументе никак не влияет на значение свойства onclick. Во фрагменте, приведенном выше, щелчок на кнопке приведет к выводу двух диалоговых окон alert(). Но важнее то, что метод addEventListener() можно вызвать несколько раз и зарегистрировать с его помощью несколько функций-обработчиков для одного и того же типа события в том же самом объекте. При появлении

события в объекте будут вызваны все обработчики, зарегистрированные для этого типа события, в порядке их регистрации.

Многократный вызов метода `addEventListener()` для одного и того же объекта с теми же самыми аргументами не дает никакого эффекта - функция-обработчик регистрируется только один раз и повторные вызовы не влияют на порядок вызова обработчиков.

Парным к методу `addEventListener()` является метод `removeEventListener()`, который принимает те же три аргумента, но не добавляет, а удаляет функцию-обработчик из объекта. Это часто бывает удобно, когда необходимо зарегистрировать временный обработчик события, а затем удалить его в какой-то момент.

Internet Explorer версии ниже IE9 не поддерживает методы `addEventListener()` и `removeEventListener()`. В версии IE5 и выше определены похожие методы, `attachEvent()` и `detachEvent()`. Поскольку модель событий в IE не поддерживает фазу перехвата, методы `attachEvent()` и `detachEvent()` принимают только два аргумента: тип события и функцию обработчика, при этом в первом аргументе методам в IE передается имя свойства обработчика с префиксом «он», а не тип события без этого префикса.

Вызов обработчиков событий

После регистрации обработчика событий веб-браузер будет вызывать его автоматически, когда в указанном объекте будет возникать событие указанного типа. В этом разделе подробно описывается порядок вызова обработчиков событий, аргументы обработчиков, контекст вызова (значение `this`) и назначение возвращаемого значения обработчика. К сожалению, некоторые из этих подробностей отличаются между IE версии 8 и ниже и другими браузерами.

Аргумент обработчика событий

При вызове обработчика событий ему обычно (за одним исключением, о котором рассказывается ниже) передается объект события в виде единственного аргумента. Свойства объекта события содержат дополнительную информацию о событии. Свойство `type`, например, определяет тип возникшего события.

В IE версии 8 и ниже обработчикам событий, зарегистрированным установкой свойства, объект события при вызове не передается. Вместо этого объект события сохраняется в глобальной переменной `window.event`. Для переносимости обработчики событий можно оформлять, как показано ниже, чтобы они использовали переменную `window.event` при вызове без аргумента:

```
function handler(event) {  
    event = event || window.event;  
    // Здесь находится реализация обработчика  
}
```

Объект события передается обработчикам событий, зарегистрированным с помощью метода `attachEvent()`, но они также могут использовать переменную `window.event`.

При регистрации обработчика события посредством HTML-атрибута браузер преобразует строку с программным кодом на языке JavaScript в функцию. Браузеры, отличные от IE, создают функцию с единственным аргументом event. В IE создается функция, не принимающая аргументов. Если в таких функциях использовать идентификатор event, он будет ссылаться на window.event. В любом случае обработчики событий, определяемые в разметке HTML, могут ссылаться на объект события, используя идентификатор event.

Контекст обработчиков событий

Когда обработчик событий регистрируется установкой свойства, это выглядит как определение нового метода элемента документа:

```
e.onclick = function() { /* реализация обработчика */ };
```

Поэтому нет ничего удивительного, что обработчики событий вызываются (с одним исключением, касающимся IE, которое описывается ниже) как методы объектов, в которых они определены. То есть в теле обработчика событий ключевое слово this ссылается на цель события.

В обработчиках ключевое слово this ссылается на целевой объект, даже когда они были зарегистрированы с помощью метода addEventListener(). Однако, к сожалению, это не относится к методу attachEvent(): обработчики, зарегистрированные с помощью метода attachEvent(), вызываются как функции, и в них ключевое слово this ссылается на глобальный (Window) объект. Эту проблему можно решить следующим способом:

```
/* Регистрирует указанную функцию как обработчик событий  
указанного типа в указанном
```

```
объекте. Гарантирует, что обработчик всегда будет вызываться как  
метод целевого объекта. */
```

```
function addEvent(target, type, handler) {  
  if (target.addEventListener)  
    target.addEventListener(type, handler, false);  
  else  
    target.attachEvent("on" + type,  
      function(event) {  
        // Вызвать обработчик как метод цели,  
        // и передать ему объект события  
        return handler.call(target, event);  
      });  
}
```

Обратите внимание, что обработчики событий, зарегистрированные таким способом, нельзя удалить, потому что ссылка на функцию-обертку, передаваемую методу attachEvent(), нигде не сохраняется, чтобы ее можно было передать методу detachEvent().

Возвращаемые значения обработчиков

Значение, возвращаемое обработчиком события, зарегистрированным установкой свойства объекта или с помощью HTML-атрибута, следует учитывать. Обычно возвращаемое значение false сообщает браузеру, что он не

должен выполнять действия, предусмотренные для этого события по умолчанию.

Например, обработчик onclick кнопки отправки формы может вернуть false, чтобы предотвратить отправку формы браузером. (Это может пригодиться, если ввод пользователя не прошел проверку на стороне клиента.) Аналогично обработчик события onkeypress поля ввода может фильтровать ввод с клавиатуры, возвращая false при вводе недопустимых символов.

Также важно значение, возвращаемое обработчиком onbeforeunload объекта Window. Это событие генерируется, когда браузер выполняет переход на другую страницу. Если этот обработчик вернет строку, она будет выведена в модальном диалоговом окне, предлагающем пользователю подтвердить свое желание покинуть страницу.

Важно понимать, что учитываются значения, возвращаемые обработчиками событий, только если обработчики зарегистрированы посредством установки свойств. Обработчики, зарегистрированные с помощью addEventListener() или attachEvent() вместо этого должны вызывать метод preventDefault() или устанавливать свойство returnValue объекта события.

Отмена событий

Значение, возвращаемое обработчиком события, зарегистрированным как свойство, можно использовать для отмены действий, выполняемых браузером по умолчанию в случае этого события. В браузерах, поддерживающих метод addEventListener(), отменить выполнение действий по умолчанию можно также вызовом метода preventDefault() объекта события. Однако в IE, версии 8 и ниже, тот же эффект достигается установкой свойства returnValue объекта события в значение false.

В следующем фрагменте демонстрируется обработчик события клика по гиперссылке, который использует все три способа отмены события (блокирует переход пользователя по ссылке):

```
window.onload = function() {  
    // Найти все ссылки  
    var a_href = document.getElementsByTagName('a');  
  
    // Добавить обработчик события click (не для IE<=8)  
    for(var i = 0; i < a_href.length; i++)  
        a_href[i].addEventListener('click', function(e) {e.preventDefault()},  
false);  
};
```

Текущий проект модуля «DOM Events 3» определяет в объекте Event свойство с именем defaultPrevented. Оно пока поддерживается не всеми браузерами, но суть его в том, что при обычных условиях оно имеет значение false и принимает значение true только в случае вызова метода preventDefault().

Отмена действий, по умолчанию связанных с событием, - это лишь одна из разновидностей отмены события. Имеется также возможность остановить распространение события. В браузерах, поддерживающих метод

addEventListener(), объект события имеет метод stopPropagation(), вызов которого прерывает дальнейшее распространение события. Если в том же целевом объекте будут зарегистрированы другие обработчики этого события, то остальные обработчики все равно будут вызваны, но никакие другие обработчики событий в других объектах не будут вызваны после вызова метода stopPropagation().

В IE версии 8 и ниже метод stopPropagation() не поддерживается. Вместо этого объект события в IE имеет свойство cancelBubble. Установка этого свойства в значение true предотвращает распространение события.

Текущий проект спецификации «DOM Events 3» определяет в объекте Event еще один метод - метод с именем stopImmediatePropagation(). Подобно методу stopPropagation(), он предотвращает распространение события по любым другим объектам. Но кроме того, он также предотвращает вызов любых других обработчиков событий, зарегистрированных в том же объекте.

ЛЕКЦИЯ №24. СПОСОБЫ ХРАНЕНИЯ ДАННЫХ.

Когда доходит до сохранения данных приложений локально, разработчики Android определённо избалованы выбором. Помимо прямого доступа к внутренним и внешним областям хранения Android-устройства, платформа Android предлагает базы данных SQLite для хранения реляционных данных и специальные файлы для хранения пар ключ-значение. Более того, приложения для Android также могут использовать сторонние базы данных, которые предлагают поддержку NoSQL.

Как использовать все эти параметры хранения в приложениях для Android. Я также помогу вам понять, как выбрать наиболее подходящий вариант хранения данных.

1. Хранение пар ключ-значение

Если вы ищете быстрый способ хранения нескольких строк или номеров, вам следует рассмотреть возможность использования файла настроек (preferences). Активности и службы Android могут использовать метод getDefaultSharedPreferences() класса PreferenceManager, чтобы сослаться на объект SharedPreferences, который может быть использован и для чтения, и для записи в файл настроек по умолчанию.

```
SharedPreferences myPreferences
=
PreferenceManager.getDefaultSharedPreferences(MyActivity.this);
```

Чтобы начать запись в файл настроек, вы должны вызвать метод edit() объекта SharedPreferences, который возвращает объект SharedPreferences.Editor.

```
SharedPreferences.Editor myEditor =
myPreferences.edit();
```

Объект `SharedPreferences.Editor` имеет несколько интуитивных методов, которые можно использовать для хранения новых пар ключ-значение в файле настроек. Например, вы можете использовать метод `putString()`, чтобы поместить пару ключ-значение со значением типа `String`. Аналогично, вы можете использовать метод `putFloat()`, чтобы поместить пару ключ-значение, чьё значение типа `float`. Следующий пример кода создаёт три пары ключ-значение:

```
myEditor.putString("NAME", "Alice");
myEditor.putInt("AGE", 25);
myEditor.putBoolean("SINGLE?", true);
```

После того, как вы добавили все пары, вы должны вызвать метод `commit()` объекта `SharedPreferences.Editor`, чтобы сохранить их.

```
myEditor.commit();
```

Чтение из объекта `SharedPreferences` гораздо проще. Всё, что вам нужно сделать, так это вызвать соответствующий метод `get*()`. Например, чтобы получить пару ключ-значение, чьё значение является `String`, вы должны вызывать метод `getString()`. Вот фрагмент кода, который извлекает все значения, которые мы добавили ранее:

```
String name = myPreferences.getString("NAME",
"unknown");
int age = myPreferences.getInt("AGE", 0);
boolean isSingle =
myPreferences.getBoolean("SINGLE?", false);
```

Как вы видите из кода выше, второй параметр всех методов `get*()` ожидает значение по-умолчанию, которое является значением, которое должно быть возвращено, если ключ не существует в файле настроек.

Обратите внимание, что файлы настроек ограничены только строками и примитивными типами данных. Если вы хотите сохранить более сложные типы данных или двоичные данные, вы должны выбрать другой вариант хранения.

2. Использование базы данных SQLite

Каждое приложение для Android может создавать и использовать базы данных SQLite для хранения больших объемов структурированных данных. Как вы уже знаете, SQLite не только лёгкая, но и очень быстрая. Если у вас есть опыт работы с системами управления реляционными базами данных и вы знакомы как с SQL, что является сокращением для Structured Query Language, и JDBC, что является сокращением для Java Database Connectivity, это может быть предпочтительным вариантом хранения.

Чтобы создать новую базу данных SQLite или открыть уже существующую, вы можете использовать метод `openOrCreateDatabase()` внутри своей активности или службы. В качестве аргументов вы должны передать имя своей базы данных и режим, в котором вы хотите её открыть. Наиболее часто используемый

режим `MODE_PRIVATE`, который гарантирует, что база данных доступна только для вашего приложения. Например, вот как вы можете открыть или создать базу данных с именем `my.db`:

```
SQLiteDatabase myDB =  
    openOrCreateDatabase("my.db",  
        MODE_PRIVATE, null);
```

После создания базы данных вы можете использовать метод `execSQL()` для запуска SQL-инструкций. В следующем коде показано, как использовать оператор SQL `CREATE TABLE` для создания таблицы названной `user`, которая имеет три столбца:

```
myDB.execSQL(  
    "CREATE TABLE IF NOT EXISTS user (name  
    VARCHAR(200), age INT, is_single INT)"  
);
```

Хотя можно вставить новые строки в таблицу с помощью метода `execSQL()`, лучше использовать метод `insert()`. Метод `insert()` ожидает объект `ContentValues`, содержащий значения для каждого столбца таблицы. Объект `ContentValues` очень похож на объект `Map` и содержит пары ключ-значение.

Вот два объекта `ContentValues`, которые вы можете использовать с таблицей `user`:

```
ContentValues row1 = new ContentValues();  
row1.put("name", "Alice");  
row1.put("age", 25);  
row1.put("is_single", 1);  
ContentValues row2 = new ContentValues();  
row2.put("name", "Bob");  
row2.put("age", 20);  
row2.put("is_single", 0);
```

Как вы могли догадаться, ключи, которые вы передаете методу `put()`, должны соответствовать именам столбцов в таблице.

Когда объекты `ContentValues` готовы, вы можете передать их методу `insert()` вместе с именем таблицы.

```
myDB.insert("user", null, row1);  
myDB.insert("user", null, row2);
```

Чтобы отправить запрос в базу данных, вы можете использовать метод `rawQuery()`, который возвращает объект `Cursor`, содержащий результаты запроса.

```
Cursor myCursor =  
    myDB.rawQuery("select name, age, is_single from  
user", null);
```

Объект Cursor может содержать ноль или несколько строк. Самый простой способ перебрать все его строки, так это вызвать метод moveToNext() внутри цикла while.

Чтобы получить значение отдельного столбца, вы должны использовать такие методы, как getString() и getInt(), которые ожидают индекс столбца. Например, вот как вы получите все значения, вставленные в таблице user:

```
while(myCursor.moveToNext()) {  
    String name = myCursor.getString(0);  
    int age = myCursor.getInt(1);  
    boolean isSingle = (myCursor.getInt(2)) == 1 ?  
true:false;  
}
```

После того, как вы получите все результаты вашего запроса, убедитесь, что вы вызвали метод close() для объекта Cursor, чтобы освободить все ресурсы, которые он хранит.

```
myCursor.close();
```

Аналогичным образом, когда вы закончили все операции с базой данных, не забудьте вызвать метод close() для объекта SQLiteDatabase.

```
myDB.close();
```

3. Использование внутреннего хранилища

Каждое приложение Android имеет свой собственный внутренний каталог хранения, взаимодействующий с ним, в котором приложение может хранить текстовые и двоичные файлы. Файлы внутри этого каталога недоступны для пользователя или других приложений, установленных на устройстве пользователя. Они также автоматически удаляются, когда пользователь удаляет приложение.

Прежде чем вы сможете использовать внутренний каталог хранилища, вы должны определить его местоположение. Для этого вы можете вызвать метод getFilesDir(), который доступен как для активности, так и для служб.

```
File internalStorageDir = getFilesDir();
```

Чтобы получить ссылку на файл внутри каталога, вы можете передать имя файла вместе с указанным вами расположением. Например, вот как вы получите ссылку на файл alice.csv:

```
File alice = new File(internalStorageDir, "alice.csv");
```

С этого момента вы можете использовать свои знания о классах и методах I/O (ввода/вывода) Java для чтения или записи в файл. В следующем фрагменте кода показано, как использовать объект FileOutputStream и метод write() для записи в файл:

```
// Create file output stream  
fos = new FileOutputStream(alice);  
// Write a line to the file
```

```
fos.write("Alice,25,1".getBytes());  
// Close the file output stream  
fos.close();
```

4. Использование внешнего хранилища

Поскольку внутреннее хранилище устройств Android обычно фиксировано и часто довольно ограничено, некоторые устройства Android поддерживают внешние носители данных, такие как съемные микро SD-карты. Я рекомендую использовать этот вариант хранения для больших файлов, таких как фотографии и видео.

В отличие от внутреннего хранилища, внешнее хранилище может не всегда быть доступно. Поэтому, перед его использованием, вы должны всегда проверять, смонтировано ли оно. Для этого используйте метод `getExternalStorageState()` класса `Environment`.

```
if(Environment.getExternalStorageState()  
    .equals(Environment.MEDIA_MOUNTED)) {  
    // External storage is usable  
} else {  
    // External storage is not usable  
    // Try again later  
}
```

Как только вы убедитесь, что внешнее хранилище доступно, вы можете получить путь к файлу внешнего хранилища для своего приложения, вызвав метод `getExternalFilesDir()` и передав `null` в качестве аргумента. Затем вы можете использовать путь для ссылки на файлы внутри каталога. Например, вот как вы можете сослаться на файл с именем `bob.jpg` в каталоге внешнего хранилища приложения:

```
File bob = new File(getExternalFilesDir(null),  
    "bob.jpg");
```

Попросив пользователя предоставить вам разрешение `WRITE_EXTERNAL_STORAGE`, вы можете получить доступ для чтения/записи ко всей файловой системе во внешнем хранилище. Затем вы можете использовать известные общедоступные каталоги для хранения ваших фотографий, фильмов и других мультимедийных файлов. Класс `Environment` предлагает метод `getExternalStoragePublicDirectory()` для определения путей этих общих каталогов.

Например, передав этому методу значение `Environment.DIRECTORY_PICTURES`, вы можете определить путь к общедоступному каталогу, в котором вы можете хранить фотографии. Аналогично, если вы передадите этому методу значение `Environment.DIRECTORY_MOVIES`, вы получите путь к общедоступному каталогу, в котором могут быть сохранены фильмы.

Вот как вы можете сослаться на файл `bob.jpg` в каталоге общих изображений:

```
File bobInPictures = new File(  
    Environment.getExternalStoragePublicDirectory(  
        Environment.DIRECTORY_PICTURES),  
    "bob.jpg"  
);
```

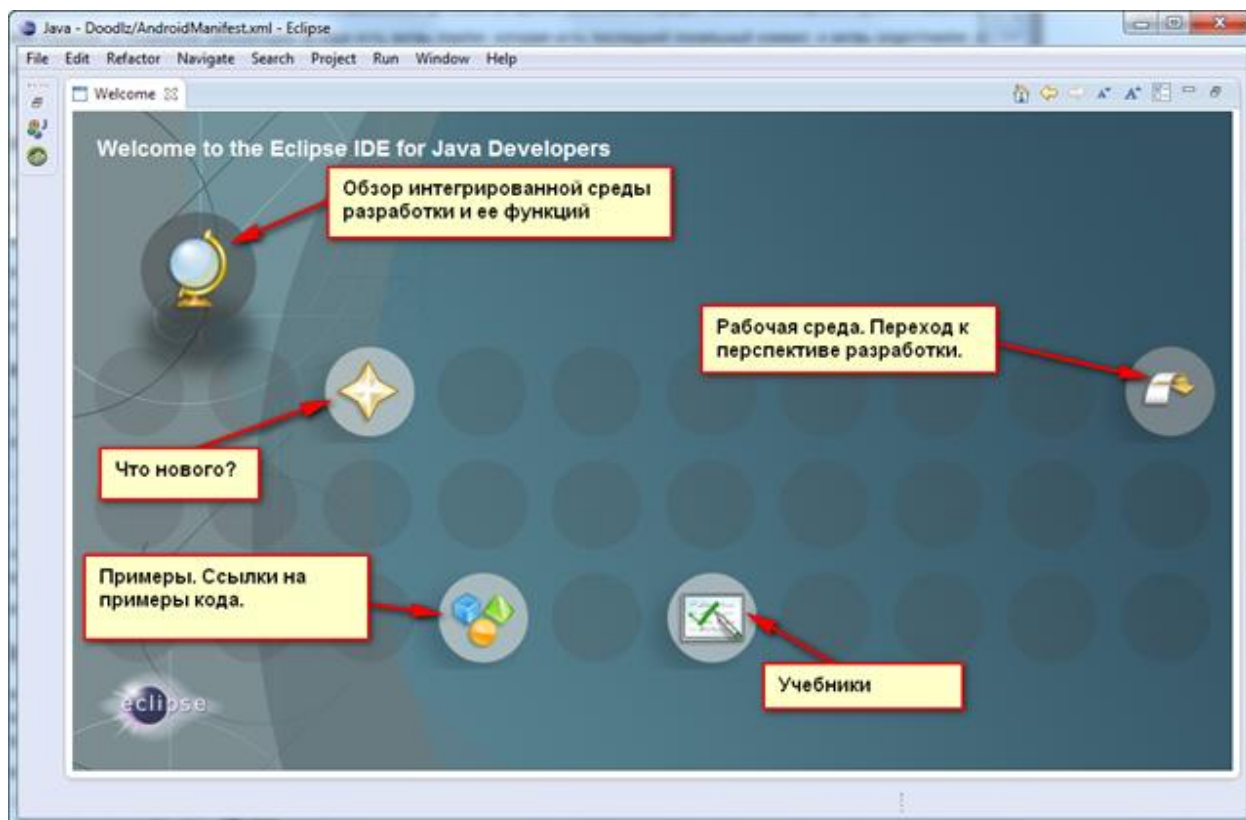
Когда у вас есть объект File, вы можете снова использовать классы FileInputStream и FileOutputStream для чтения или записи на него.

Заключение

Теперь вы знаете, как максимально использовать возможности локального хранилища, предоставляемые Android SDK. Независимо от выбранного вами варианта хранения операции чтения/записи могут потребовать много времени, если задействованы большие объемы данных. Поэтому, чтобы убедиться, что основной поток пользовательского интерфейса всегда остается отзывчивым, вы должны рассмотреть возможность запуска операций в другом потоке.

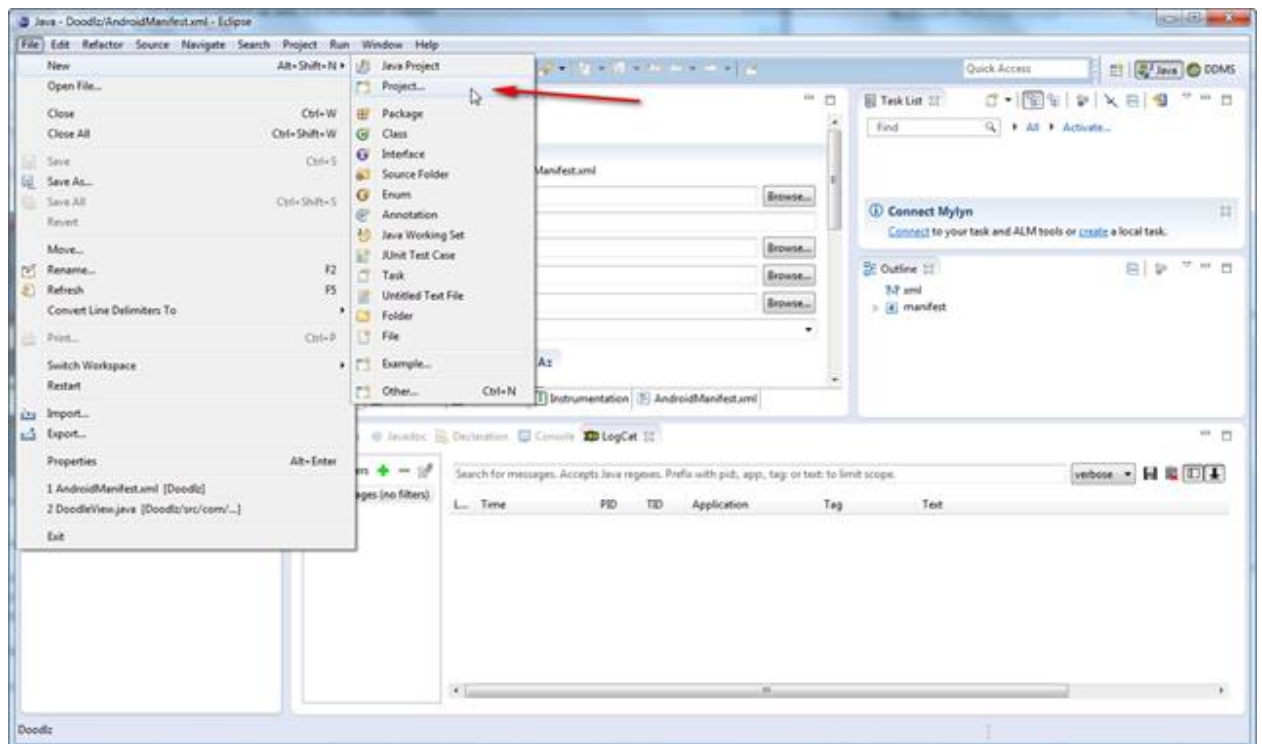
ЛЕКЦИЯ №25-№26. СТРУКТУРА ПРОЕКТА ANDROID. КАТАЛОГИ РЕСУРСОВ. ФАЙЛ R.JAVA.

В интегрированной среде разработки Eclipse можно управлять, редактировать, компилировать, выполнять и отлаживать приложения. Подключаемый модуль ADT Plugin для Eclipse предоставляет пользователям дополнительные инструменты, применяемые для разработки Android-приложений. Можно также воспользоваться подключаемым модулем ADT Plugin для управления различными версиями платформы Android. Это требуется в случае разработки приложений, предназначенных для различных устройств, на которых установлены различные версии операционной системы Android. После первого запуска среды Eclipse появится вкладка Welcome(Добро пожаловать):

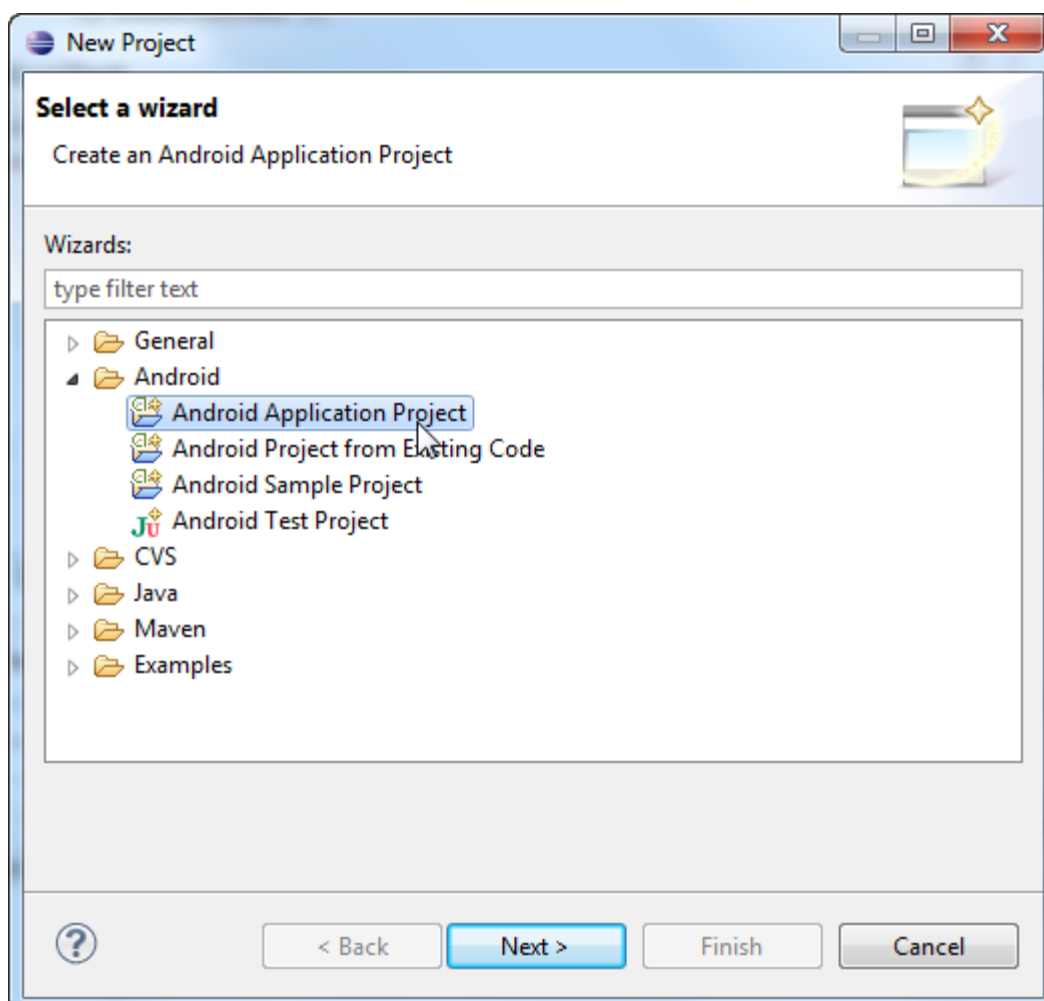


На этой вкладке находится ряд ссылок-пиктограмм щелкнув на которые вы попадете в соответствующие разделы. Щелкните на кнопке Workbench(Рабочая среда), чтобы отобразить перспективу разработки Java, в которой начинается разработка Android-приложений. В Eclipse поддерживается разработка приложений на нескольких языках программирования. Каждый установленный набор инструментов Eclipse представлен отдельной перспективой разработки.

Создание нового проекта
 Чтобы начать программировать на Android в Eclipse, выполните команды File->NewProject... (Файл->НовыйПроект...) для отображения диалогового окна New Project(Новый проект)



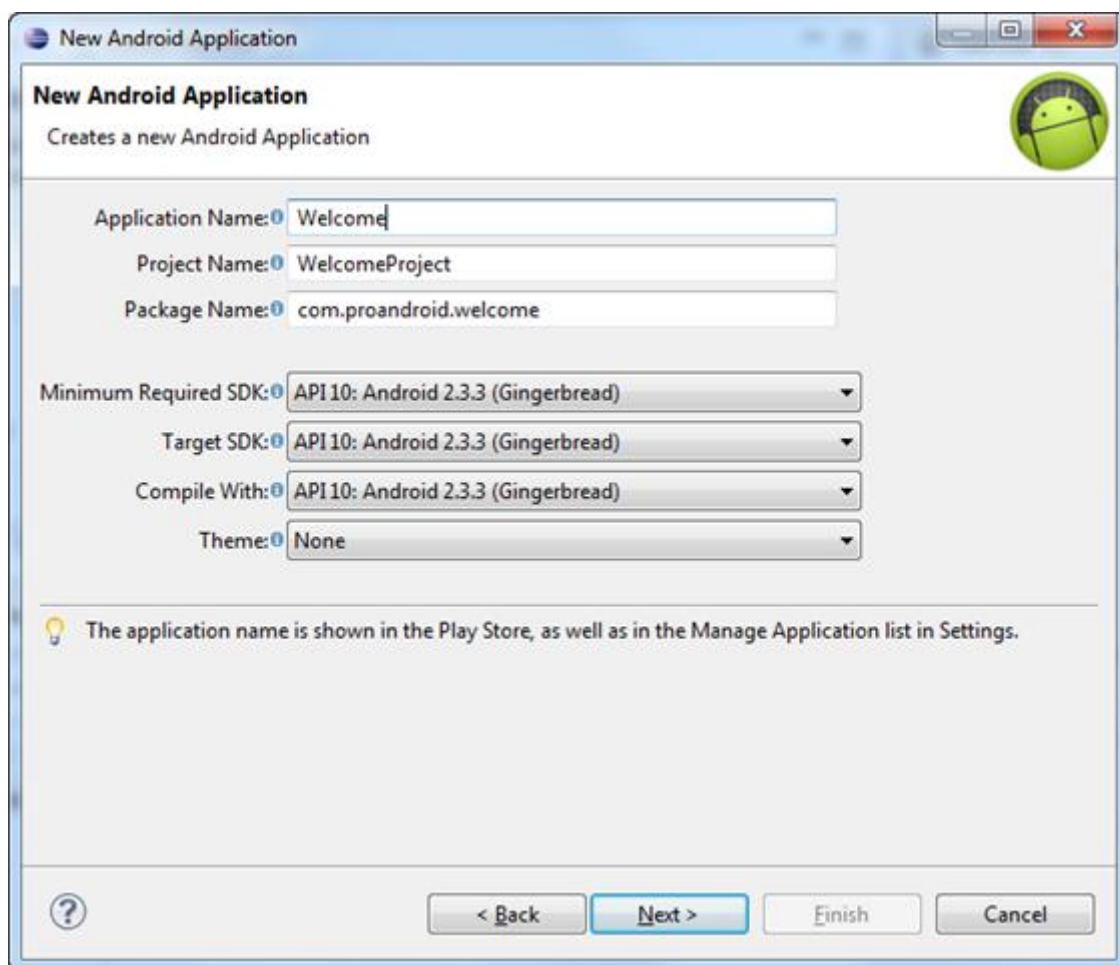
Это окно также можно отобразить с помощью раскрывающегося списка New(Создать) панели инструментов.



Раскройте узел Android, выберите параметр Android Application Project (Проект Приложения Android) и щелкните на кнопке Next>(Далее>). На экране появится диалоговое окно New Android Project(Новый проект Android)

Проект— это группа связанных файлов, например файлы кода и изображения, которые формируют приложение. В диалоговом окне New Projectможно создать проект «с нуля» либо воспользоваться существующим исходным кодом (например кодом из книги или примеров поставляемых с Android SDK).

Кстати сказать, окошко New Android Application сильно поменялось, начиная с названия и кончая содержанием. Так что будем разбираться по ходу пьесы. И так поехали! Заполняем поля как показано на скриншоте



Application Name (Имя Приложения) – это имя которое будет отображаться в PlayStore, а так же в диспетчере приложений на устройстве, на которое будет установлено приложение.

Project Name (Имя Проекта) – имя проекта используемое только в Eclipse. Должно быть уникальным в рабочей среде. Обычно оно такое же как и Application Name, но я сделал другим чтобы стало яснее где какое имя будет

отображаться.

Package Name (Имя Пакета) – это имя ДОЛЖНО БЫТЬ УНИКАЛЬНЫМ ИДЕНТИФИКАТОРОМ вашего приложения. То есть вы должны быть уверены что ни кто так свое приложение больше не назовет. И это имя должно оставаться одним и тем же на протяжении всего жизненного цикла приложения (версии приложения могут меняться, но имя должно оставаться одним и тем же).

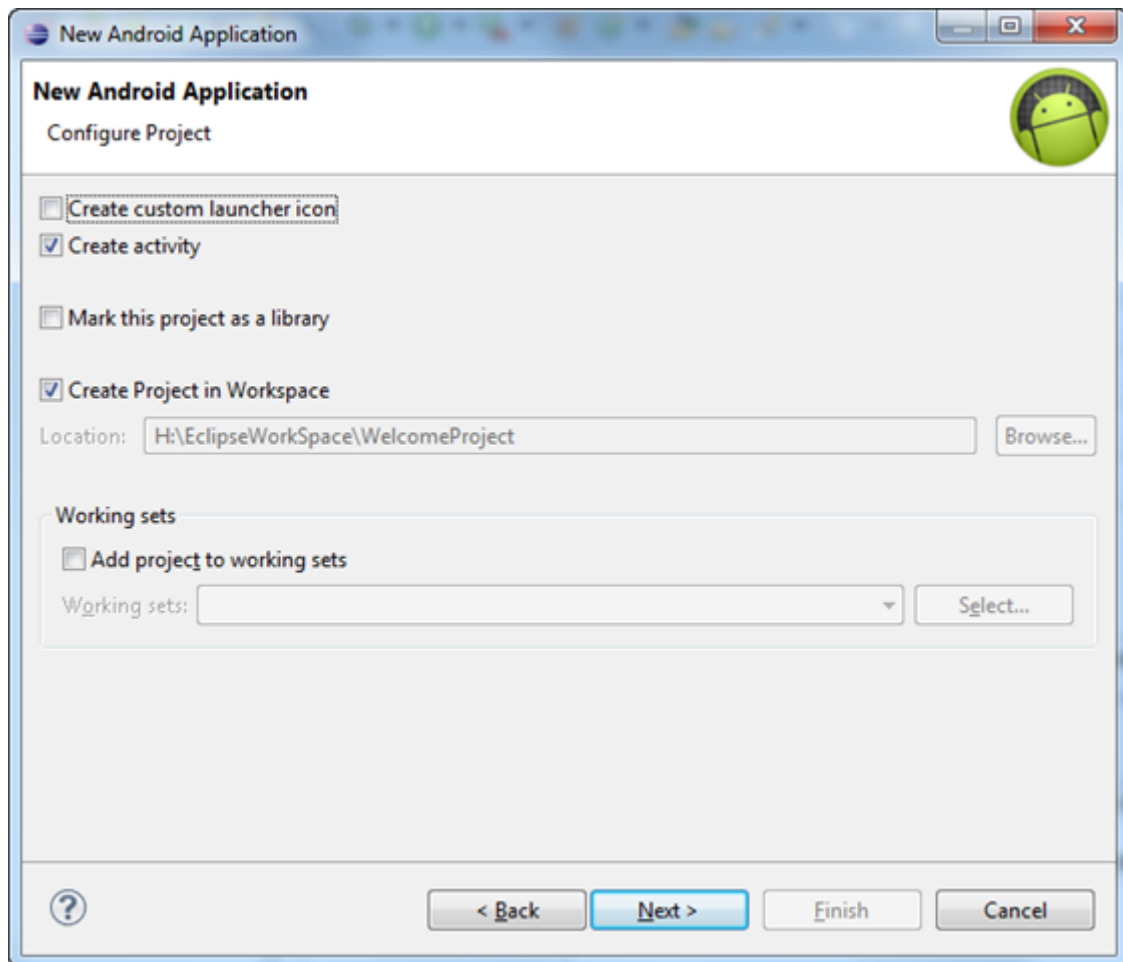
Minimum Required SDK – минимальная версия Android необходимая для функционирования вашего приложения. Это означает по существу минимальную версию SDK, которая необходима, чтобы приложение нормально работало. И кроме того, Андроид просто не позволит установить приложение, если на устройстве стоит версия Андроид у которой номер API меньший, чем указанный в этом параметре.

Target SDK – целевая или максимальная версия Android на которой приложение будет точно работать и было протестировано. Рекомендуется устанавливать этот параметр как можно в более высокое значение.

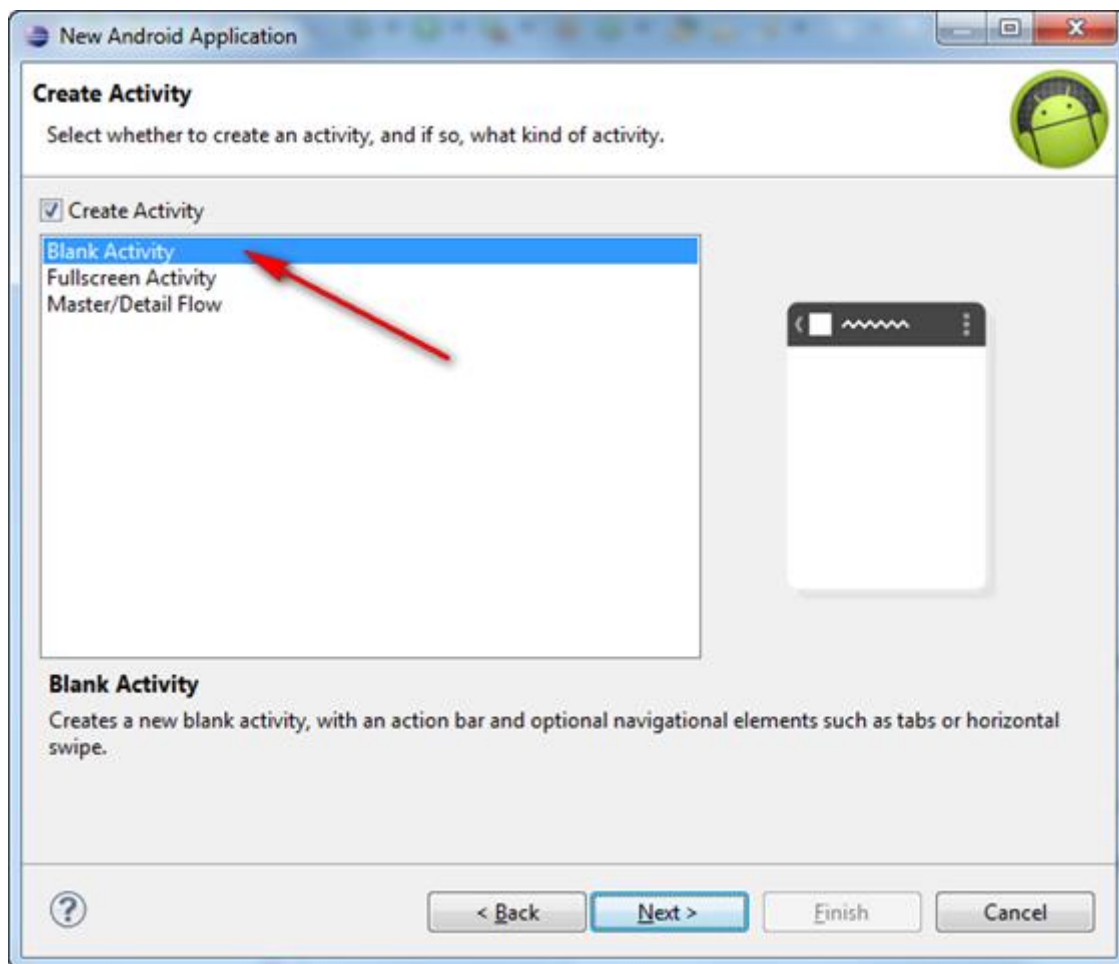
Compile SDK – версия SDK из установленных вами с помощью которой будет компилироваться приложение.

Theme – тема оформления приложения. В принципе, подробное описание всех этих пунктов есть на буржуйском языке и доступно при наведении мышки на синенький значек с буквой i.

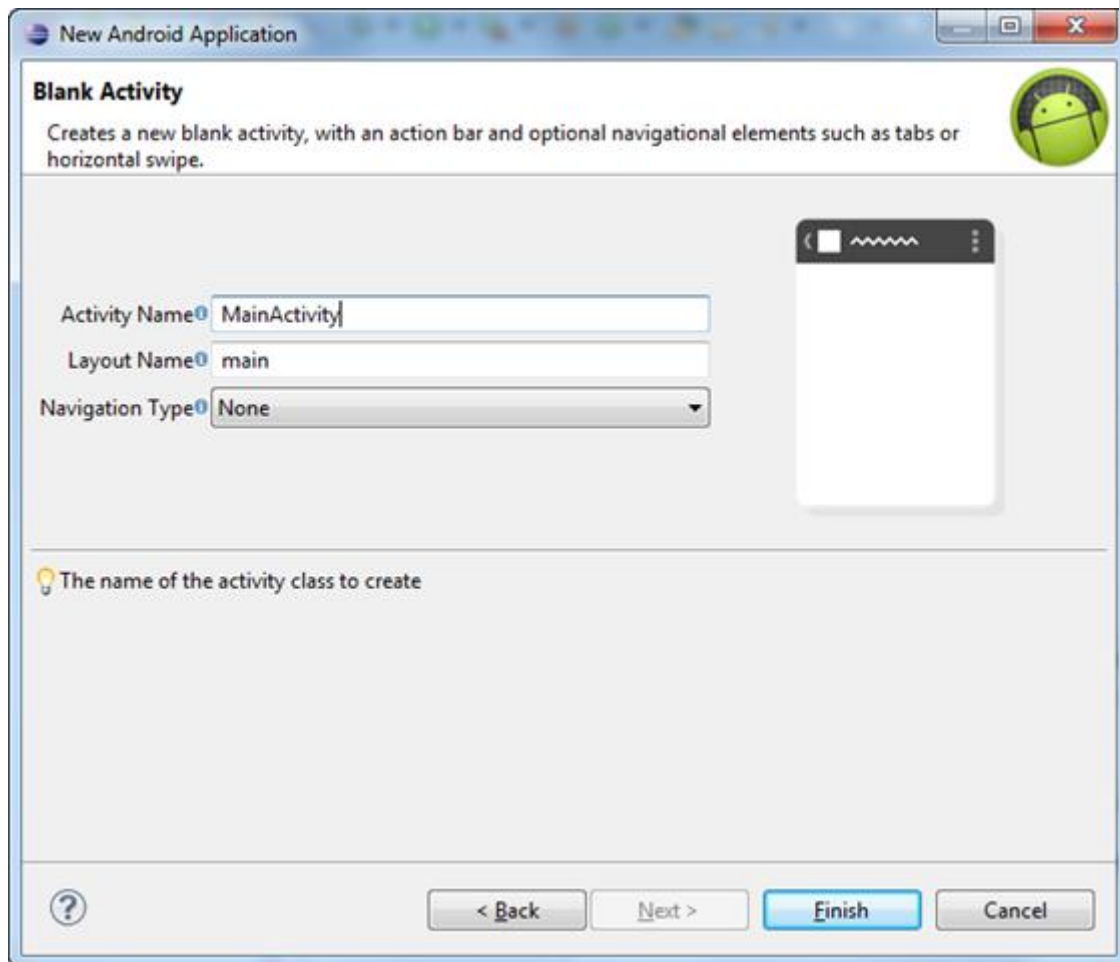
И так жмем Next и видим следующее окошко



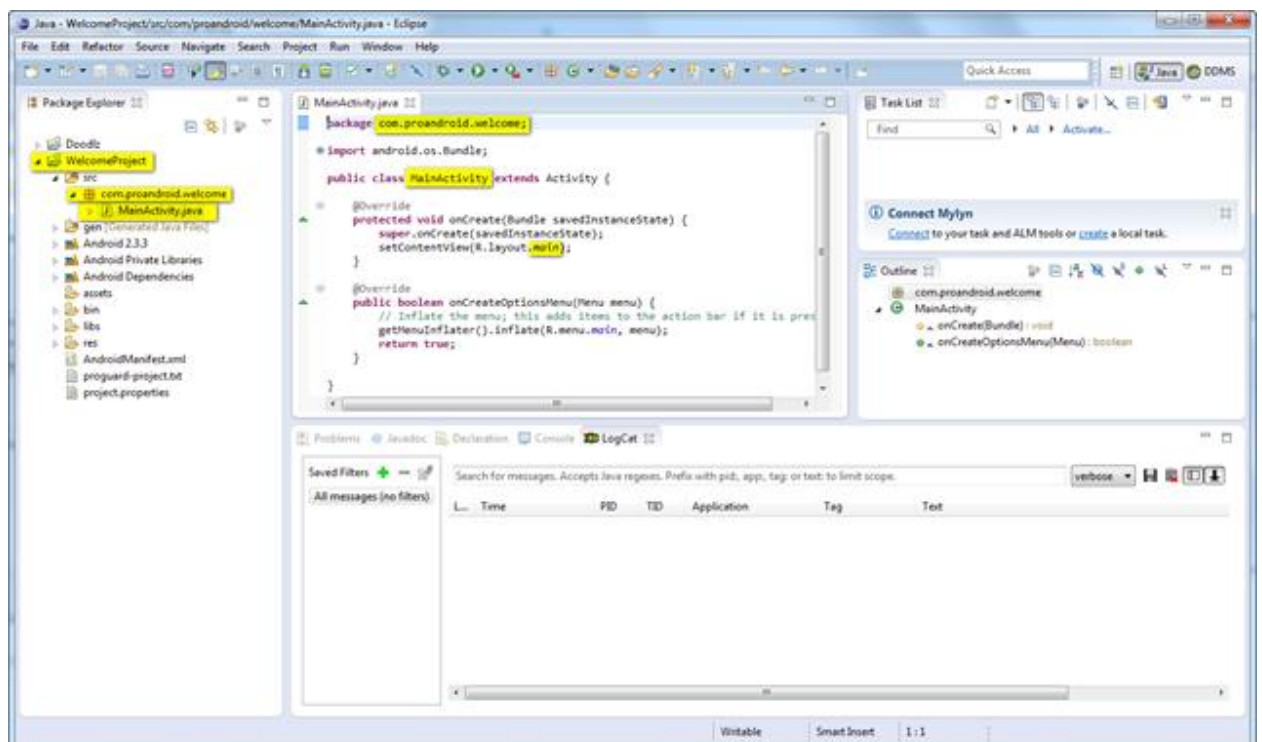
убираем галку с Create custom launcher icon и жмем Next



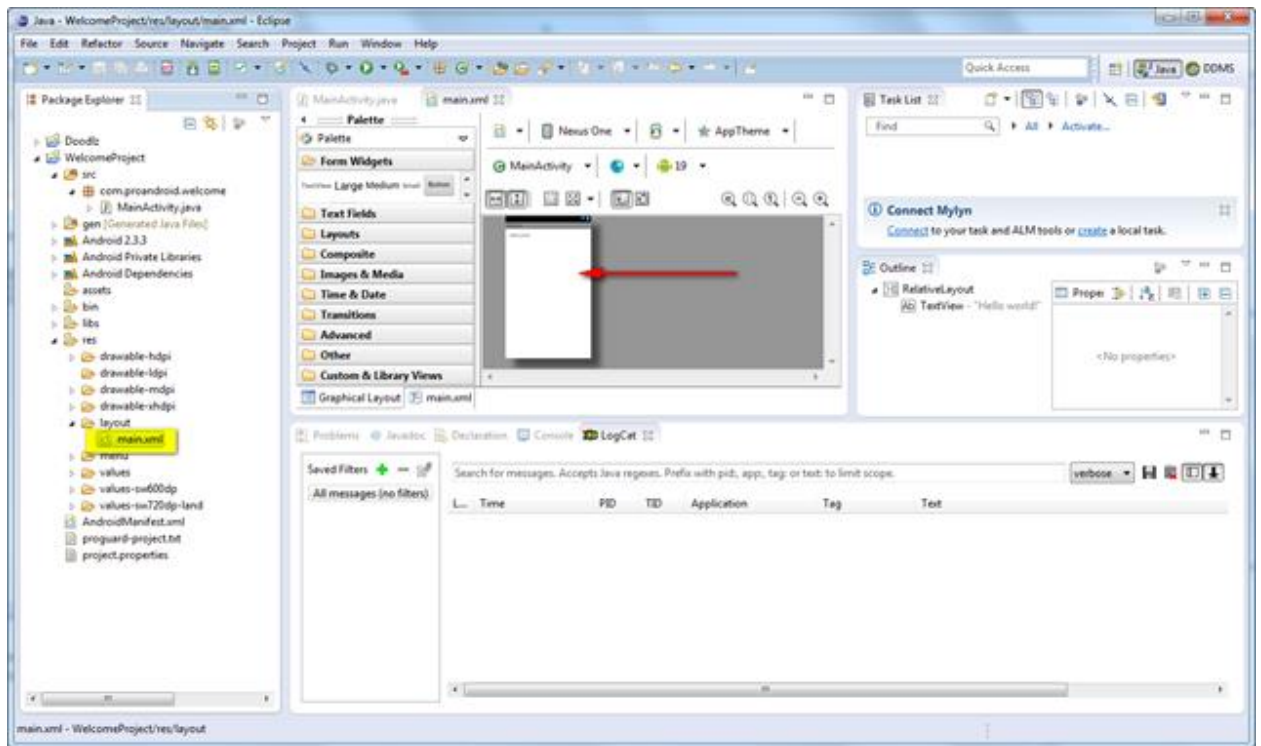
выбираем Blank Activity и ждем Next и видим следующее окошко



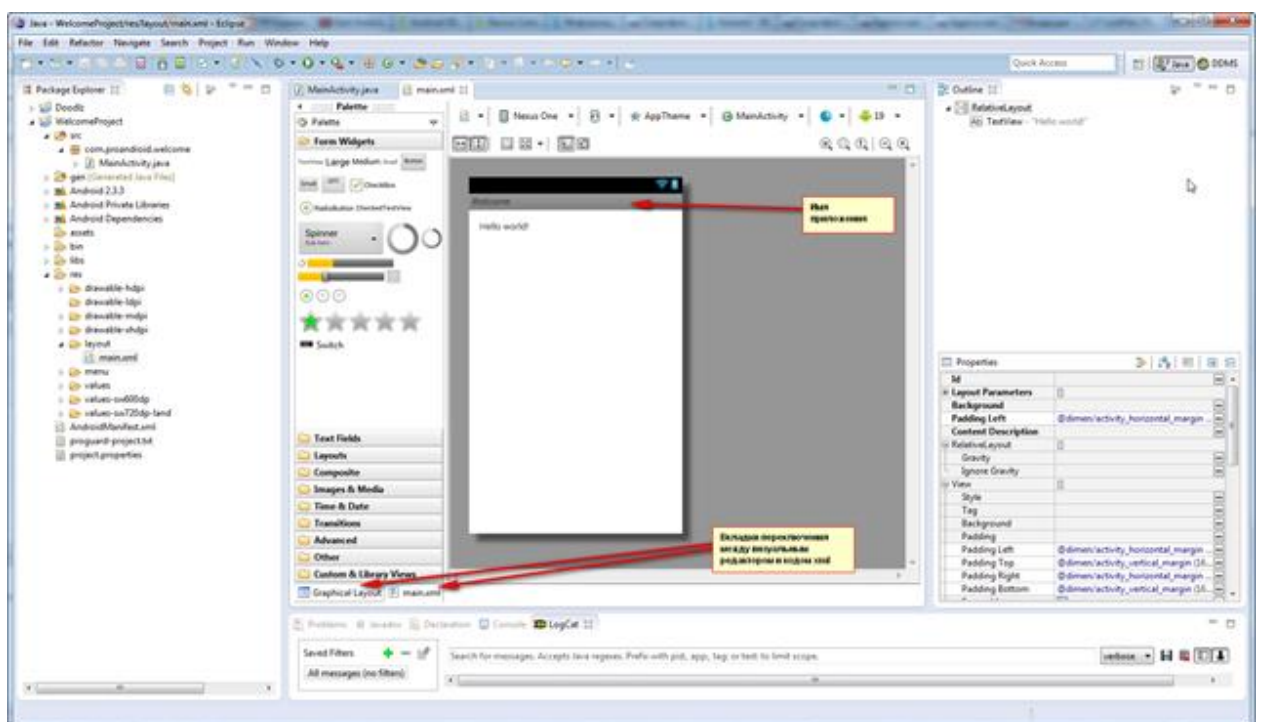
заполняем как на экране и жмем Finish и видим перспективу разработки проекта Java



Желтым я подсветил то, на что стоит обратить особое внимание, чтобы стало понятно куда какие названия прилепились. Далее раскрываем узел, затем узел layout и дважды кликаем на main.xml и о чудо! видим как будет выглядеть наше приложение! Щелкнув оп main.xml мы вызвали редактор ADT Visual Layout Editor. С его помощью можно создать графический интерфейс пользователя путем перетаскивания в окно приложения компонентов GUI, таких как Buttons, TextViews, ImageViews и др.



Немного перенастроим нашу среду чтобы все было получше видно



В узле проекта содержимое проекта организовано в виде различных файлов и папок, в том числе:

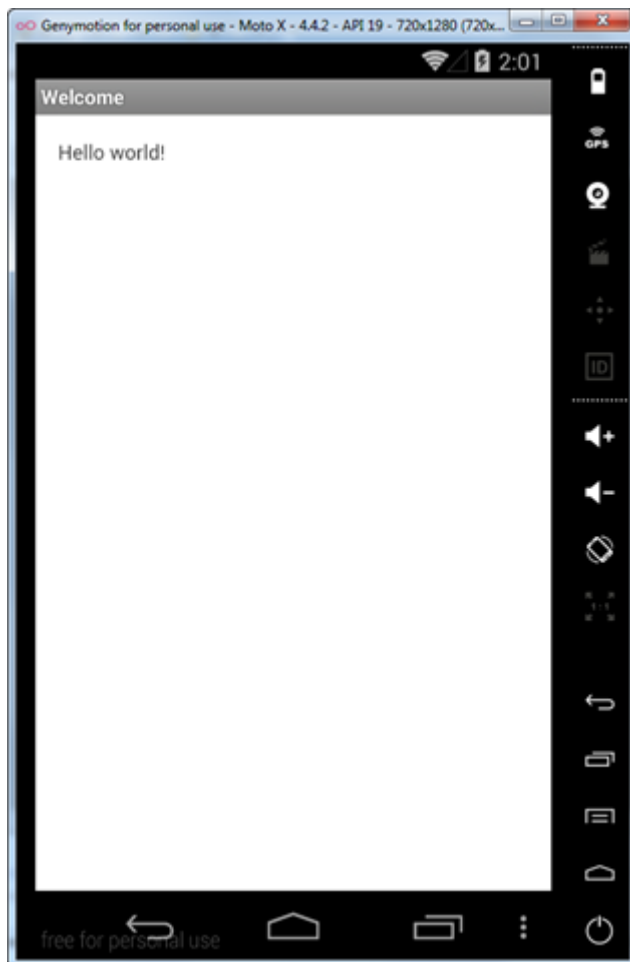
- src— папка, включающая исходные файлы проекта Java;
- gen— папка, содержащая файлы Java, сгенерированные IDE;
- Android 2.3.3— папка, включающая версию Android framework, выбранную при создании приложения;
- res— папка, в которой находятся файлы ресурсов, связанные с приложением, такие как макеты GUI и изображения, используемые в приложении.

другие папочки рассмотрим позже.

При создании Android-приложения с помощью Eclipse макет интерфейса пользователя хранится в XML-файле, который по умолчанию называется main.xml. Благодаря определению элементов GUI в XML-файле можно легко отделить логику приложения от его презентации. Файлы макетов являются ресурсами приложения и хранятся в папке resпроекта. Макеты GUI находятся в подпапке папки layout.

Чтобы просмотреть XML-содержимое файла, выберите вкладку с именем файла макета (в данном случае, main.xml). Чтобы вернуться обратно к представлению Visual Layout Editor, выберите вкладку Graphical Layout(Графический макет).

В принципе это приложение можно уже запустить на исполнение. Я запустил его используя Genymotion, так как это гораздо быстрее чем эмулятор Андроид из SDK и даже Intel.

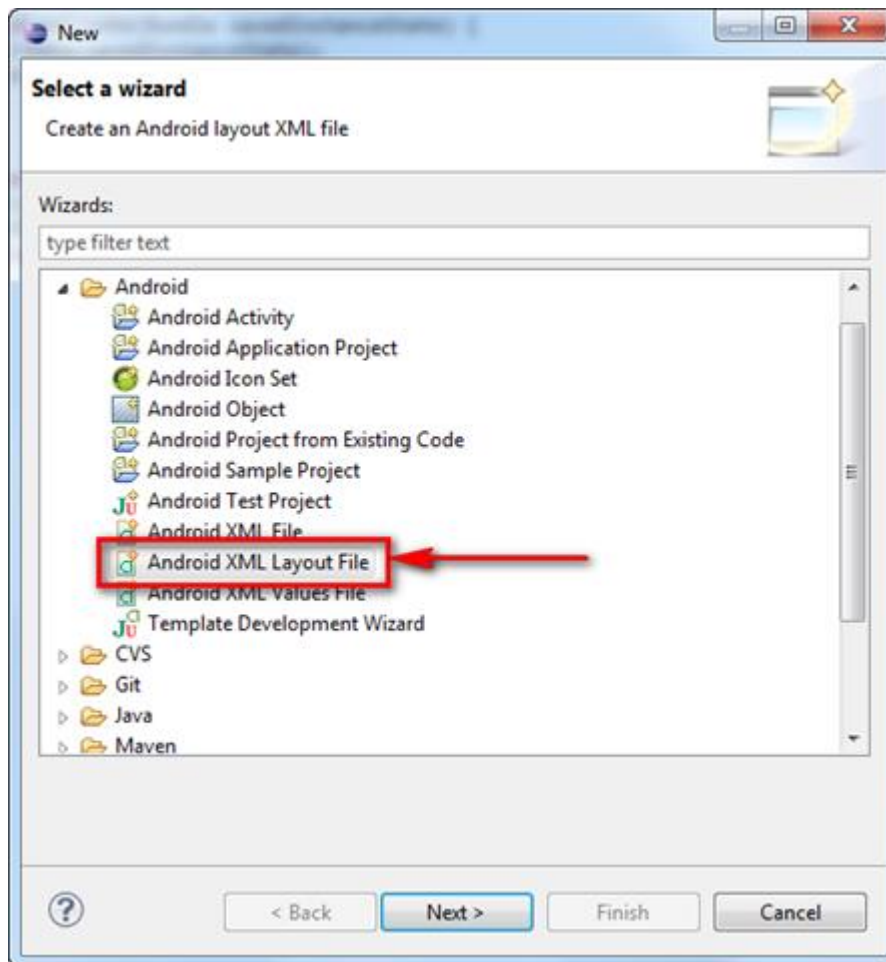


Полезное видео по данной теме

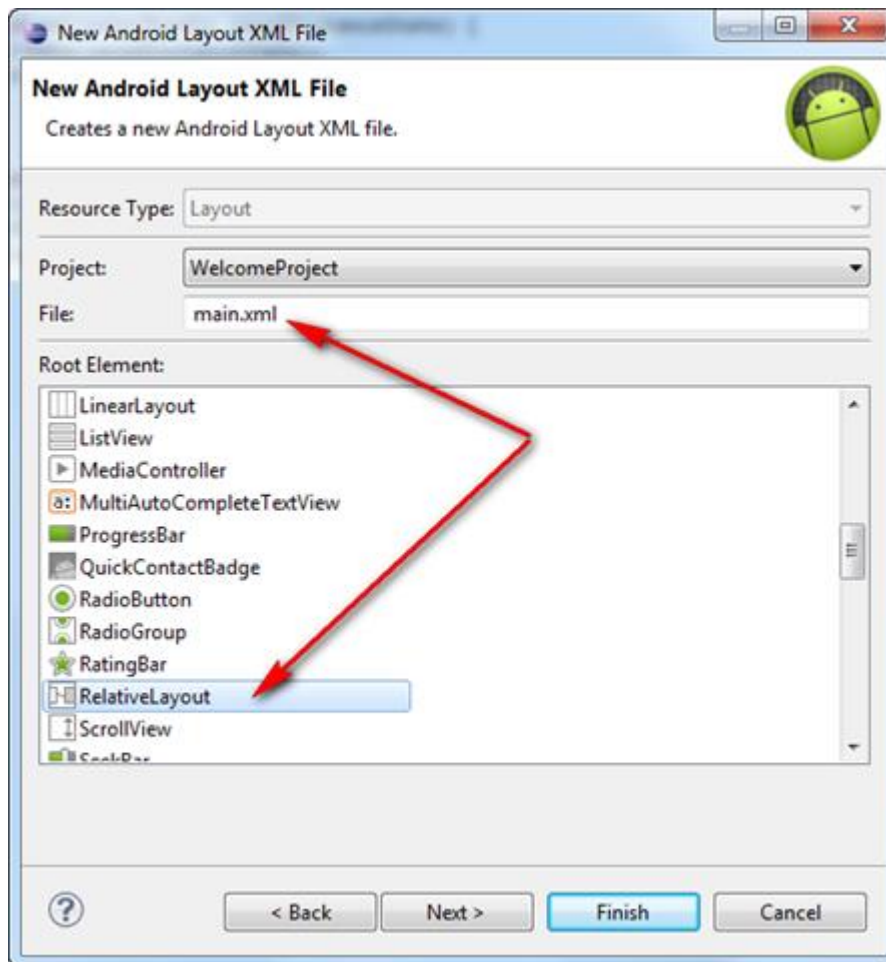
И так продолжаем эксперименты с первым нашим проектом.

Удаление и воссоздание файла `main.xml`
В процессе создания приложения для этой главы вместо заданного по умолчанию файла `main.xml` будет использоваться новый файл макета, `RelativeLayout`(в этом файле определяется относительное расположение компонентов). Чтобы заменить заданный по умолчанию файл `main.xml`, выполните следующие действия:

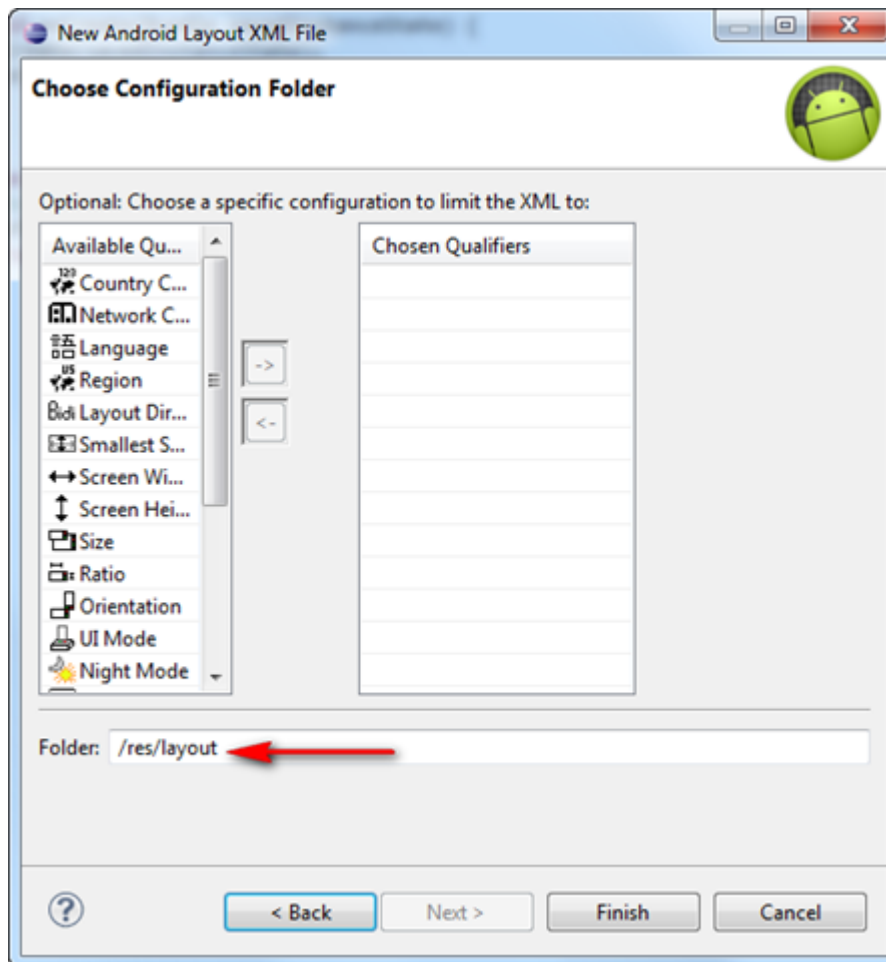
- 1) Убедитесь в том, что файл `main.xml` закрыт, потом щелкните на нем правой кнопкой мыши (в папке `/res/layout` проекта) и выберите команду `Delete`(Удалить) для удаления файла.
- 2) В узле `Android` выберите параметр `Android XML Layout File` и жмите `Next`



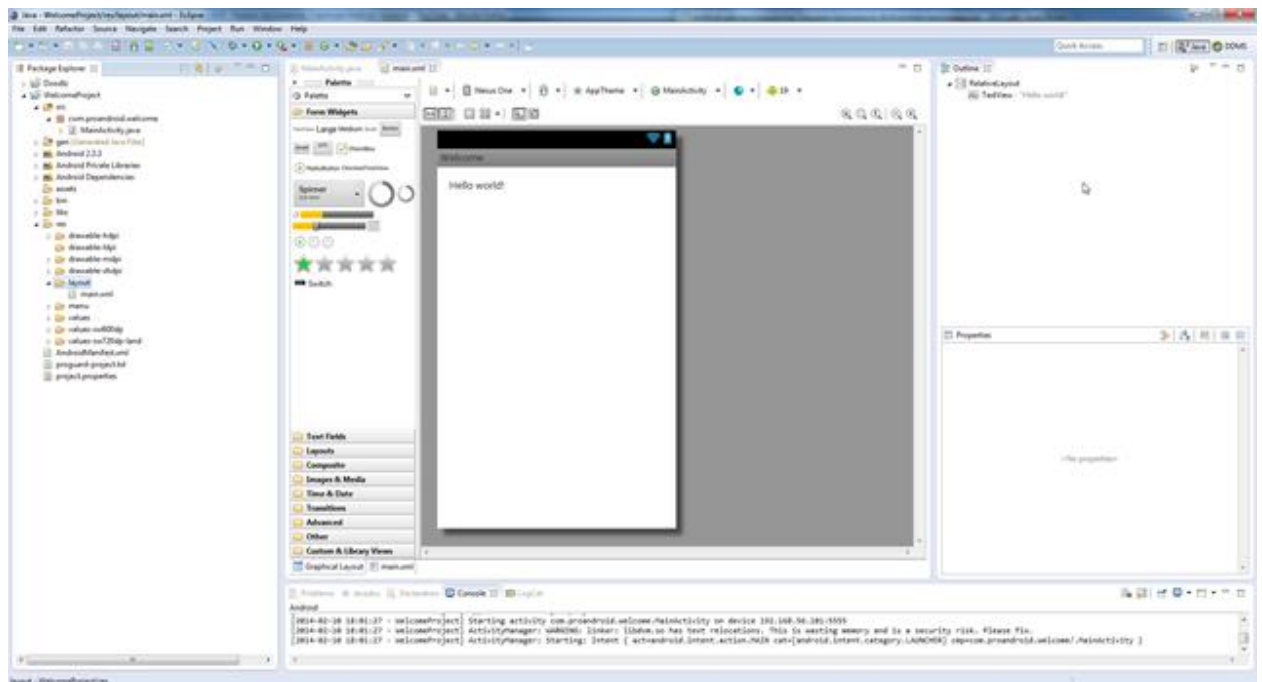
3) Далее указываем имя создаваемого файла main.xml и выбираем RelativeLayout и жмем Next



4) Удостоверяемся что файл создается в каталоге /res/layout

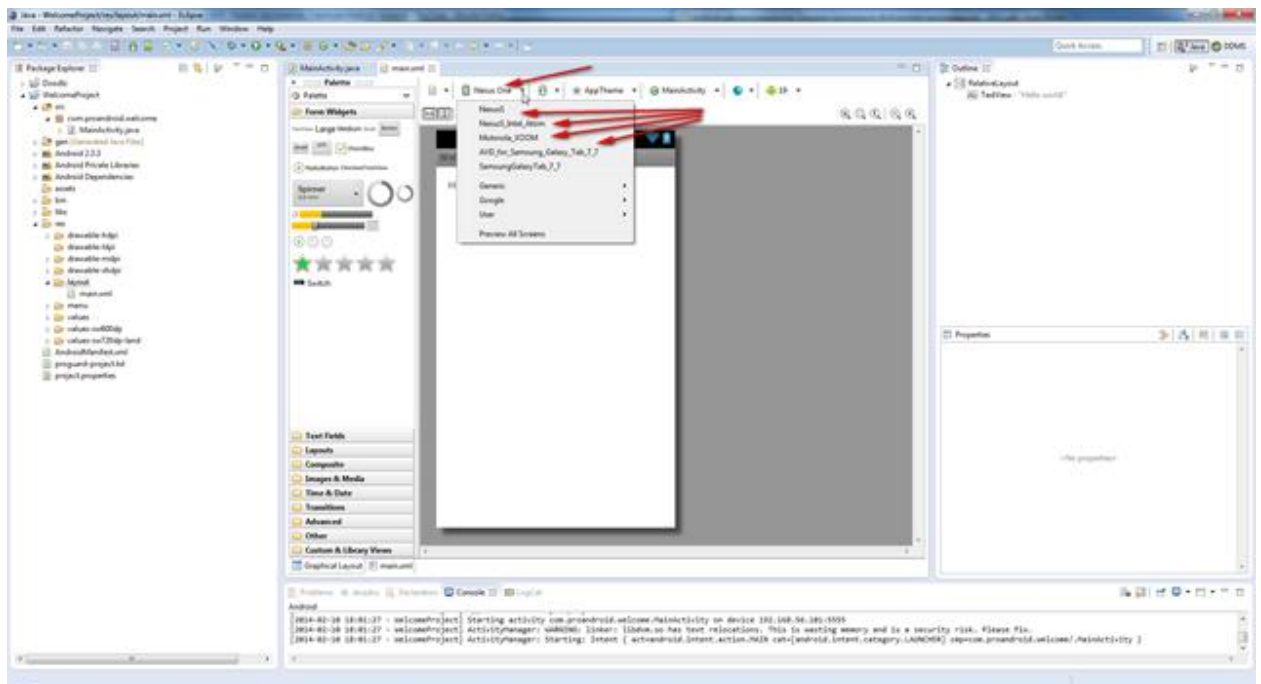


и жмем Finish и попадаем в уже знакомое окошко



Настройка размера и разрешения экрана в Visual Layout Editor
Поскольку операционная система Android может выполняться на множестве различных устройств, в комплект поставки Visual Layout Editor входит ряд

различных конфигураций устройств, представляющих разные размеры и разрешения экрана. Эти на-стройки выбираются в раскрывающемся списке Device Configurations (Конфигурации устройства)



Если стандартные конфигурации не соответствуют устройству, для которого разрабатывается приложение, можно создать собственную конфигурацию устройства «с нуля» либо путем копирования и изменения одной из имеющихся конфигураций.

Для создания примеров в книге мы использовали основное тестовое устройство Samsung Nexus S, которое снабжено 4-дюймовым экраном с разрешением 480×800 пикселей (WVGA). Рекомендуется разрабатывать масштабируемую библиотеку Android GUI, которая позволит корректно формировать изображения на экранах различных устройств. Благодаря свойству масштабируемости не требуется точное соответствие области проектирования Visual Layout Editor экранам физических устройств. Достаточно выбрать подобную конфигурацию устройства.

Размеры и разрешение изображений и экрана

Поскольку экраны устройств Android имеют различные размеры, разрешения и пиксельные плотности экрана (выражается в точках на дюйм или DPI), поддерживаются изображения (и другие ресурсы), которые выбираются операционной системой в соответствии с пиксельной плотностью экрана устройства. Поэтому в папке res проекта находятся четыре подпапки: drawable-xhdpi (сверхвысокая плотность), drawable-hdpi (высокая плотность), drawable-mdpi (средняя плотность), drawable-ldpi (низкая плотность).

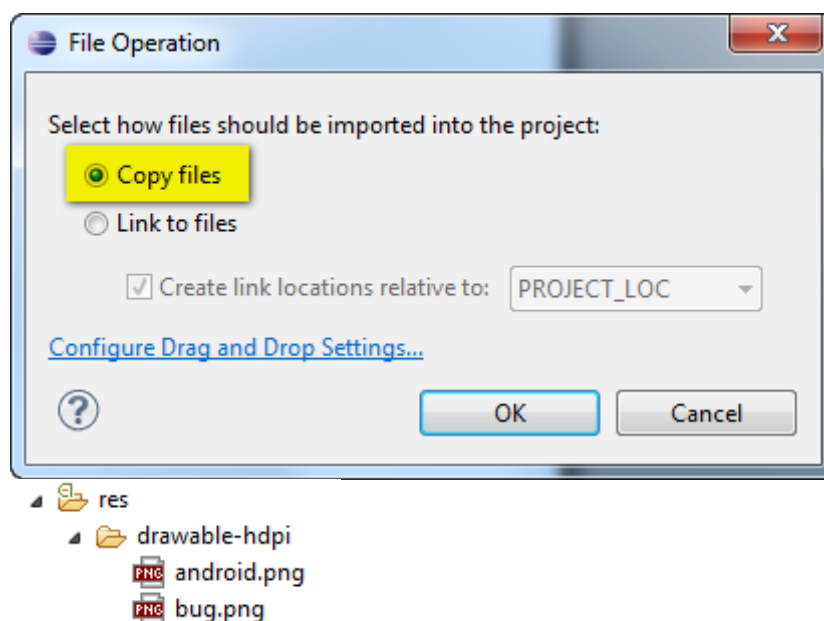
Пиксельные плотности устройств Android

xhđpi	Сверхвысокая плотность	—	примерно 320 точек	на дюйм
hđpi	Высокая плотность	—	примерно 240 точек	на дюйм
mdpi	Средняя плотность	—	примерно 160 точек	на дюйм
ldpi	Низкая плотность	—	примерно 120 точек	на дюйм
nodpi Отсутствие масштабирования ресурсов независимо от плотности экрана				

Шаг 1. Добавление изображений в проект

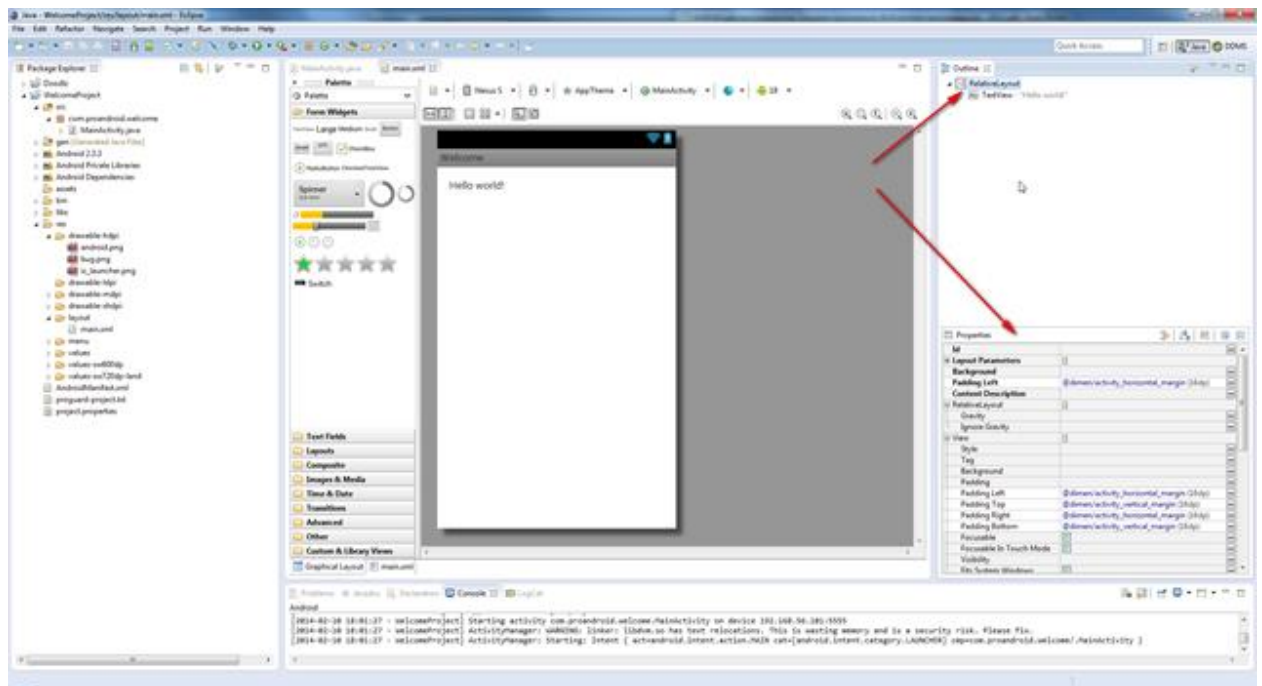
Для данного приложения в проект нужно добавить изображения Deitel bug (bug.png) и логотипа Android (android.png). Эти изображения находятся в папке images вместе с примерами книги. Чтобы добавить изображения в проект, выполните следующие действия:

1. В окне Package Explorer откройте папку проекта res.
2. Найдите и откройте папку images, которая находится среди примеров книги, потом перетащите изображения в папку, которая находится в подпапке drawable-hdpi папки res.



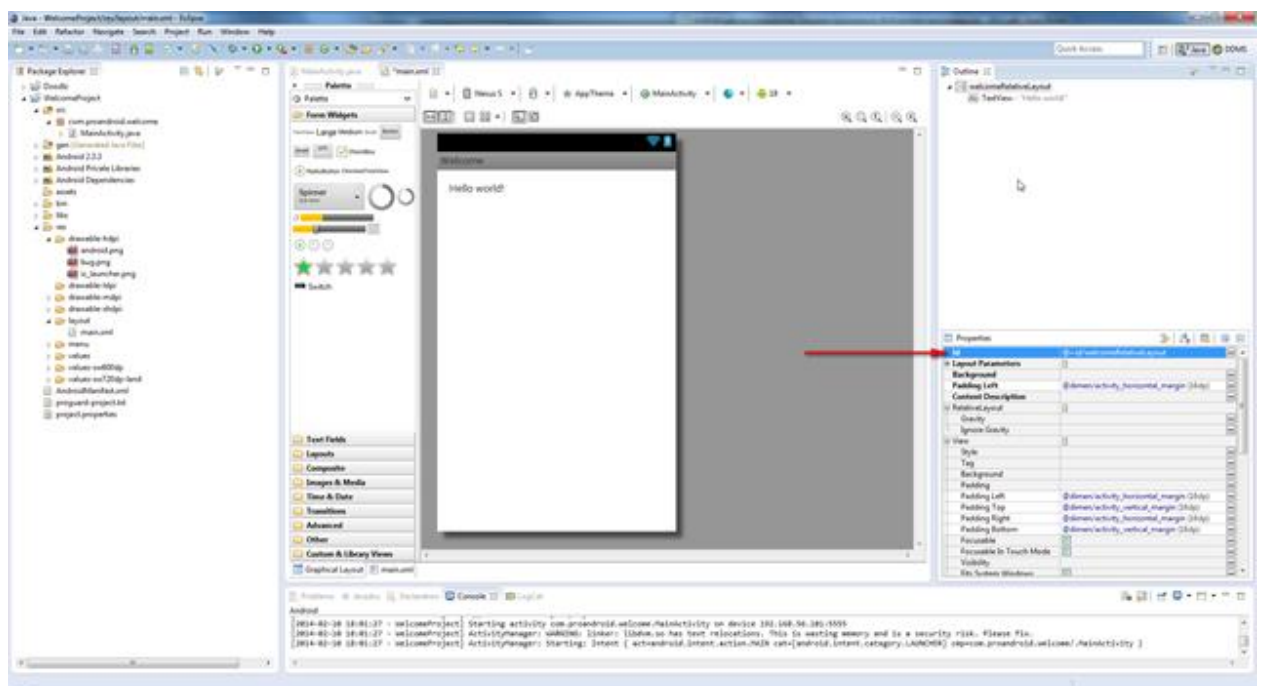
Теперь изображения могут использоваться в приложении.

Шаг 2. Изменения свойства Id макета RelativeLayout
С помощью окна Properties(Свойства) можно сконфигурировать свойства выбранного макета или компонента без непосредственного редактирования XML-кода. Если окно Properties не отображается, отобразите его двойным щелчком на пункте RelativeLayout в окне Outline(Структура).



Выберите компонент RelativeLayout, затем, выполнив прокрутку в окне Properties, выберите свойство Id и присвойте ему значение:

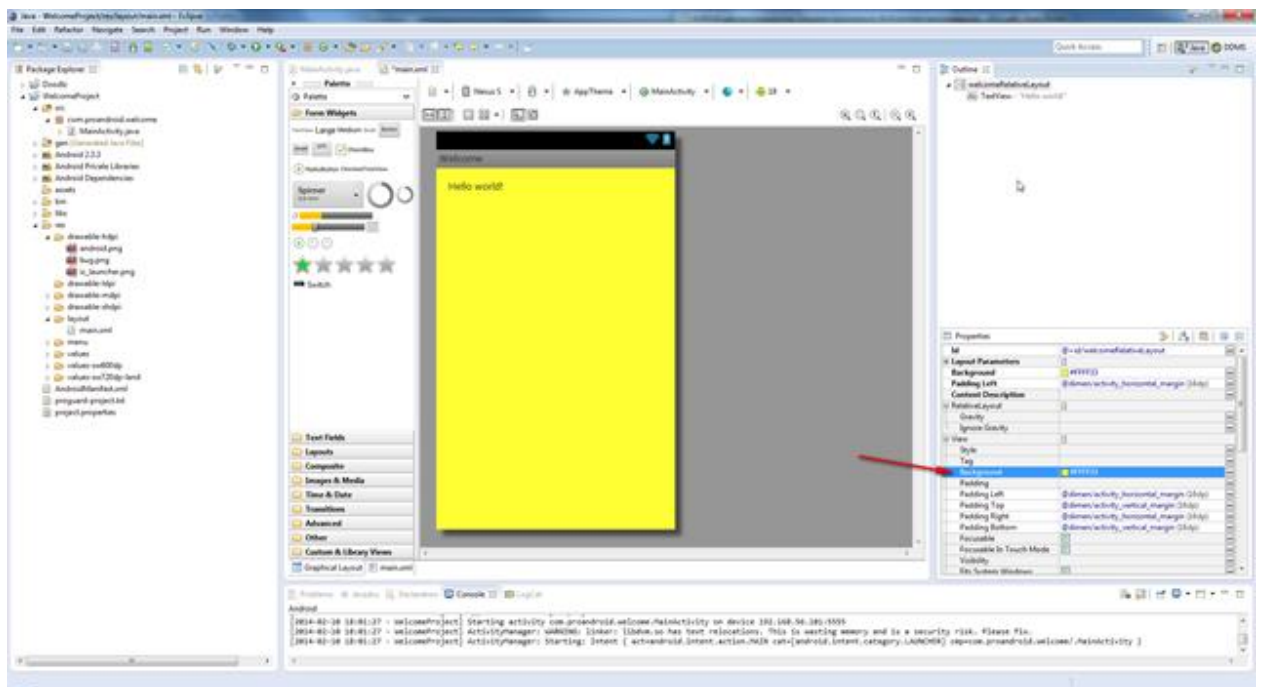
@+id/welcomeRelativeLayout



Имя каждого объекта определяется с помощью его свойства Id. Это свойство может применяться для получения доступа и изменения компонента, при этом не нужно знать точное местоположение компонента в XML-коде. Как будет показано позднее, свойство Id также может применяться для указания относительного позиционирования компонентов в макете RelativeLayout.

Символ + в конструкции @+id определяет создание нового идентификатора (имя переменной), который указан справа от символа /

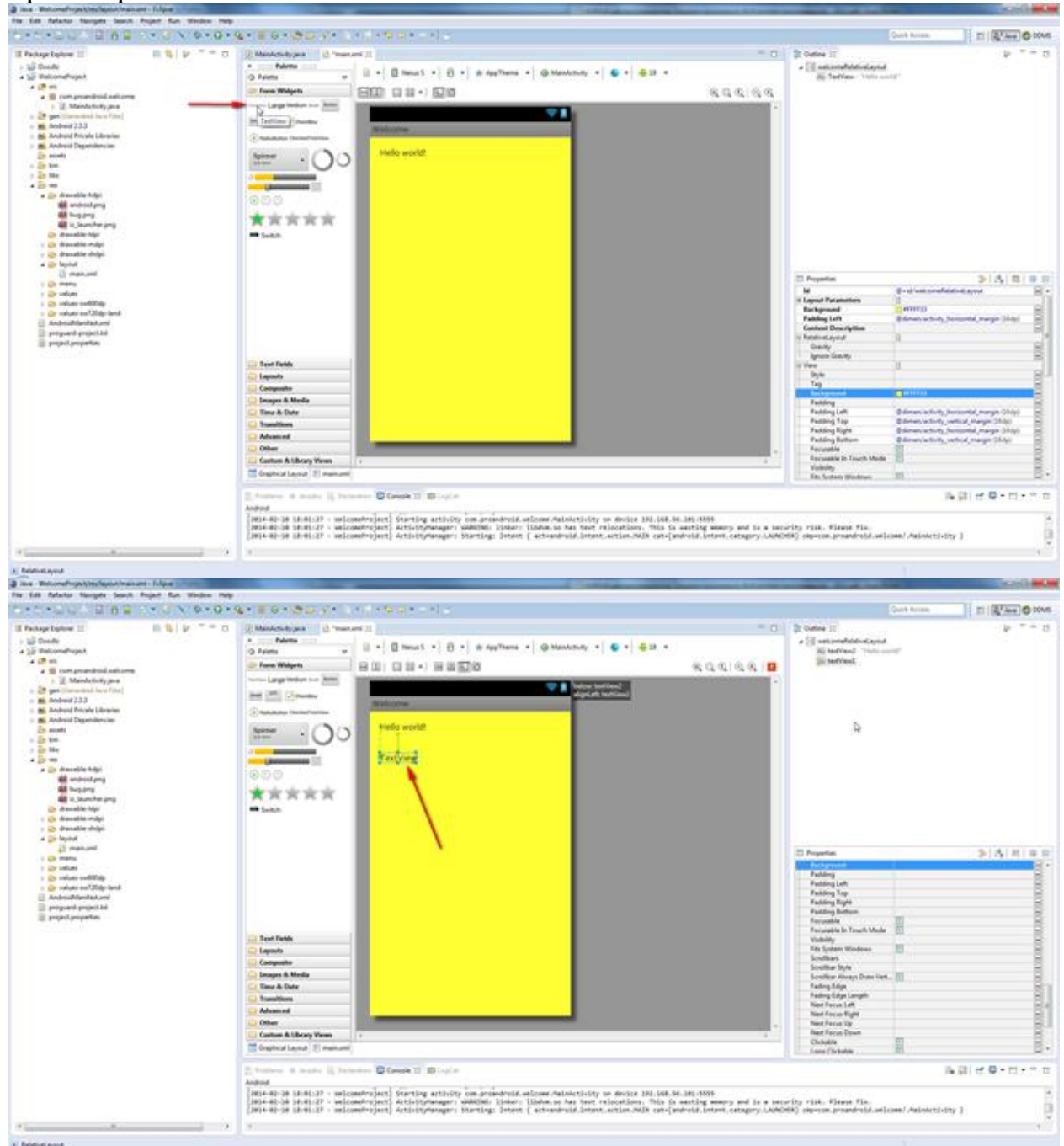
Шаг 3. Присваивание свойству Background у RelativeLayout значения По умолчанию в качестве фонового цвета макета выбирается белый (в нашем случае), но его можно изменить (например, на желтый). Любой цвет представляет собой результат смешения красного, синего и зеленого компонентов, которые называются RGB-компонентами. Каждый из этих компонентов может принимать целочисленное значение в диапазоне от 0 до 255. Первый компонент задает количество красного в общем цвете, второй — количество зеленого, а третий — количество синего цвета. Во время работы в интегрированной среде разработки (IDE) для определения цвета используется шестнадцатеричный формат. То есть RGB-компоненты представлены в виде значений в диапазоне 00–FF. Чтобы изменить фоновый цвет, найдите свойство Background в окне Properties и присвойте ему значение #FFFF33



Формат #RRGGBB— это пары шестнадцатеричных чисел, которые представляют красный, зеленый и синий цвета соответственно. В Android также поддерживаются альфа-значения (прозрачность), которые изменяются в диапазоне 0–255. Значение 0 соответствует полной прозрачности, а значение 255 — полной непрозрачности. Если планируется использовать альфа-значения, укажите цвет в формате #AARRGGBB, где первые две шестнадцатеричных цифры соответствуют альфа-значению. Если обе цифры каждого компонента цвета одинаковы, можно воспользоваться форма-том #RGBили #ARGB. Например, значение #FFFтракуется как #FFFFFF.

Шаг 4. Добавление компонента TextView

На этом шаге добавим в пользовательский интерфейс компонент TextView. В списке Form Widgets(Виджеты форм), находящемся в левой части окна Visual Layout Editor, найдите компонент TextViewи перетащите его в область проектирования



После добавления нового компонента в пользовательский интерфейс происходит его автоматический выбор и отображение свойств в окне Properties.

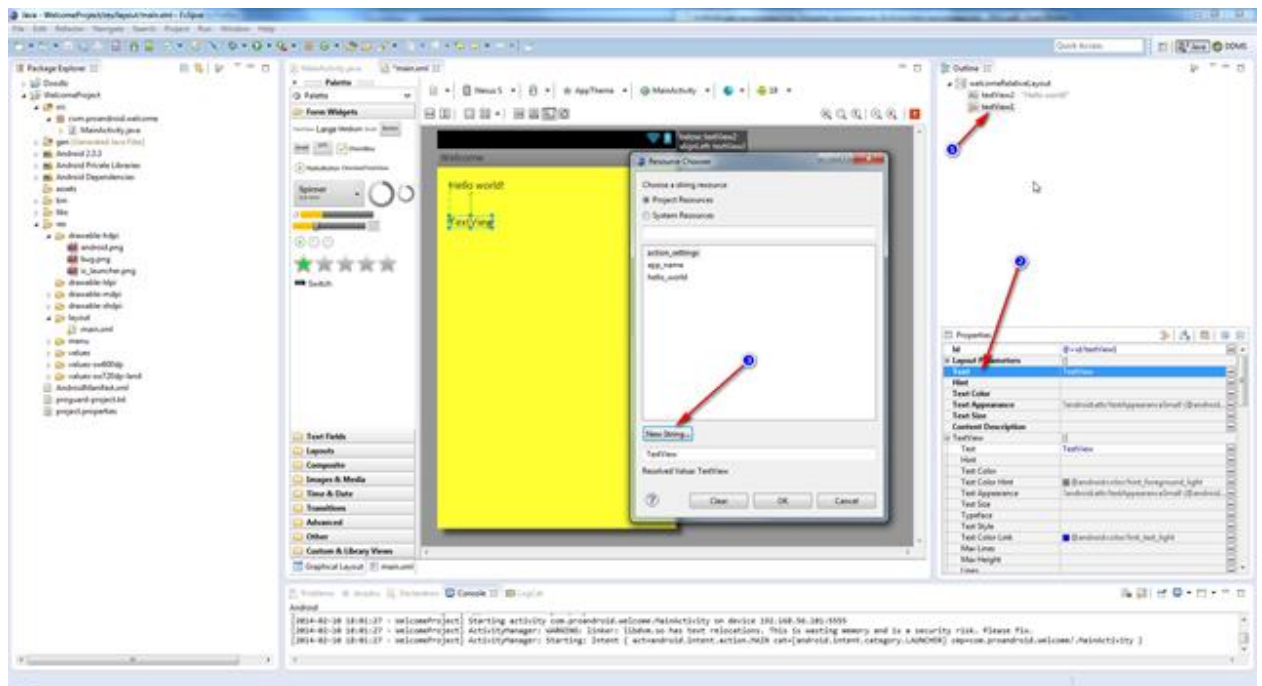
Шаг 5. Настройка свойства Text компонента TextView с помощью строковых ресурсов

Согласно документации Android по ресурсам приложений (developer.android.com/guide/topics/resources/index.html) считается хорошим тоном хранить строки, массивы строк, изображения, цвета, размеры шрифтов, размерности и другие ресурсы приложения так, чтобы их можно было использовать отдельно от кода приложения. Например, после экстернализации цветных значений все компоненты, использующие один и тот же цвет, могут быть перекрашены путем простого изменения значения цвета в центральном файле ресурса. Если нужно локализовать приложения, создав версии на нескольких разных языках, сохраните строки отдельно от кода приложения. В результате у вас в дальнейшем появится возможность простого изменения этих строк. В папке res проекта находится подпапка values, в которой помещен файл strings.xml. Этот файл применяется для хранения строк. Чтобы сформировать локализованные строки для других языков, создайте отдельные папки values для каждого используемого языка. Например, в папке values-fr может находиться файл strings.xml для французского языка, а в папке values-es — файл strings.xml для испанского языка. Можно также формировать названия этих папок с учетом информации о регионе. Например, в папке values-en-US может находиться файл strings.xml для американского диалекта английского языка, а в папке values-en-GB — файл strings.xml для диалекта английского языка, используемого в Великобритании. Чтобы получить дополнительные сведения о локализации, обратитесь к ресурсам:

[developer.android.com/guide/topics/resources/](http://developer.android.com/guide/topics/resources/providing-resources.html);
[providing-resources.html](http://developer.android.com/guide/topics/resources/providing-resources.html#AlternativeResources). AlternativeResources;
developer.android.com/guide/topics/resources/localization.html.

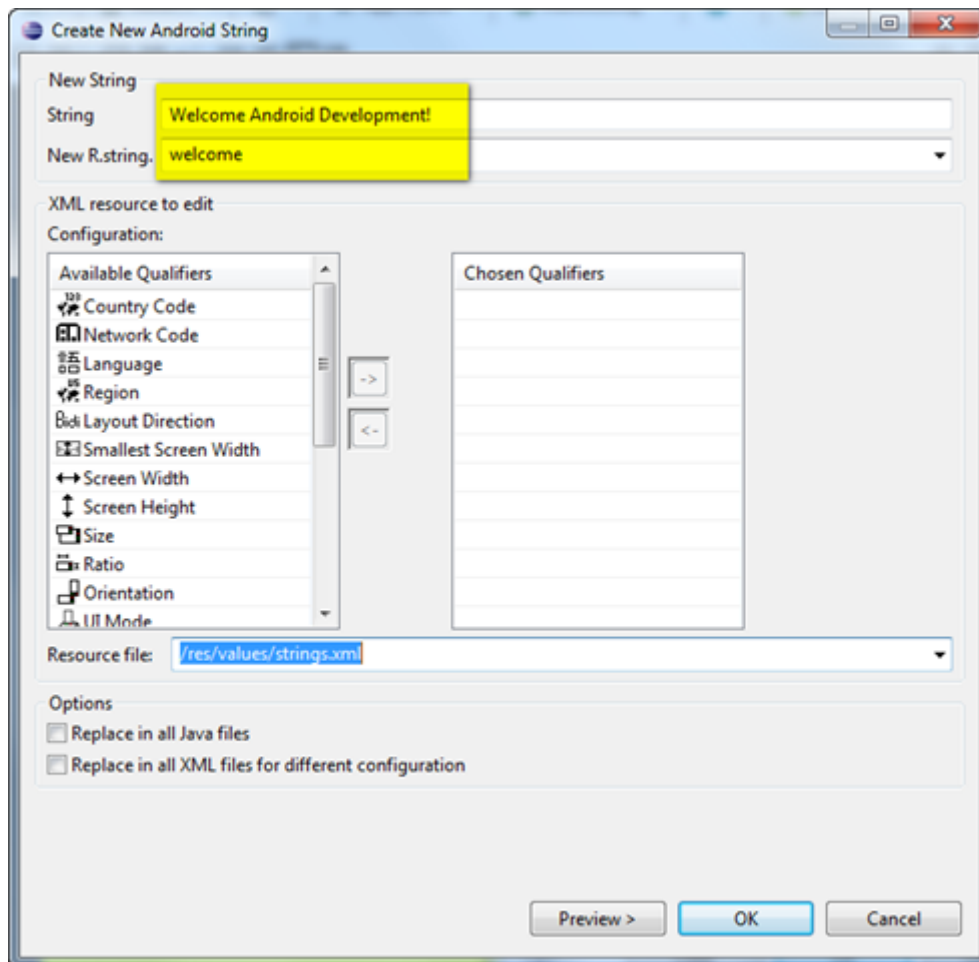
Чтобы установить значение свойства Text компонента TextView, создадим новый строковый ресурс в файле strings.xml.

1. Выберите компонент TextView.
2. В окне Properties найдите свойство Text, щелкните на значении, заданном по умолчанию, затем щелкните на кнопке с многоточием. Эта кнопка находится в правой части поля значения свойства и служит для отображения диалогового окна Resource Chooser (Выбор ресурсов).

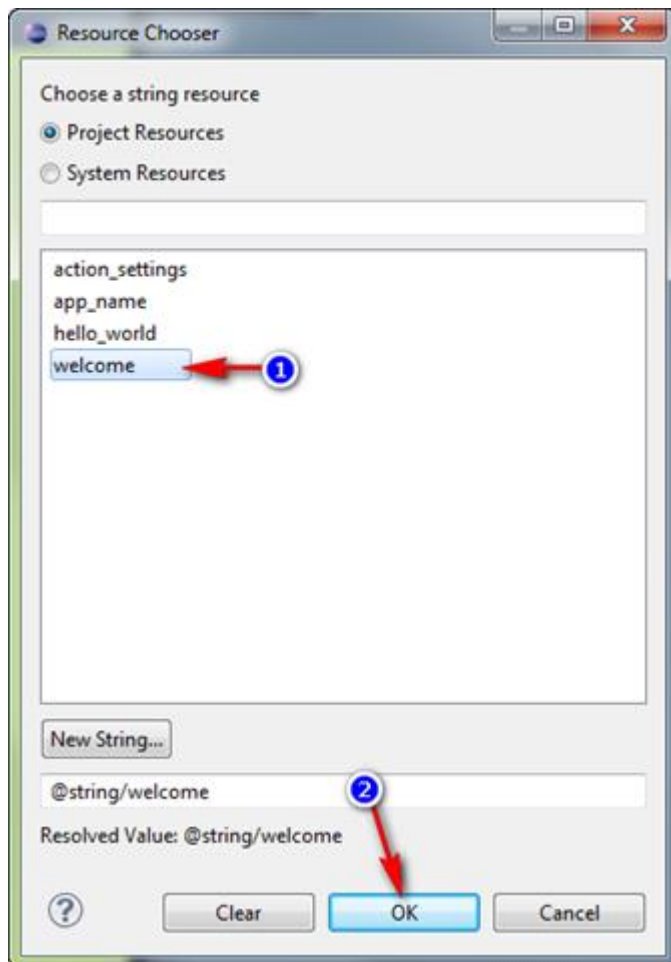


3. В диалоговом окне Resource Chooser щелкните на кнопке New String...(Новая строка...), чтобы отобразить диалоговое окно Create New Android String(Создать новую Android строку)

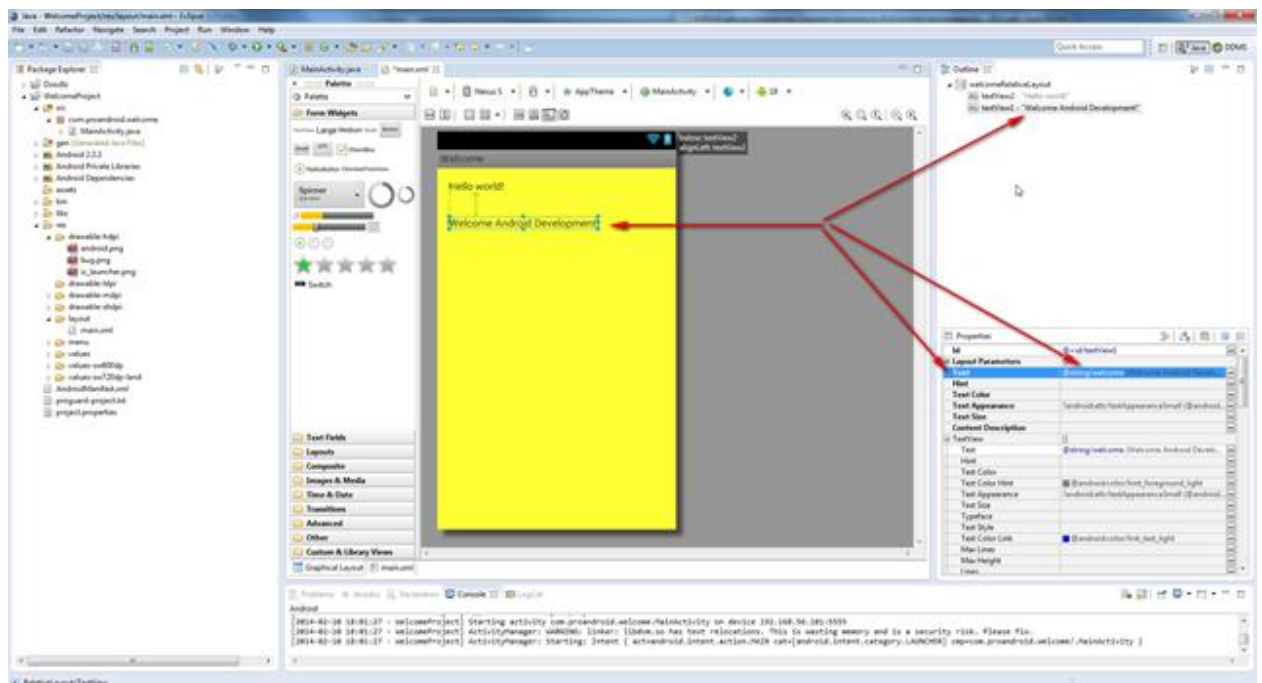
4. Заполните поля String и New R.string, потом щелкните на кнопке OK, чтобы скрыть диалоговое окно Create New Android String и вернуться к окну Resource Chooser.



5. Автоматически выбирается новый строковый ресурс welcome. Щелкните на кнопке ОК для выбора этого ресурса.



После выполнения указанных выше шагов в окне Properties появляется свойство Text. Запись в форме @string свидетельствует о том, что существующий строковый ресурс может быть выбран в файле strings.xml, а имя welcome говорит о том, что строковый ресурс выбран в данный момент времени.



Основное преимущество определения строковых значений заключается в облегчении локализации приложения, осуществляемой путем создания дополнительных

файлов XML-ресурсов для других языков. В каждом файле используется одно и то же имя в поле New R.stringи поддерживается интернационализированная строка в поле String. Потом Android может выбрать соответствующий файл ресурса, основываясь на языке, предпочтительном для пользователя устройства. Дополнительные сведения о локализации можно найти на веб-сайте developer.android.com/guide/topics/resources/localization.html.

Шаг 6. Настройка свойств Text size и Padding top компонента TextView — пиксели, независимые от плотности и от масштабирования

Размеры компонентов GUI и текста на экране Android могут определяться с помощью различных единиц измерения. Документация, описывающая различные размеры экранов, находится на веб-сайте по адресу developer.android.com/guide/practices/screens_support.html и рекомендует для определения размеров компонентов GUI и других экранных элементов использовать пиксели, независимые от плотности, а размеры шрифтов задавать с помощью пикселей, независимых от масштабирования.

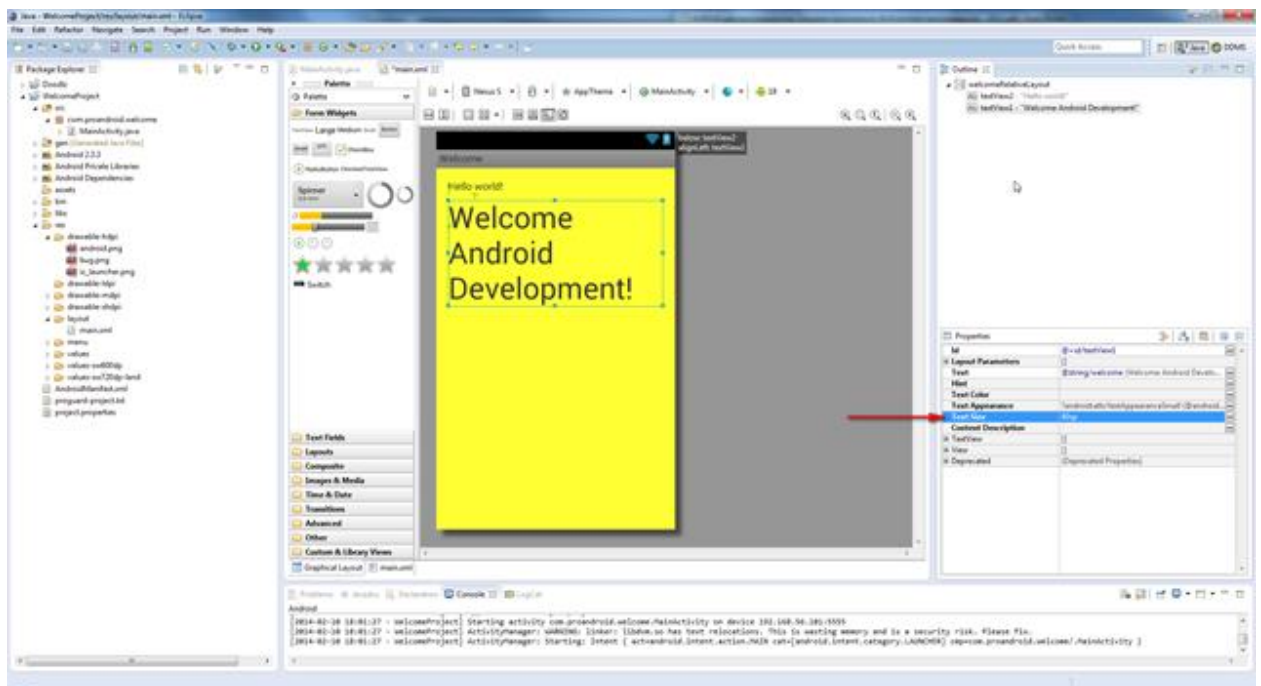
Единица измерения	Описание
px	Пиксель
dp или dip	Пиксель, независимый от плотности
sp	Пиксель, независимый от масштабирования
in	Дюймы
mm	Миллиметры

Задание размеров в пикселях, независимых от плотности (dp или dip), позволяет платформе Android автоматически масштабировать графический интерфейс пользователя в зависимости от плотности пикселей экрана физического устройства.

Размер пикселя, независимого от плотности, эквивалентен размеру физического пикселя на экране с разрешением 160 dpi (точек на дюйм). На экране с разрешением 240 dpi размер пикселя, независимого от плотности, будет масштабироваться на коэффициент 240/160 (то есть 1,5). Таким образом, компонент, размер которого составляет 100 пикселей, независимых от плотности, будет масштабирован до размера в 150 физических пикселей на таком экране. На экране с разрешением 120 точек на дюйм каждый независимый от плотности пиксель масштабируется на коэффициент 120/160 (то есть 0,75). Значит, 100 независимых от плотностей пикселей превратятся на таком экране в 75 физических пикселей. Пиксели, независимые от масштаба, масштабируются

так же, как и пиксели, независимые от плотности, но их масштаб зависит также и от предпочитаемого размера шрифта, выбираемого пользователем. А теперь увеличим размер шрифта для компонента TextView и добавим небольшой отступ над компонентом TextView.

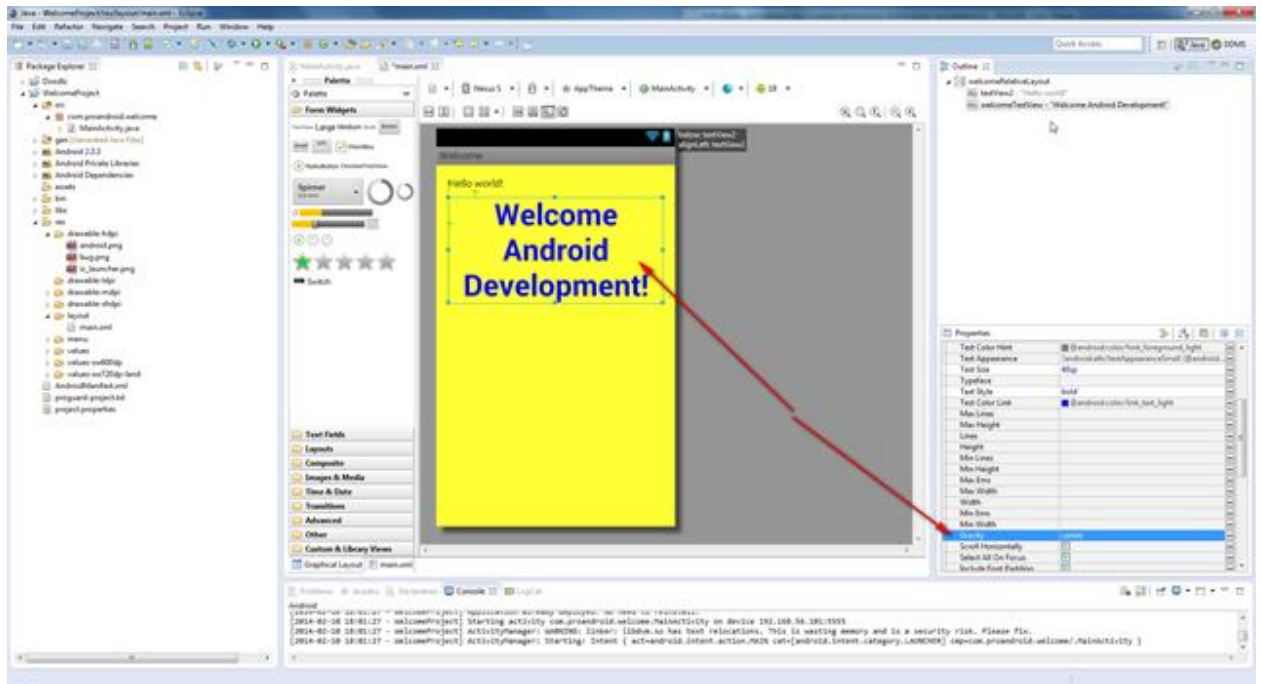
1. Чтобы изменить размер шрифта, выберите компонент TextView, а потом присвойте свойству Text size значение 40sp.
2. Чтобы увеличить отступ между верхним краем макета и компонентом TextView, выберите свойство Layout margin top property в разделе Misc(Разное) окна Properties и присвойте ему значение 10dp.



Шаг 7. Настройка дополнительных свойств компонента TextView

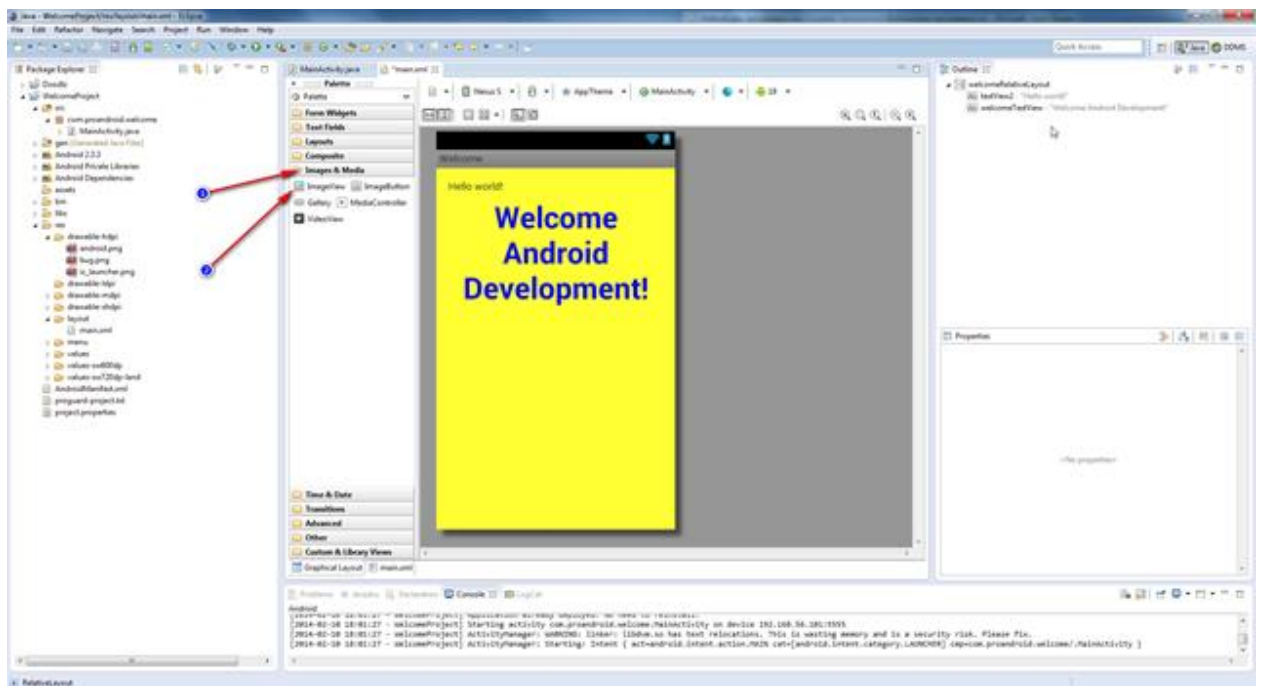
Сконфигурируйте дополнительные свойства компонента TextView следующим образом:

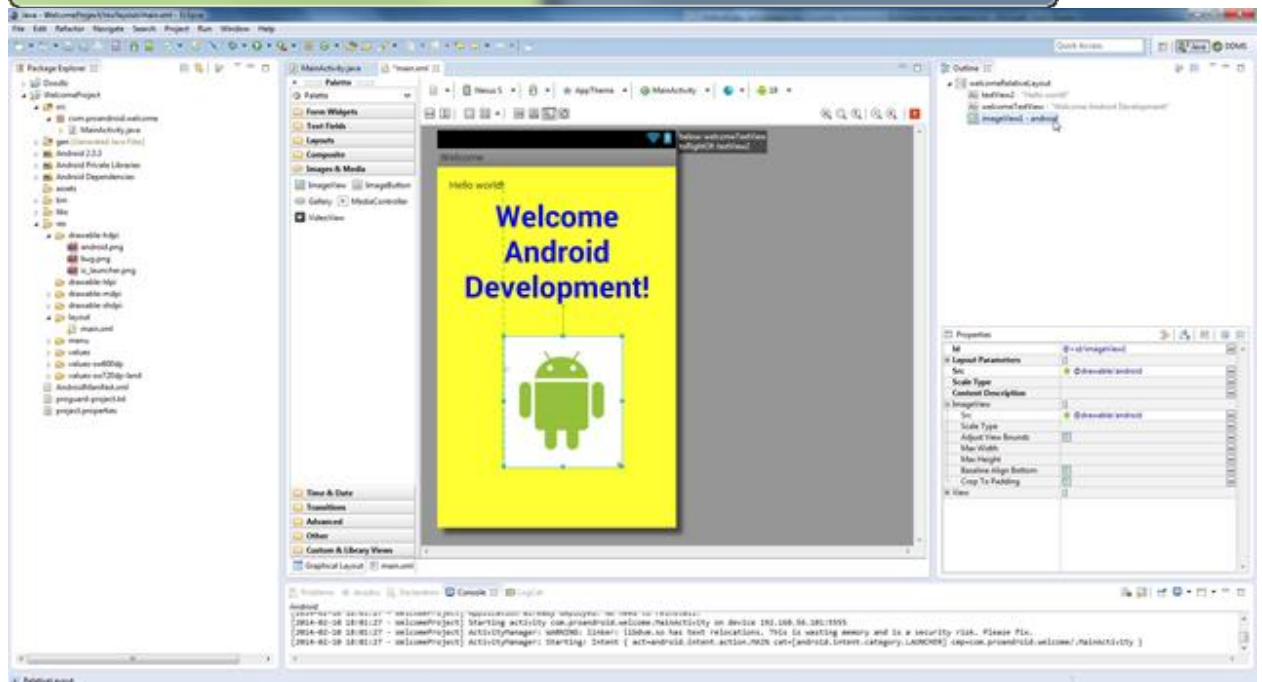
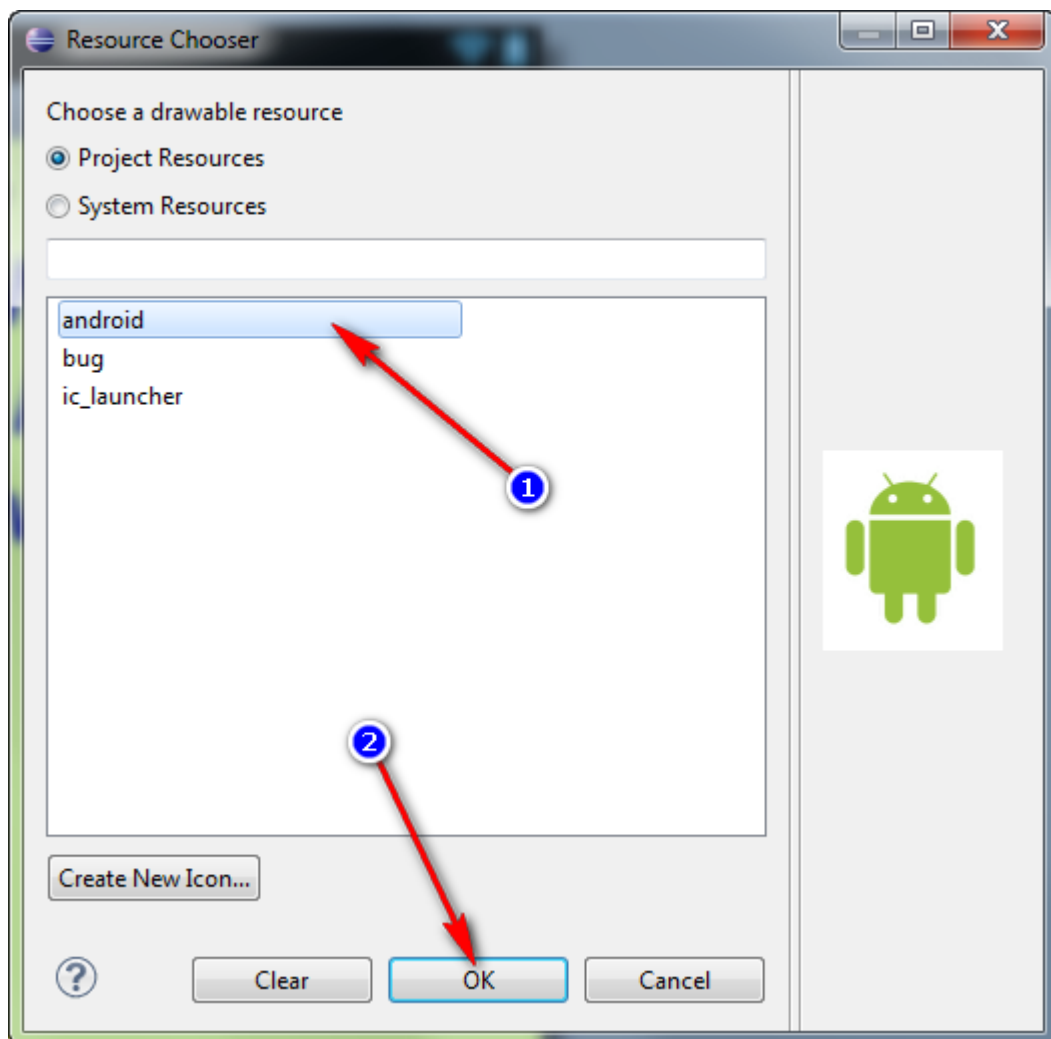
1. Свойству Id присвойте значение `@+id/welcomeTextView`.
2. Свойству Text color присвойте значение `#00F`(синий).
3. Свойству Text style присвойте значение `bold`. Чтобы выполнить эту операцию, щелкните в области поля Value этого свойства, затем щелкните на кнопке с многоточием для отображения диалогового окна, в котором выбирается стиль шрифта. Установите флажок `bold`, потом щелкните на кнопке `OK` для выбора стиля текста.
4. Чтобы отцентрировать текст компонента TextView при его расположении в нескольких строках свойству Gravity присвойте значение `center`. Для выполнения этой операции щелкните в области поля Value этого свойства, потом щелкните на кнопке с многоточием для отображения диалогового окна параметров свойства Gravity



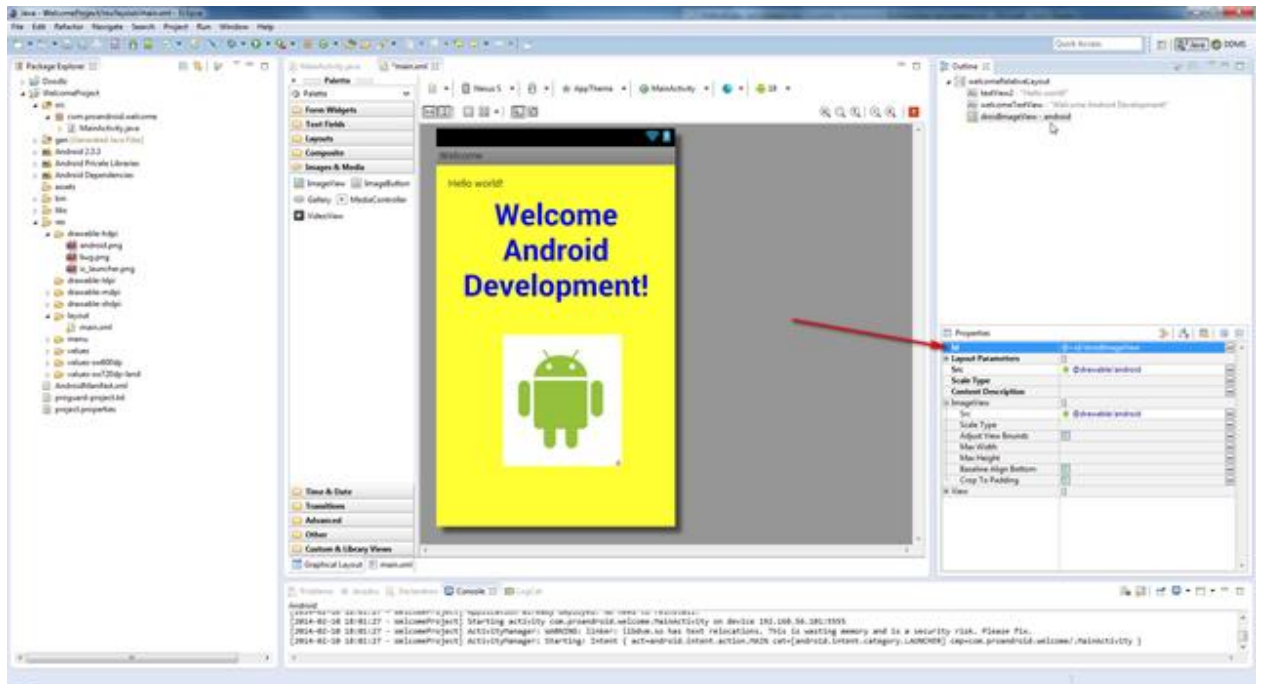
Шаг 8. Отображение логотипов Android и Deitel Bug с помощью компонентов ImageViews

1. Перетащите компонент ImageView, находящийся в категории Images & Media (Изображения и медиаресурсы) палитры Visual Layout Editor, в окно Outline(Макет), . Новый компонент ImageView появится ниже узла welcomeTextView.

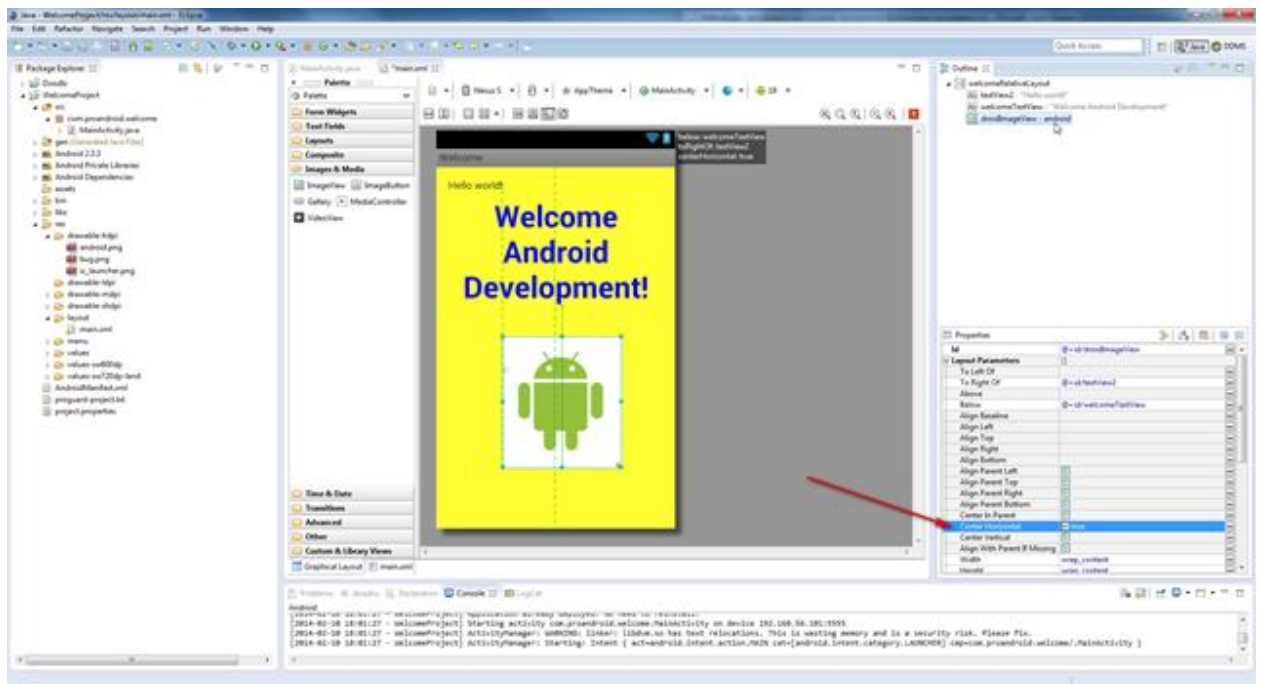




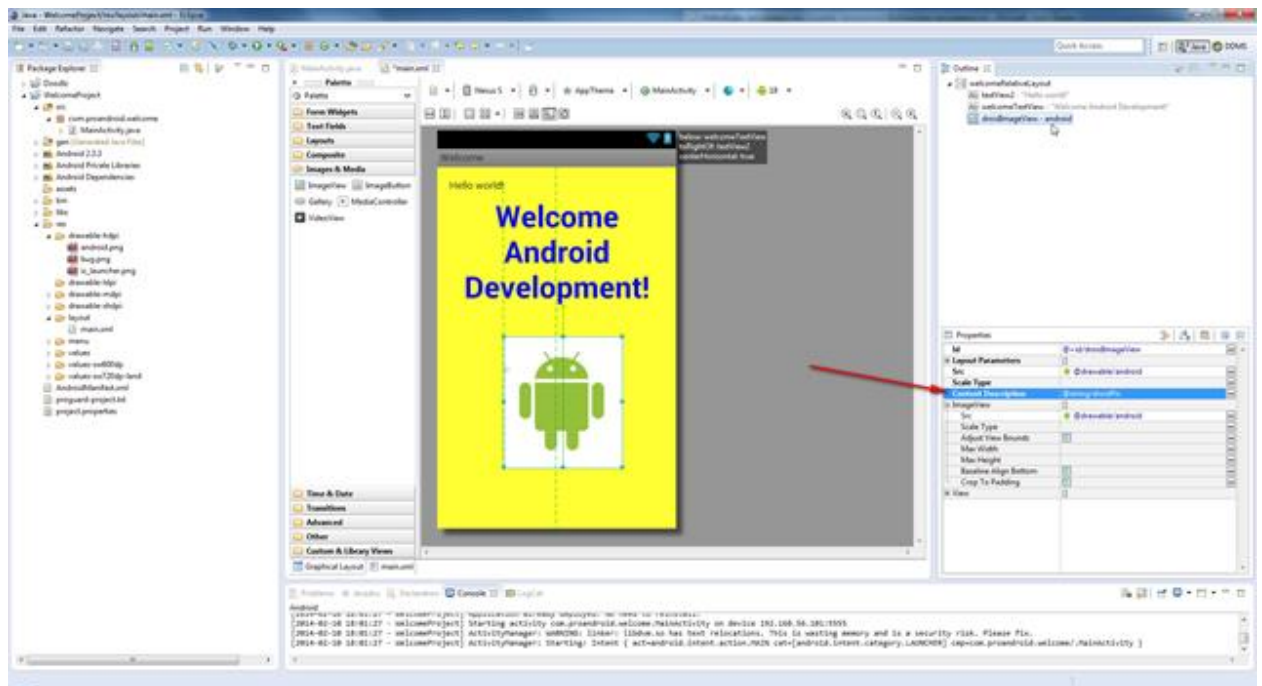
2. Свойству Id компонента ImageView присвойте значение @+id/droidImageView. В окне Outline отобразится название объекта droidImageView.



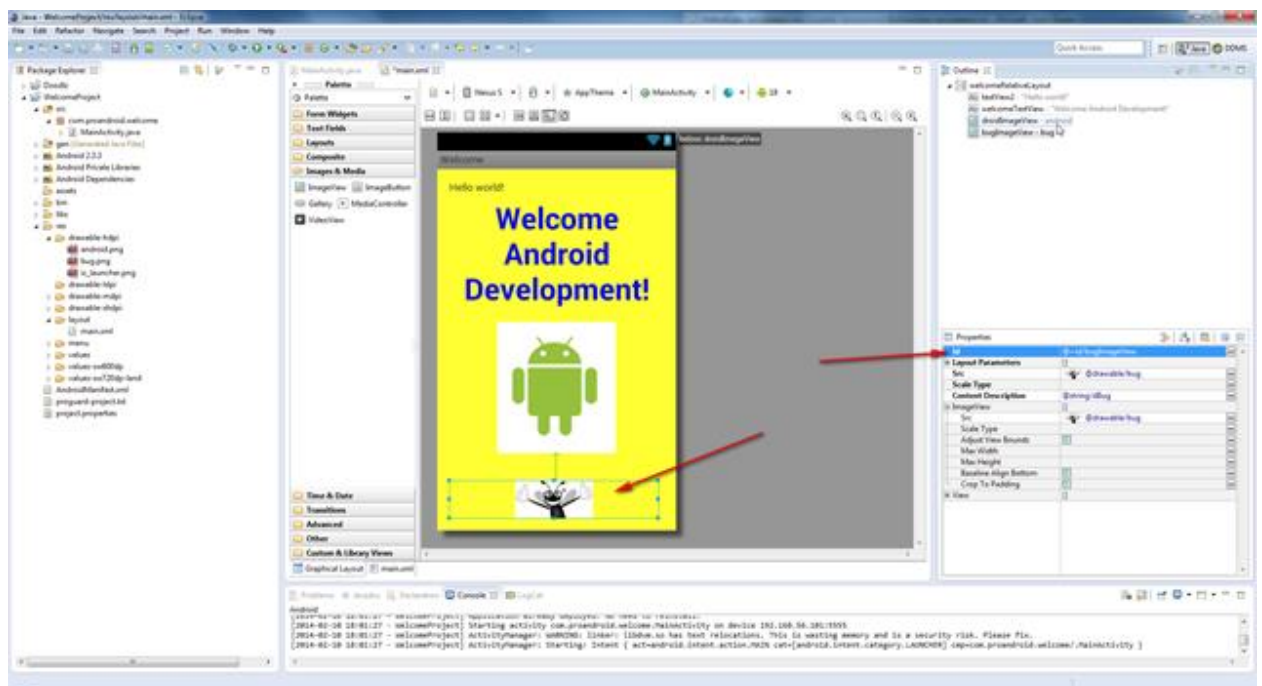
3. Присвойте свойству `Layout center horizontal` объекта `droidImageView` значение `true`, чтобы отцентрировать компонент `ImageView` внутри макета.



4. Полю `Content Description` присвойте значение `@string/droidPic`



5. Подобные действия делаем для рисунка bug.png. При-своей свойству Id компонента значение @+id/bugImageView.

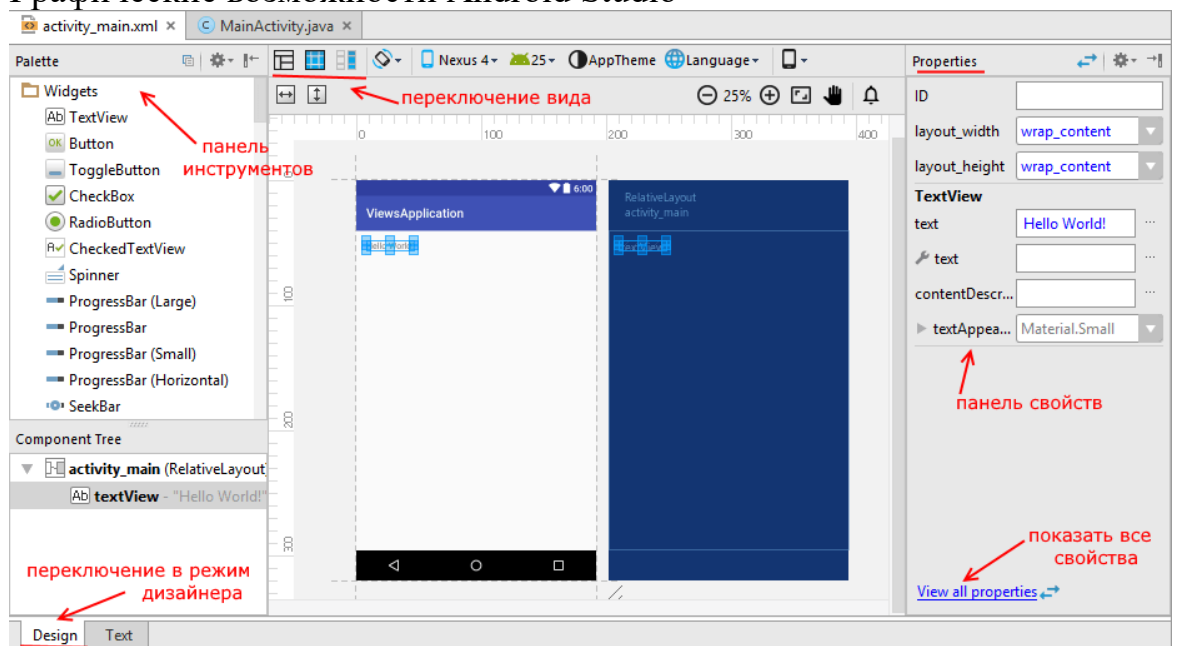


Ну и запустим наше первое приложение :)



ЛЕКЦИЯ №27. ГРАФИЧЕСКИЙ ИНТЕРФЕЙС ПОЛЬЗОВАТЕЛЯ В ANDROID-ПРИЛОЖЕНИЯХ.

Графические возможности Android Studio



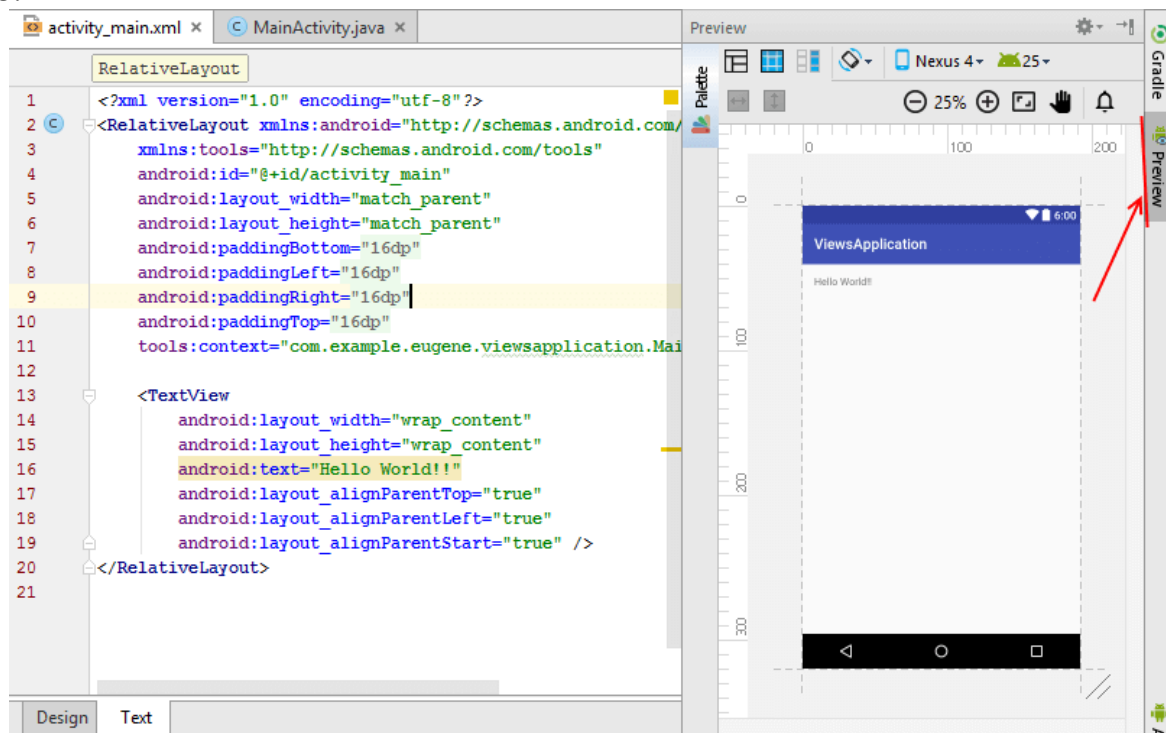
Мы можем открыть файл `activity_main.xml` и внизу с помощью кнопки Design переключиться в режим дизайнера к графическому представлению интерфейсу в виде эскиза смартфона.

Слева будет находиться панель инструментов, с которой мы можем переносить нужный элемент мышкой на эскиз смартфона. И все перенесенные элементы будут автоматически добавляться в файл `activity_main.xml`. С помощью мыши мы можем изменять позиционирование уже добавленных элементов, перенося их в другое место на смартфоне.

Справа будет окно Properties - панель свойств выделенного элемента. Здесь мы можем изменить значения свойств элемента. И опять же после изменения свойств изменится и содержимое файла `activity_main.xml`.

То есть при любых изменениях в режиме дизайнера будет происходить синхронизация с файлом `activity_main.xml`. Все равно, что мы вручную изменяли бы код непосредственно в файле `activity_main.xml`.

Но даже если мы предпочитаем работать с разметкой интерфейса в текстовом виде, то даже здесь мы можем включить предварительный просмотр для файла `activity_main.xml`. Для этого после переключения в текстовый режим необходимо нажать на вкладку Preview справа в Android Studio:



Это очень удобно, так как сразу позволяет посмотреть, как будет выглядеть приложение. А при любых изменениях область предварительного просмотра будет автоматически синхронизироваться с содержанием файла `activity_main.xml`.

В приложениях Android визуальный интерфейс нередко загружается из специальных файлов xml, которые хранят разметку. Эти файлы являются ресурсами разметки. Подобный подход напоминает создание веб-сайтов,

когда интерфейс определяется в файлах html, а логика приложения - в коде javascript.

Объявление пользовательского интерфейса в файлах XML позволяет отделить интерфейс приложения от кода. Что означает, что мы можем изменять определение интерфейса без изменения кода java. Например, в приложении могут быть определены разметки в файлах XML для различных ориентаций монитора, различных размеров устройств, различных языков и т.д. Кроме того, объявление разметки в XML позволяет легче визуализировать структуру интерфейса и облегчает отладку.

Файлы разметки графического интерфейса располагаются в проекте в каталоге res/layout. По умолчанию при создании проекта уже есть один файл ресурсов разметки activity_main.xml, который может выглядеть примерно так:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context="com.example.eugene.viewsapplication.MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</android.support.constraint.ConstraintLayout>
```

В файле определяются все графические элементы и их атрибуты, которые составляют интерфейс. При создании разметки в XML следует соблюдать некоторые правила: каждый файл разметки должен содержать один корневой элемент, который должен представлять объект View или ViewGroup.

В данном случае корневым элементом является элемент ConstraintLayout, который содержит элемент TextView.

При компиляции каждый XML-файл разметки компилируется в ресурс View. Загрузка ресурса разметки осуществляется в методе Activity.onCreate. Чтобы установить разметку для текущего объекта activity, надо в метод setContentView в качестве параметра передать ссылку на ресурс разметки.

Для получения ссылки на ресурс в коде java необходимо использовать выражение R.layout.[название_ресурса]. Название ресурса layout будет совпадать с именем файла, поэтому чтобы использовать файл activity_main.xml в качестве

источника визуального интерфейса, нам надо изменить код MainActivity следующим образом:

```
package com.example.eugene.viewsapplication;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

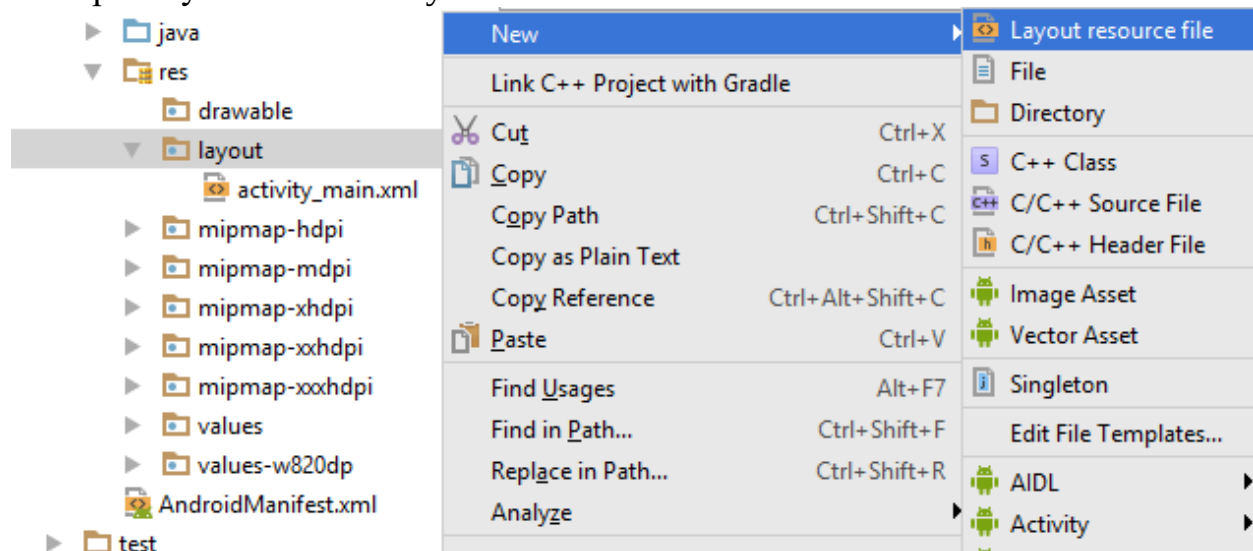
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

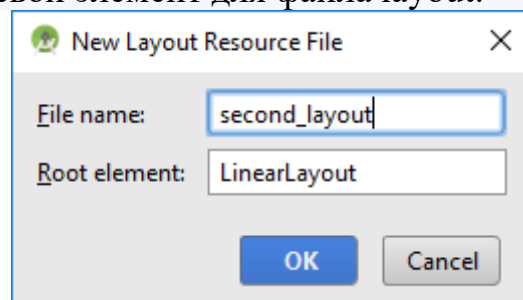
Добавление файла layout

Но у нас может быть и несколько различных ресурсов layout. Как правило, каждый отдельный класс Activity использует свой файл layout. Либо для одного класса Activity может использоваться сразу несколько различных файлов layout.

К примеру, добавим в проект новый файл разметки интерфейса. Для этого нажмем на папку res/layout правой кнопкой мыши и в появившемся меню выберем пункт New -> Layout resource file:



После этого в специальном окошке будет предложено указать имя и корневой элемент для файла layout:



В качестве названия укажем `second_layout`, а в качестве типа корневого элемента оставим `LinearLayout`.

После этого в папку `res/layout` будет добавлен новый файл `second_layout.xml`, с которым мы можем работать точно также, как и с `activity_main.xml`. В частности, откроем файл `second_layout.xml` и изменим его содержимое следующим образом:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView android:id="@+id/header"
        android:text="Second Activity"
        android:textSize="26dp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</LinearLayout>
```

Здесь определено текстовое поле `TextView`, которое выводит на экран простейший текст "Second Activity".

Получение элементов в коде и их идентификаторы

Кроме текста, ширины и высоты текстовое поле устанавливает еще один важный атрибут - `id`. Знак `+` в записи `android:id="@+id/header"` означает, что если для элемента не определен `id` со значением `header`, то его следует определить.

Чтобы использовать этот файл в качестве основного интерфейса, перейдем к `MainActivity` и изменим ее код:

```
package com.example.eugene.viewsapplication;
```

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.TextView;
```

```
public class MainActivity extends AppCompatActivity {
```

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        // устанавливаем в качестве интерфейса файл second_layout.xml
        setContentView(R.layout.second_layout);
```

```
        // получаем элемент textView
```

```
        TextView textView = (TextView) findViewById(R.id.header);
```

```
        // переустанавливаем у него текст
```

```
        textView.setText("Hello Android 7!");
```

```
    }
```

```
}
```

С помощью метода `setContentView()` устанавливается разметка из файла `second_layout.xml`.

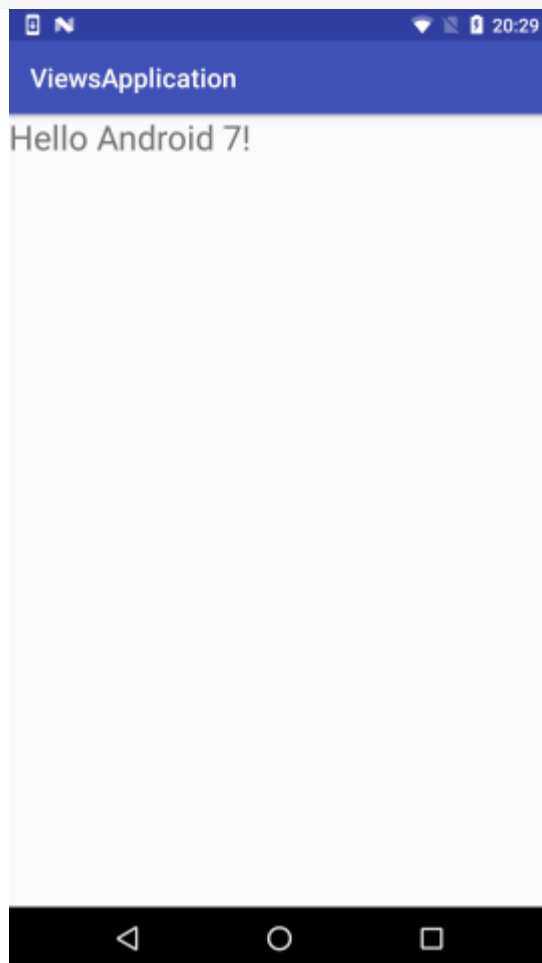
Другой важный момент, который стоит отметить - получение визуального элемента `TextView`. Так как в его коде мы определили атрибут `android:id`, то через этот `id` мы можем его получить.

Для получения элементов по `id` класс `Activity` имеет метод `findViewById()`. В этот метод передается идентификатор ресурса в виде `R.id.[идентификатор_элемента]`. Этот метод возвращает объект `View` - объект базового класса для всех элементов, поэтому результат метода еще необходимо привести к типу `TextView`.

Далее мы можем что-то сделать с этим элементом, в данном случае изменяем его текст.

Причем что важно, получение элемента происходит после того, как в методе `setContentView` была установлена разметка, в которой этот визуальный элемент был определен.

И если мы запустим проект, то увидим, что `TextView` выводит новый текст:



ЛЕКЦИЯ №28-№29. АРХИТЕКТУРА ПЛАТФОРМЫ ANDROID.

Платформа Android представляет собой программный стек операционной системы на основе Linux, предназначенный для управления

устройством (аппаратурой), памятью и процессами. Различные компоненты этого стека представляют собой несколько уровней иерархии и, в целом, обеспечивают функционирование мобильного устройства. Одни из этих компонентов необходимы для работы непосредственно с оборудованием устройства, другие обеспечивают функции связанные с телефонией, позиционированием, мультимедиа и так далее, а третьи предоставляют приложениям фреймворк для реализации многочисленных возможностей мобильного устройства. Другими словами, платформа Android включает в себя непосредственно операционную систему, программное обеспечение промежуточного уровня (middleware) и набор приложений (как встроенных, так и сторонних)[1].

Как мы уже выяснили, Android - это программный стек, то есть целый набор различных программных компонентов (о которых мы будем неоднократно говорить на протяжении всей статьи). С другой стороны, устройство на платформе Android представляет собой набор аппаратных средств, таких как экран, клавиши, батарея, процессор, память всевозможные датчики и сенсоры и так далее. Все это позволяет нам говорить о платформе Android как о едином аппаратно-программном комплексе, который имеет следующую структуру:

- Аппаратура - набор элементов, обеспечивающих функционирование устройства.
- Linux Kernel (+Drivers) - ядро операционной системы Linux и набор драйверов, которые обеспечивают базовые механизмы управления оборудованием и распределения памяти, управления задачами, обеспечения безопасности и так далее.
- Библиотеки (C/C++), используемые различными компонентами операционной системы.
- Dalvik Virtual Machine (DVM) - виртуальная машина Dalvik, которая обеспечивает среду выполнения Android приложений и компонентов операционной системы (ОС).
- Core Library - основные Android библиотеки, содержат большинство функций доступных в ядре библиотеки языка Java (Java API), а так же специальные функции операционной системы Android.
- Application Framework (каркас приложений) - набор Java классов (API), предоставляющий приложениям интерфейс к функциям операционной системы и библиотекам.
- Встроенные и пользовательские приложения.

Архитектура платформы Android

Архитектуру Android можно разделить на шесть уровней: уровень оборудования; уровень ядра Linux; уровень нативных библиотек; уровень среды выполнения Android; уровень каркаса приложений (Application Framework); уровень приложений. На рисунке 2.1 показана архитектура платформы Android.

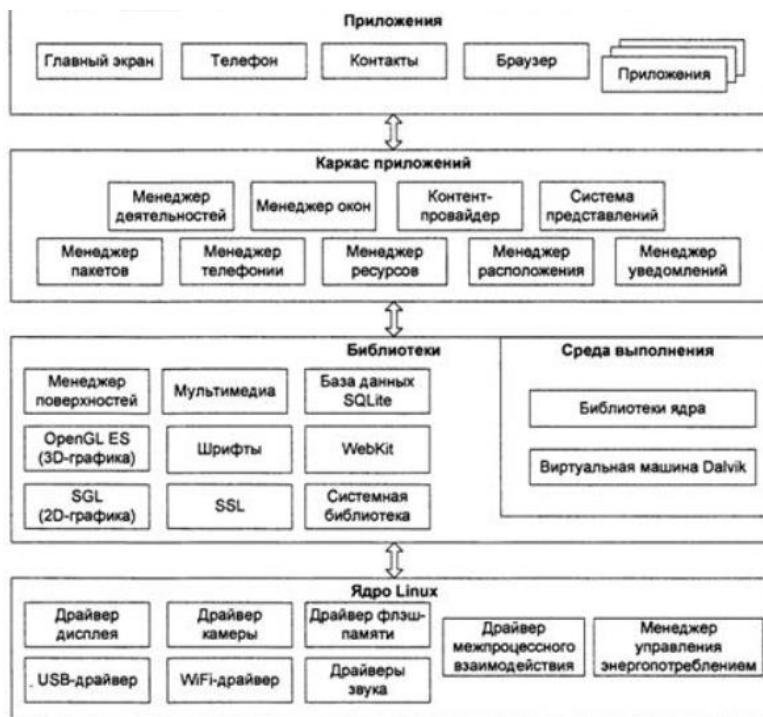


Рисунок 2.1 - Архитектура платформы Android.

Ядро Linux и набор драйверов. Центром платформы Android является ядро операционной системы Linux (версии 2.6.25 - 3.0.31, в зависимости от версии Android), которое отвечает за решение основных задач операционной системы и взаимодействие с оборудованием.

На этом уровне располагаются основные службы для управления процессами и распределения памяти, управления файловой системой и безопасностью, а так же службы для управления сетевыми функциями (сетевой стек). Только на уровне ядра происходит непосредственное взаимодействие с оборудованием устройства посредством набора драйверов. Основными компонентами уровня ядра являются:

- драйвер межпроцессного взаимодействия (IPC Driver).
- драйвер управления энергопотреблением (Android Power Management).
- набор драйверов для оборудования, входящего в состав мобильного устройства[2].

Хотя операционная система Android основано на ядре Linux, между ними (Android и Linux в чистом виде) существуют некоторые различия. Так Android, например, содержит свои механизмы распределения памяти, взаимодействие между процессами и др.

Драйвер IPC. Приложения и службы в Android работают в отдельных (защищенных) процессах, которые могут взаимодействовать между собой и имеют доступ к общей памяти. Платформа Android обеспечивает механизм IPC (Inter-process Communication), который является основным механизмом взаимодействия между процессами в системе. Драйвер IPC обеспечивает функции взаимодействия процессов, создания и обработки пулов потоков в процессах, подсчета и отображения ссылок на объекты в процессах, синхронных запросов между процессами.

Управление энергопотреблением. Система управления энергопотреблением (Android Power Management) представляет собой оптимизированный для мобильных устройств (с учетом их специфических особенностей), стандартный драйвер управления питанием Linux. Основная функция этой системы заключается в экономном использовании заряда батареи мобильного устройства за счет поддержки режимов пониженного энергопотребления.

Драйверы оборудования. Программный стек Android был разработан с учетом колоссальной гибкости, необходимой при выборе аппаратной конфигурации устройств. Другими словами концепция архитектуры программного стека Android предусматривает возможность использования многих различных аппаратных компонентов и средств, которые могут предоставлять дополнительную функциональность мобильным устройствам[3].

Как было сказано выше, непосредственное взаимодействие с аппаратурой устройства происходит только на уровне ядра операционной системы Linux, а значит, для остальных уровней программного стека Android все различия аппаратуры должны быть прозрачными. Именно по этой причине уровень ядра включает в себя все необходимые драйверы оборудования, присутствующего на устройстве. Набор драйверов может быть различным в зависимости от производителя и модели устройства (то есть в зависимости от аппаратной конфигурации).

Нативные библиотеки. Следующий уровень над ядром Linux включает набор библиотек (написанных на C/C++), которые используются различными компонентами ОС. Библиотеки этого уровня по своему функциональному назначению можно разделить на следующие группы:

- Системная библиотека Bionic (libc), написанная на языке C (библиотека времени выполнения).
- Менеджер поверхностей.
- Функциональные библиотеки C/C++.

Системная библиотека Bionic (libc). Системная библиотека базируется на лицензии Berkeley Software Distribution (BSD) и разработана для мобильных устройств на основе Linux. Компания Google разработала свою собственную версию этой библиотеки (libc Bionic) по следующим причинам:

- Экономия памяти. Так как эта библиотека загружается в каждый процесс, её размер должен быть как можно меньше. Библиотека Bionic имеет размер всего около 200 Кбайт.
- Максимальное быстродействие. Библиотека Bionic оптимизирована для работы на мобильном устройстве.

Библиотека Bionic имеет встроенную поддержку важных для Android системных служб в частности поддержку службы регистрации системных событий. Библиотека Bionic не совместима с GNU libc и стандартом POSIX.

Менеджер поверхностей. Система Android использует композитный менеджер поверхностей, похожий на Compoz (композитный менеджер окон для X Window System, использующий для ускорения 3D графики OpenGL).

Менеджер поверхностей обрабатывает весь рендеринг (отрисовку) поверхности на фреймовый буфер[4]. Менеджер может объединить 2D- и 3D-поверхности и поверхности от нескольких приложений. Вместо того чтобы производить отрисовку графики напрямую в буфер дисплея, система посылает поступающие команды отрисовки в закадровый буфер, где они накапливаются вместе с другими, составляя некую композицию, а потом выводятся на экран.

Функциональные библиотеки. Библиотеки, используемые различными компонентами операционной системы Android. Некоторые из них перечислены ниже:

- Media Framework - обеспечивает поддержку функций мультимедиа (воспроизведение аудио и видео) и поддержку различных медиа форматов (MPEG4, H.264, MP3, WMA, WAV, AAC, AMR, JPG, PNG)
- SQLite -- библиотека поддержки реляционной базы данных SQLite.
- OpenGL ES - движок для работы с 3D-графикой
- FreeType - библиотека шрифтов, предназначенная для работы с растровыми и векторными шрифтами.
- LibWebCore - обеспечивает работу web-браузера (движок WebKit).
- SGL - поддержка 2D графики. Android также содержит собственную графическую 2D-библиотеку Skia, которая написана на C/C++ (Skia также формирует ядро популярного браузера Google Chrome).
- SSL (Secure Sockets Layer) - библиотека предназначена для работы с сокетами в безопасном (защищенном) режиме.

Для разработчиков доступ к функциям библиотек этого уровня реализован через использование Application Framework (уровень каркаса приложений), который будет подробно рассмотрен далее.

Среда выполнения Android (Android Runtime). Android Runtime Environment состоит из набора базовых (основных) библиотек Android и виртуальной машины Dalvik. Именно этот уровень обеспечивает среду выполнения для приложений Android.

Базовые библиотеки обеспечивают большую часть низкоуровневой функциональности, доступной библиотекам ядра языка Java (Java API), а так же специальные функции операционной системы Android. Эти библиотеки являются основной платформой разработки Java-приложений для Android.

Dalvik Virtual Machine (DVM). Основанная на регистрах виртуальная Java машина, разработанная и написанная Дэном Борнштейном (англ. Dan Bornstein) и другими, как часть мобильной платформы Android.

Виртуальная машина Dalvik оптимизирована для низкого потребления памяти, это нестандартная регистр ориентированная виртуальная машина, хорошо подходящая для исполнения на RISC-архитектурах процессоров, часто используемых в мобильных и встраиваемых устройствах, таких, как коммуникаторы и планшетные компьютеры. Большинство виртуальных машин, используемых в настольных системах, являются стек ориентированными, включая стандартную виртуальную машину Java от Oracle.

Программы для Dalvik пишутся на языке Java. Несмотря на это, стандартный байт-код Java не используется, вместо него Dalvik VM исполняет байт-код собственного формата. После компиляции исходных текстов программы на Java (при помощи `javac`) утилита `dx` из «Android SDK» преобразует `.class` файлы в формат `.dex`, пригодный для интерпретации в Dalvik.

В общем можно сказать, что Dalvik это среда для выполнения компонентов операционной системы (уровня приложений) Android и пользовательских приложений. Каждый процесс выполняется в своём, изолированном адресном пространстве. Когда пользователь запускает приложение (либо операционная система запускает один из своих компонентов), Ядро виртуальной машины Dalvik (Zygote Dalvik VM) создает отдельный, защищенный процесс в общей памяти, в котором непосредственно разворачивается VM, как среда для запуска приложения. Другими словами, изнутри Android выглядит как набор виртуальных машин Dalvik, в каждой из которых выполняется прикладная задача (приложение).

Уровень каркаса приложений (Application Framework). Уровень каркаса приложений является связующим звеном между уровнем нативных библиотек и среды выполнения Android и уровнем приложений. Этот уровень предоставляет приложениям интерфейсы (набор Java классов, API) к функциям операционной системы. На этом уровне находятся основные службы Android для управления жизненным циклом приложений, пакетами, ресурсами и т. д. Пользовательские приложения имеют полный доступ к тем же API, что и системные приложения.

Библиотеки и службы этого уровня являются основой для всех приложений (как системных, так и пользовательских) в операционной системе Android и включают в себя:

- Богатый и расширяемый набор элементов пользовательского интерфейса (представлений, Views). Это текстовые поля, кнопки, списки, таблицы, встраиваемый web браузер и т.д., элементы, которые используются практически во всех приложениях.

- Activity Manager - управляет жизненным циклом приложений и стеком Activity (история запусков Activities). Обеспечивает систему навигации между Activity.

- Package Manager - менеджер пакетов управляет установкой и развертыванием пакетов прикладных программ на устройстве.

- Resource Manager - содержит функции для доступа к ресурсам приложения (строкам, значениям, графике, макетам пользовательского интерфейса и др.).

- Content Providers - провайдеры контента, позволяют приложениям получать доступ к данным других приложений или предоставлять доступ к своим данным.

- Telephony Manager - менеджер телефонии, предоставляет доступ (API) к основным функциям телефонии (получение информации о сотовой сети, звонки и т.д.).

- Location Manager - менеджер местоположения, предоставляет приложениям функции для работы с информацией о текущем местоположении и его изменении.

- Notification Manager - менеджер уведомлений, предоставляет API для работы с системой уведомлений Android.

Компоненты уровня каркаса приложений построены таким образом, что бы облегчить их использование и сделать доступ к функциям операционной системы и нативным библиотекам при разработке приложений как можно более простым.

Уровень приложений. Мобильное устройство на платформе Android поставляется с набором основных или, другими словами системных приложений (таких как телефон, почта, сообщения, контакты, браузер, карты и т.д.). Так же существует возможность добавлять (устанавливать) сторонние (или пользовательские) приложения в систему. Все эти приложения написаны на языке Java. Важным фактом является то, что в Android нет разницы между системными и сторонними приложениями. Это означает в частности, что любое системное приложение можно заменить альтернативным (сторонним) приложением.

Любое приложение имеет полный доступ к функционалу, предоставляемому операционной системой. Архитектура всех приложений построена с учетом возможности легкого использования частей или функций других приложений.

ЛЕКЦИЯ №30. XML-РАЗМЕТКА ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ.

В приложениях Android визуальный интерфейс нередко загружается из специальных файлов xml, которые хранят разметку. Эти файлы являются ресурсами разметки. Подобный подход напоминает создание веб-сайтов, когда интерфейс определяется в файлах html, а логика приложения - в коде javascript.

Объявление пользовательского интерфейса в файлах XML позволяет отделить интерфейс приложения от кода. Что означает, что мы можем изменять определение интерфейса без изменения кода java. Например, в приложении могут быть определены разметки в файлах XML для различных ориентаций монитора, различных размеров устройств, различных языков и т.д. Кроме того, объявление разметки в XML позволяет легче визуализировать структуру интерфейса и облегчает отладку.

Файлы разметки графического интерфейса располагаются в проекте в каталоге res/layout. По умолчанию при создании проекта уже есть один файл ресурсов разметки activity_main.xml, который может выглядеть примерно так:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
```

```

xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context="com.example.eugene.viewsapplication.MainActivity">

```

```

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

```

```

</android.support.constraint.ConstraintLayout>

```

В файле определяются все графические элементы и их атрибуты, которые составляют интерфейс. При создании разметки в XML следует соблюдать некоторые правила: каждый файл разметки должен содержать один корневой элемент, который должен представлять объект View или ViewGroup.

В данном случае корневым элементом является элемент ConstraintLayout, который содержит элемент TextView.

При компиляции каждый XML-файл разметки компилируется в ресурс View. Загрузка ресурса разметки осуществляется в методе Activity.onCreate. Чтобы установить разметку для текущего объекта activity, надо в метод setContentView в качестве параметра передать ссылку на ресурс разметки.

Для получения ссылки на ресурс в коде java необходимо использовать выражение R.layout.[название_ресурса]. Название ресурса layout будет совпадать с именем файла, поэтому чтобы использовать файл activity_main.xml в качестве источника визуального интерфейса, нам надо изменить код MainActivity следующим образом:

```

package com.example.eugene.viewsapplication;

```

```

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

```

```

public class MainActivity extends AppCompatActivity {

```

```

    @Override

```

```

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

```

```

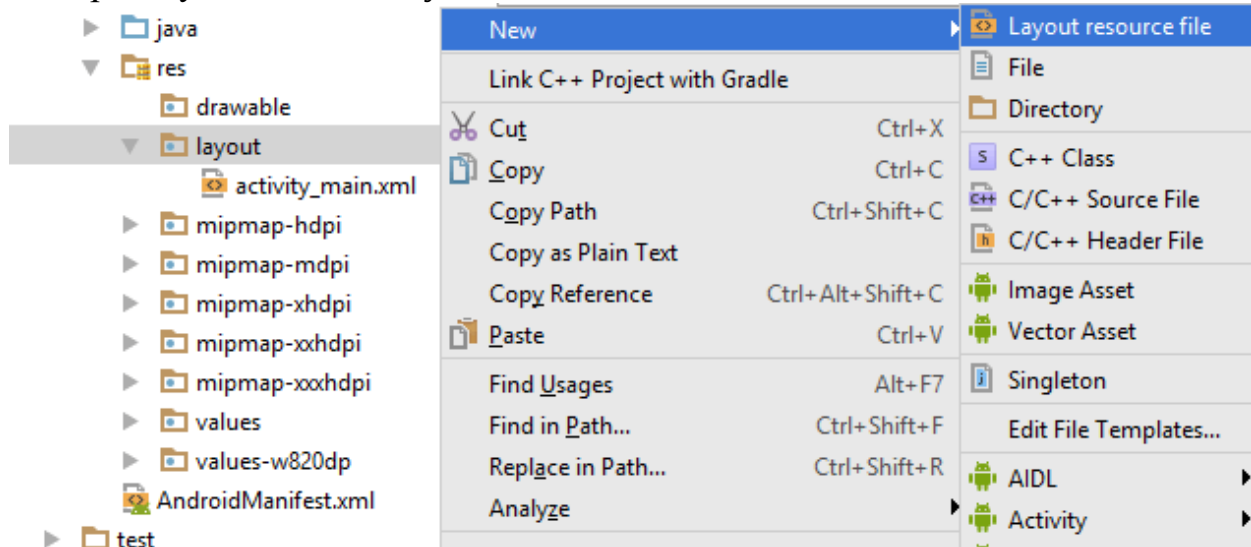
}

```

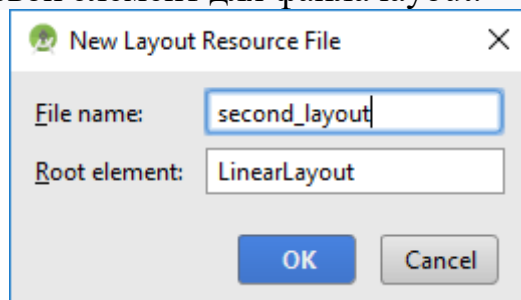
Добавление файла layout

Но у нас может быть и несколько различных ресурсов layout. Как правило, каждый отдельный класс Activity использует свой файл layout. Либо для одного класса Activity может использоваться сразу несколько различных файлов layout.

К примеру, добавим в проект новый файл разметки интерфейса. Для этого нажмем на папку res/layout правой кнопкой мыши и в появившемся меню выберем пункт New -> Layout resource file:



После этого в специальном окошке будет предложено указать имя и корневой элемент для файла layout:



В качестве названия укажем second_layout, а в качестве типа корневого элемента оставим LinearLayout.

После этого в папку res/layout будет добавлен новый файл second_layout.xml, с которым мы можем работать точно также, как и с activity_main.xml. В частности, откроем файл second_layout.xml и изменим его содержимое следующим образом:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView android:id="@+id/header"
        android:text="Second Activity"
        android:textSize="26dp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
```

</LinearLayout>

Здесь определено текстовое поле `TextView`, которое выводит на экран простейший текст "Second Activity".

Получение элементов в коде и их идентификаторы

Кроме текста, ширины и высоты текстовое поле устанавливает еще один важный атрибут - `id`. Знак `+` в записи `android:id="@+id/header"` означает, что если для элемента не определен `id` со значением `header`, то его следует определить.

Чтобы использовать этот файл в качестве основного интерфейса, перейдем к `MainActivity` и изменим ее код:

```
package com.example.eugene.viewsapplication;
```

```
import android.support.v7.app.AppCompatActivity;
```

```
import android.os.Bundle;
```

```
import android.widget.TextView;
```

```
public class MainActivity extends AppCompatActivity {
```

```
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);
```

```
        // устанавливаем в качестве интерфейса файл second_layout.xml  
        setContentView(R.layout.second_layout);
```

```
        // получаем элемент textView
```

```
        TextView textView = (TextView) findViewById(R.id.header);
```

```
        // переустанавливаем у него текст
```

```
        textView.setText("Hello Android 7!");
```

```
    }
```

```
}
```

С помощью метода `setContentView()` устанавливается разметка из файла `second_layout.xml`.

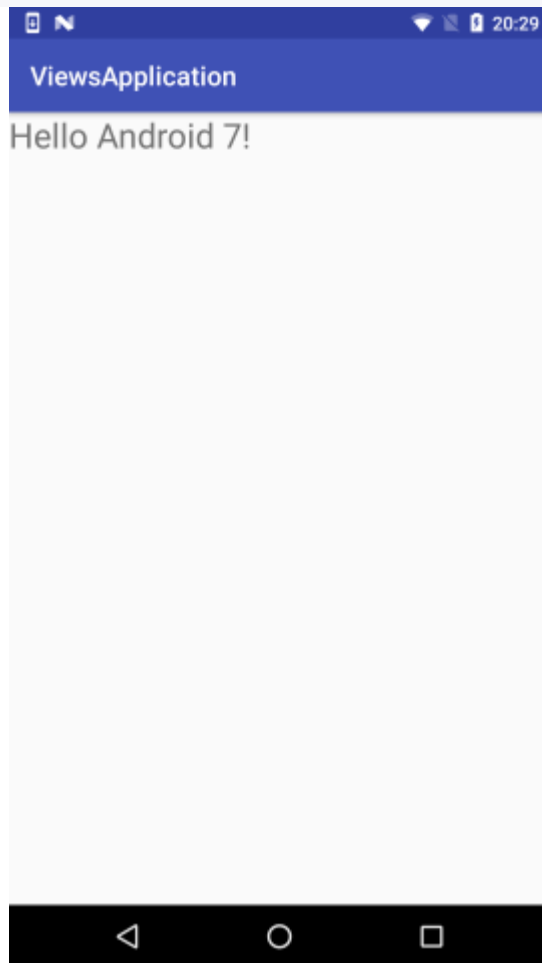
Другой важный момент, который стоит отметить - получение визуального элемента `TextView`. Так как в его коде мы определили атрибут `android:id`, то через этот `id` мы можем его получить.

Для получения элементов по `id` класс `Activity` имеет метод `findViewById()`. В этот метод передается идентификатор ресурса в виде `R.id.[идентификатор_элемента]`. Этот метод возвращает объект `View` - объект базового класса для всех элементов, поэтому результат метода еще необходимо привести к типу `TextView`.

Далее мы можем что-то сделать с этим элементом, в данном случае изменяем его текст.

Причем что важно, получение элемента происходит после того, как в методе `setContentView` была установлена разметка, в которой этот визуальный элемент был определен.

И если мы запустим проект, то увидим, что TextView выводит новый текст:



Определение размеров

В ОС Android мы можем использовать различные типы измерений:

px: пиксели текущего экрана. Однако эта единица измерения не рекомендуется, так как реальное представление внешнего вида может изменяться в зависимости от устройства; каждое устройство имеет определенный набор пикселей на дюйм, поэтому количество пикселей на экране может также меняться

dp: (device-independent pixels) независимые от плотности экрана пиксели. Абстрактная единица измерения, основанная на физической плотности экрана с разрешением 160 dpi (точек на дюйм). В этом случае 1dp = 1px. Если размер экрана больше или меньше, чем 160dpi, количество пикселей, которые применяются для отрисовки 1dp соответственно увеличивается или уменьшается. Например, на экране с 240 dpi 1dp=1,5px, а на экране с 320dpi 1dp=2px. Общая формула для получения количества физических пикселей из dp: $px = dp * (dpi / 160)$

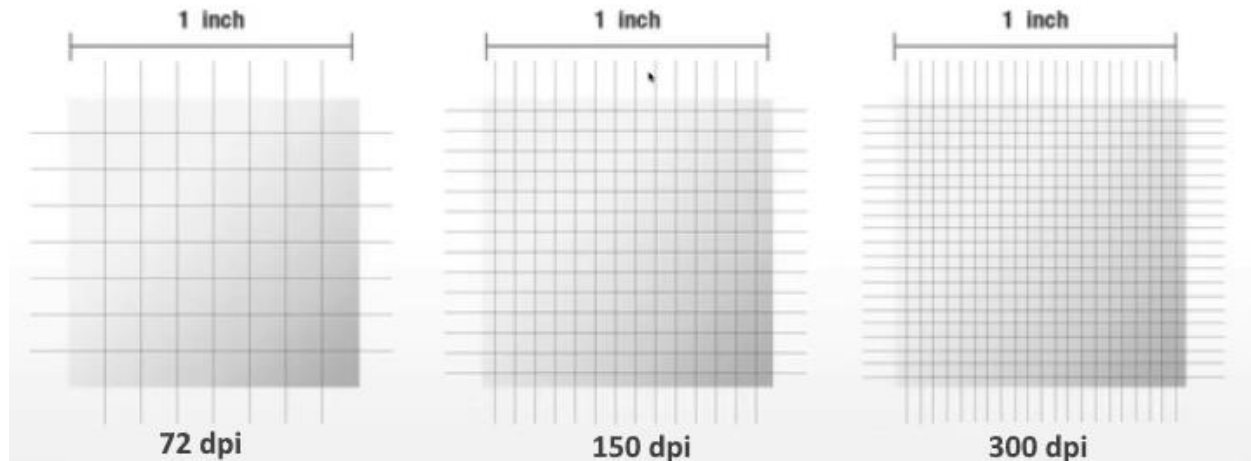
sp: (scale-independent pixels) независимые от масштабирования пиксели. Допускают настройку размеров, производимую пользователем. Рекомендуются для работы со шрифтами.

pt: 1/72 дюйма, базируются на физических размерах экрана

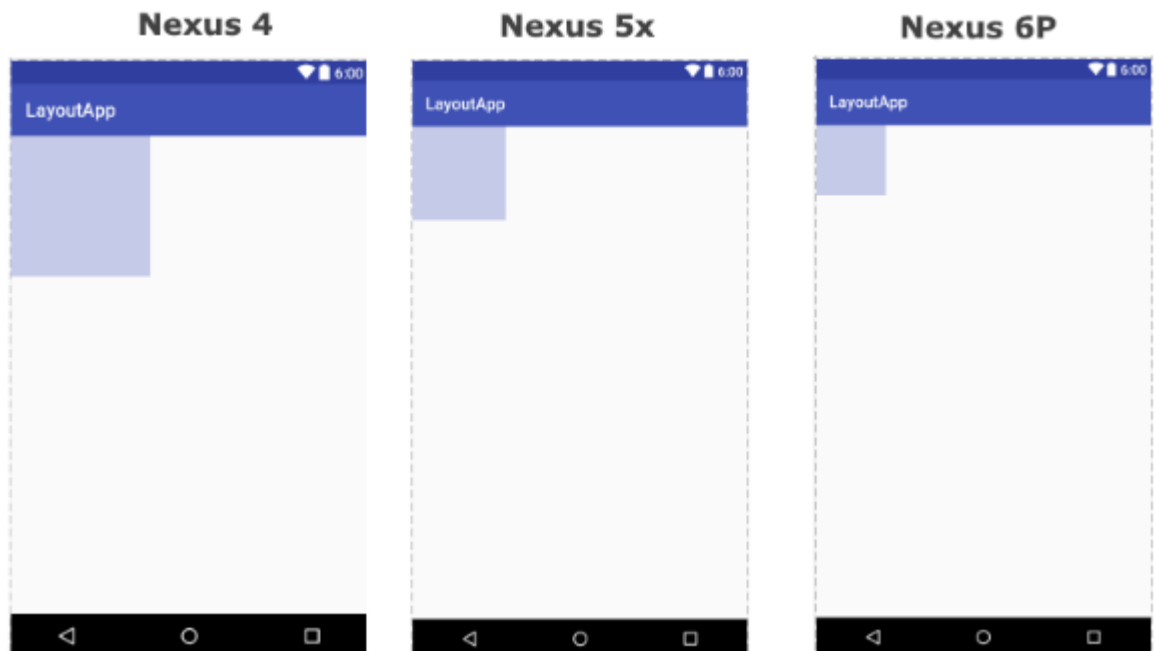
mm: миллиметры

in: дюймы

Предпочтительными единицами для использования являются dp. Это связано с тем, что мир мобильных устройств на Android сильно фрагментирован в плане разрешения и размеров экрана. И чем больше плотность пикселей на дюйм, тем соответственно больше пикселей нам будет доступно:

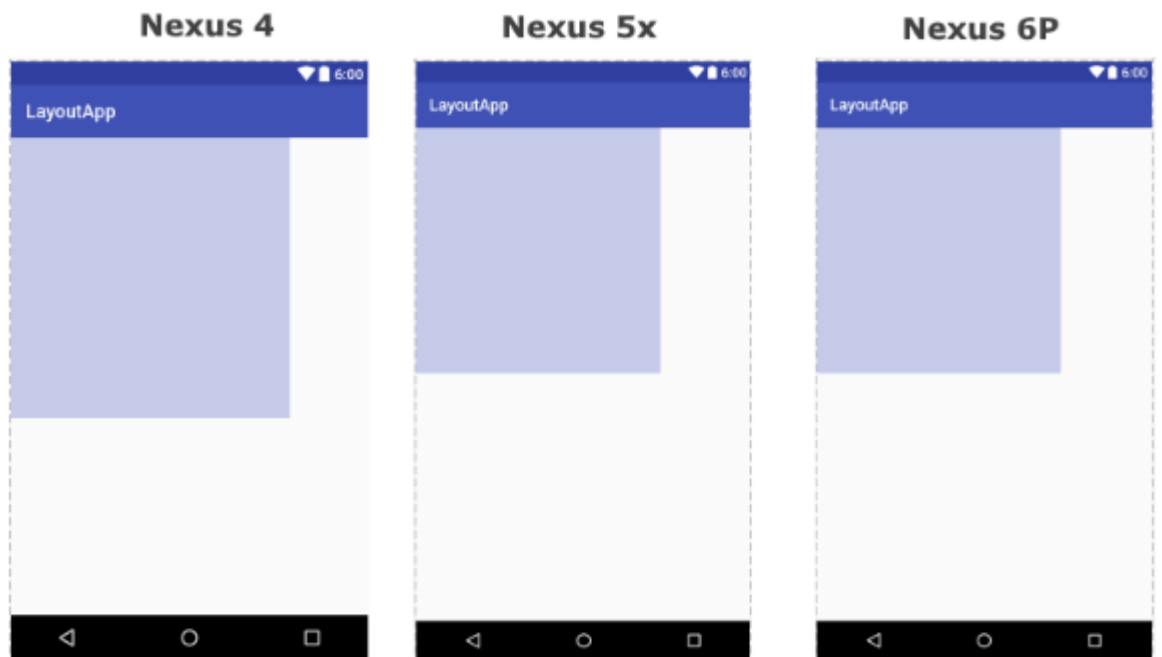


Используя же стандартные физические пиксели мы можем столкнуться с проблемой, что размеры элементов также будут сильно варьироваться в зависимости от плотности пикселей устройства. Например, возьмем 3 устройства с различными характеристиками экрана Nexus 4, Nexus 5X и Nexus 6P и выведем на экран квадрат размером 300px на 300px:



В одном случае квадрат по ширине будет занимать 40%, в другом - треть ширины, в третьем - 20%.

Теперь также возьмем квадрат со сторонами 300x300, но теперь вместо физических пикселей используем единицы dp:



Теперь же размеры квадрата на разных устройствах выглядят более консистентно.

Для упрощения работы с размерами все размеры разбиты на несколько групп:

- ldpi (low): ~120dpi

- mdpi (medium): ~160dpi

- hdpi (high): ~240dpi (к данной группе можно отнести такое устройство как Nexus One)

- xhdpi (extra-high): ~320dpi (Nexus 4)

- xxhdpi (extra-extra-high): ~480dpi (Nexus 5/5X, Samsung Galaxy S5)

- xxxhdpi (extra-extra-extra-high): ~640dpi (Nexus 6/6P, Samsung Galaxy S6)

Ширина и высота элементов

Все визуальные элементы, которые мы используем в приложении, как правило, упорядочиваются на экране с помощью контейнеров. В Android подобными контейнерами служат такие классы как `RelativeLayout`, `LinearLayout`, `GridLayout`, `TableLayout`, `ConstraintLayout`, `FrameLayout`. Все они по разному располагают элементы и управляют ими, но есть некоторые общие моменты при компоновке визуальных компонентов, которые мы сейчас рассмотрим.

Для организации элементов внутри контейнера используются параметры разметки. Для их задания в файле `xml` используются атрибуты, которые начинаются с префикса `layout_`. В частности, к таким параметрам относятся атрибуты `layout_height` и `layout_width`, которые используются для установки размеров и могут принимать одно из следующих значений:

- точные размеры элемента, например 96 dp

- значение `wrap_content`: элемент растягивается до тех границ, которые достаточны, чтобы вместить все его содержимое

значение `match_parent`: элемент заполняет всю область родительского контейнера

Например:

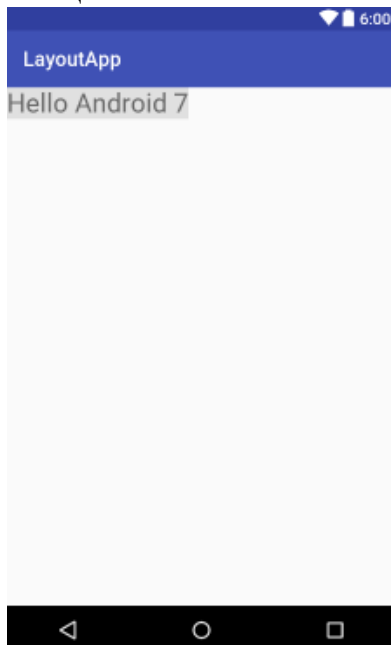
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/activity_main"

    android:layout_width="match_parent"
    android:layout_height="match_parent"
    >

    <TextView
        android:text="Hello Android 7"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:textSize="26sp"
        android:background="#e0e0e0" />
</RelativeLayout>
```

Контейнер самого верхнего уровня, в качестве которого в данном случае выступает `RelativeLayout`, для высоты и ширины имеет значение `match_parent`, то есть он будет заполнять всю область для `activity` - как правило, весь экран.

А элемент `TextView` растягивается до тех значений, которые достаточны для размещения его текста.

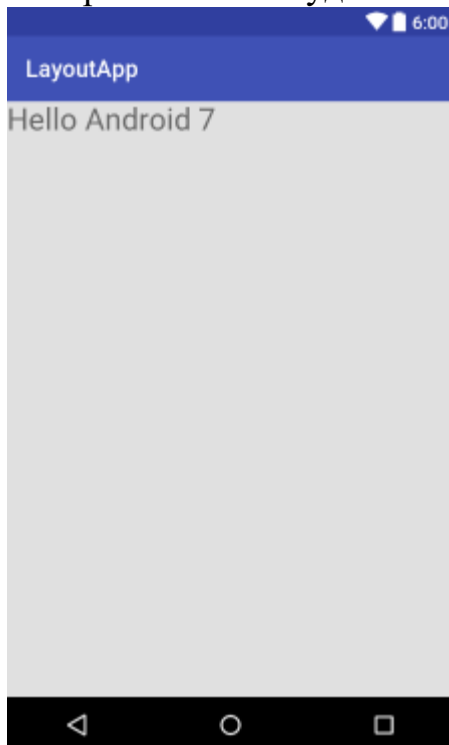


Теперь изменим высоту и ширину на `match_parent`:

```
<TextView
    android:text="Hello Android 7"
    android:layout_height="match_parent"
    android:layout_width="match_parent"
    android:textSize="26sp"
```

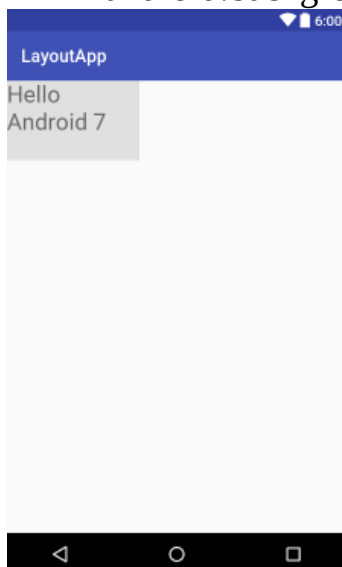
```
android:background="#e0e0e0" />
```

Теперь TextView будет заполнять все пространство контейнера:



Либо мы можем установить точные значения:

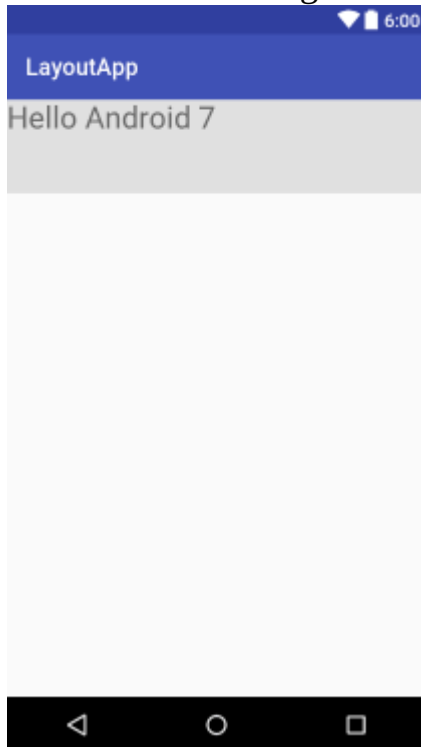
```
<TextView  
    android:text="Hello Android 7"  
    android:layout_height="90dp"  
    android:layout_width="150dp"  
    android:textSize="26sp"  
    android:background="#e0e0e0" />
```



Также мы можем комбинировать несколько значений, например, растянуть по ширине и установить точные значения для высоты:

```
<TextView  
    android:text="Hello Android 7"  
    android:layout_height="80dp"
```

```
android:layout_width="match_parent"
android:textSize="26sp"
android:background="#e0e0e0" />
```



Если для установки ширины и длины используется значение `wrap_content`, то мы можем дополнительно ограничить минимальные и максимальные значения с помощью атрибутов `minWidth/maxWidth` и `minHeight/maxHeight`:

```
<TextView android:text="Hello Android 7"
```

```
    android:minWidth="200dp"
    android:maxWidth="250dp"
    android:minHeight="100dp"
    android:maxHeight="200dp"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:textSize="26sp"
    android:background="#e0e0e0" />
```

В этом случае ширина `TextView` будет такой, которая достаточна для вмещения текста, но не больше значения `maxWidth` и не меньше значения `minWidth`. То же самое для установки высоты.

Программная установка ширины и высоты

Если элемент, к примеру, тот же `TextView` создается в коде `java`, то для установки высоты и ширины можно использовать метод `setLayoutParams()`. Так, изменим код `MainActivity`:

```
package com.example.eugene.layoutapp;
```

```
import android.support.v7.app.AppCompatActivity;
```

```

import android.os.Bundle;
import android.view.ViewGroup;
import android.widget.RelativeLayout;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        RelativeLayout relativeLayout = new RelativeLayout(this);
        TextView textView1 = new TextView(this);
        textView1.setText("Hello Android 7");
        textView1.setTextSize(26);

        // устанавливаем размеры
        textView1.setLayoutParams(new ViewGroup.LayoutParams
            (ViewGroup.LayoutParams.WRAP_CONTENT,
            ViewGroup.LayoutParams.WRAP_CONTENT));
        // добавляем TextView в RelativeLayout
        relativeLayout.addView(textView1);
        setContentView(relativeLayout);
    }
}

```

В метод `setLayoutParams()` передается объект `ViewGroup.LayoutParams`. Этот объект инициализируется двумя параметрами: шириной и высотой. Для указания ширины и высоты можно использовать одну из констант `ViewGroup.LayoutParams.WRAP_CONTENT` или `ViewGroup.LayoutParams.MATCH_PARENT`.

Также мы можем передать точные значения или комбинировать типы значений:

```

textView1.setLayoutParams(new
    ViewGroup.LayoutParams(ViewGroup.LayoutParams.MATCH_PARENT,
    200));

```

Внутренние и внешние отступы

Параметры разметки позволяют задать отступы как от внешних границ элемента до границ контейнера, так и внутри самого элемента между его границами и содержимым.

Padding

Для установки внутренних отступов применяется атрибут `android:padding`. Он устанавливает отступы контента от всех четырех сторон контейнера. Можно устанавливать отступы только от одной стороны контейнера, применяя следующие атрибуты: `android:paddingLeft`, `android:paddingRight`, `android:paddingTop` и `android:paddingBottom`.

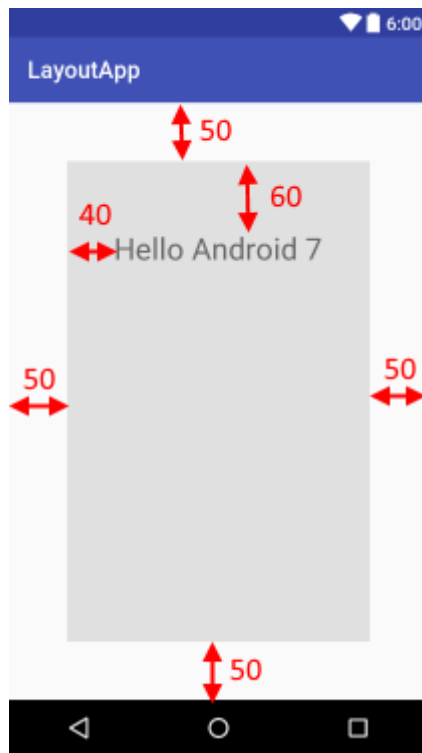
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="50dp">

    <TextView android:text="Hello Android 7"
        android:layout_height="match_parent"
        android:layout_width="match_parent"
        android:textSize="26sp"
        android:background="#e0e0e0"
        android:paddingTop="60dp"
        android:paddingLeft="40dp"/>

</RelativeLayout>
```

У контейнера `RelativeLayout` установлен только один общий внутренний отступ в 50 единиц.

У `TextView` установлен отступ между верхней границей элемента и его внутренним содержимым (то есть текстом) в 60 единиц и отступ слева в 40 единиц:



Margin

Для установки внешних отступов используется атрибут `layout_margin`. Данный атрибут имеет модификации, которые позволяют задать отступ только от одной стороны: `android:layout_marginBottom`, `android:layout_marginTop`, `android:layout_marginLeft` и `android:layout_marginRight` (отступы соответственно от нижней, верхней, левой и правой границ):

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<RelativeLayout
```

```
xmlns:android="http://schemas.android.com/apk/res/android"
```

```
android:id="@+id/activity_main"
```

```
android:layout_width="match_parent"
```

```
android:layout_height="match_parent"
```

```
android:paddingTop="50dp">
```

```
<TextView android:text="Hello Android 7"
```

```
android:layout_height="match_parent"
```

```
android:layout_width="match_parent"
```



```
android:layout_marginTop="50dp"
android:layout_marginBottom="60dp"
android:layout_marginLeft="60dp"
android:layout_marginRight="60dp

android:textSize="26sp"
android:background="#e0e0e0"/>
```

</RelativeLayout>

Здесь у TextView задаются отступы от каждой стороны RelativeLayout. И стоит отметить, что у самого RelativeLayout установлен внутренний отступ сверху в 50 единиц, который также учитывается при формировании внешнего отступа сверху у TextView, поэтому общее расстояние от верхней границы TextView до верхнего края RelativeLayout будет составлять 100 единиц:



Программная установка отступов

Для программной установки внутренних отступов у элементы вызывается метод `setPadding(left, top, right, bottom)`, в который передаются четыре значения для каждой из сторон.

Для установки внешних отступов необходимо реализовать объект `LayoutParams` для того контейнера, который применяется. И затем вызвать у этого объекта `LayoutParams` метод `setMargins(left, top, right, bottom)`:

```
package com.example.eugene.layoutapp;
```

```
import android.support.v7.app.AppCompatActivity;
```

```
import android.os.Bundle;
```

```
import android.view.ViewGroup;
```

```
import android.widget.RelativeLayout;
```

```
import android.widget.TextView;
```

```
public class MainActivity extends AppCompatActivity {
```

```
    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
```

```
        RelativeLayout relativeLayout = new RelativeLayout(this);
```

```
        TextView textView1 = new TextView(this);
```

```
        textView1.setBackgroundColor(0xFFBDBDBD);
```

```
        textView1.setText("Hello Android 7");
```

```
        textView1.setTextSize(26);
```

```
        RelativeLayout.LayoutParams layoutParams = new  
        RelativeLayout.LayoutParams
```

```
            (ViewGroup.LayoutParams.MATCH_PARENT, 200);
```

```
        // установка внешних отступов
```

```

        layoutParams.setMargins(30,40,50,60);
        // устанавливаем размеры
        textView1.setLayoutParams(layoutParams);
        // установка внутренних отступов
        textView1.setPadding(30,30,30,30);
        // добавляем TextView в RelativeLayout
        relativeLayout.addView(textView1);
        setContentView(relativeLayout);
    }
}

```

LinearLayout

Контейнер LinearLayout представляет объект ViewGroup, который упорядочивает все дочерние элементы в одном направлении: по горизонтали или по вертикали. Все элементы расположены один за другим. Направление разметки указывается с помощью атрибута android:orientation.

Если, например, ориентация разметки вертикальная (android:orientation="vertical"), то все элементы располагаются в столбик - по одному элементу на каждой строке. Если ориентация горизонтальная (android:orientation="horizontal"), то элементы располагаются в одну строку. Например, расположим элементы в горизонтальный ряд:

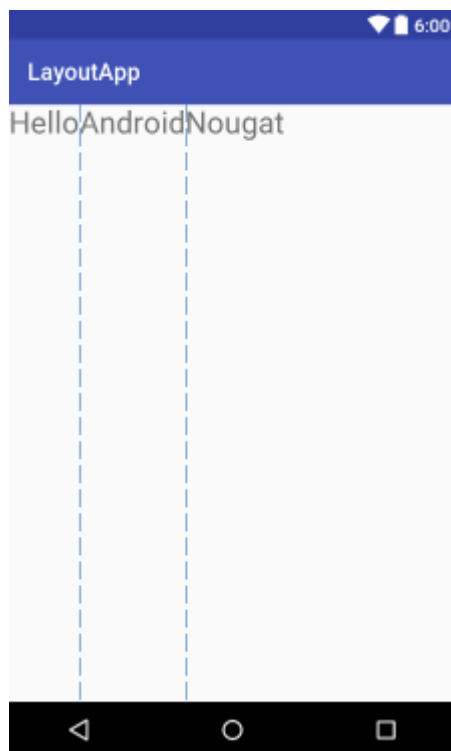
```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal" >

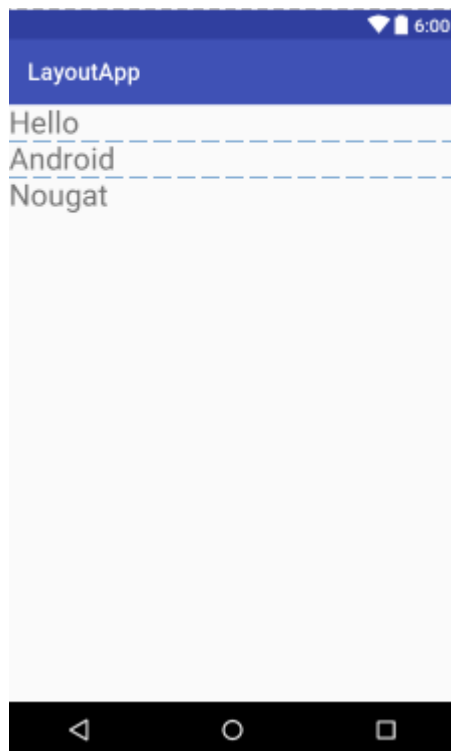
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"

```

```
        android:text="Hello"
        android:textSize="26sp" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Android"
        android:textSize="26sp" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Nougat"
        android:textSize="26sp" />
</LinearLayout>
```



Если бы мы указали для LinearLayout атрибут `android:orientation="vertical"`, то элементы размещались бы по вертикали:



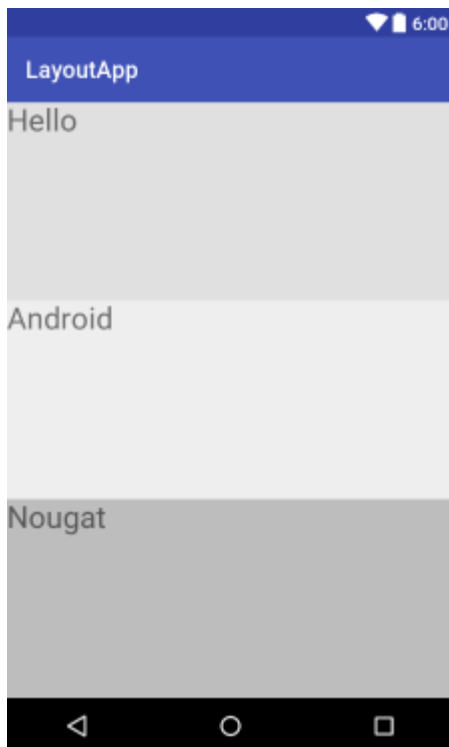
LinearLayout поддерживает такое свойство, как вес элемента, которое передается атрибутом `android:layout_weight`. Это свойство принимает значение, указывающее, какую часть оставшегося свободного места контейнера по отношению к другим объектам займет данный элемент. Например, если один элемент у нас будет иметь для свойства `android:layout_weight` значение 2, а другой - значение 1, то в сумме они дадут 3, поэтому первый элемент будет занимать $\frac{2}{3}$ оставшегося пространства, а второй - $\frac{1}{3}$.

Если все элементы имеют значение `android:layout_weight="1"`, то все эти элементы будут равномерно распределены по всей площади контейнера:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <TextView
        android:layout_width="match_parent"
```

```
        android:layout_height="0dp"
        android:text="Hello"
        android:background="#e0e0e0"
        android:layout_weight="1"
        android:textSize="26sp" />
    <TextView
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:background="#eeeeee"
        android:text="Android"
        android:layout_weight="1"
        android:textSize="26sp" />
    <TextView
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:text="Nougat"
        android:background="#bdbdbd"
        android:layout_weight="1"
        android:textSize="26sp" />
</LinearLayout>
```

В данном случае LinearLayout имеет вертикальную ориентацию, поэтому все элементы будут располагаться сверху вниз. Все три элемента имеют значение `android:layout_weight="1"`, поэтому сумма весов всех элементов будет равна 3, а каждый элемент получит по трети пространства в LinearLayout:



При этом так как у нас вертикальный стек, то нам надо также установить для свойства `layout_height` значение `0dp`. Если бы `LinearLayout` имел горизонтальную ориентацию, то для свойства `layout_width` надо было бы установить значение `0dp`.

Еще один атрибут `android:weightSum` позволяет указать сумму весов всех элементов. Например:

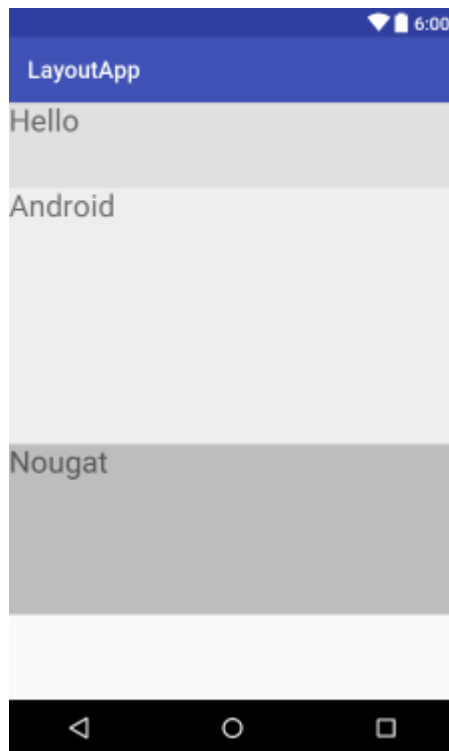
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:weightSum="7">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:text="Hello"
```

```
        android:background="#e0e0e0"
        android:layout_weight="1"
        android:textSize="26sp" />
<TextView
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:background="#eeeeee"
    android:text="Android"
    android:layout_weight="3"
    android:textSize="26sp" />
<TextView
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:text="Nougat"
    android:background="#bdbdbd"
    android:layout_weight="2"
    android:textSize="26sp" />
</LinearLayout>
```

LinearLayout здесь задает сумму весов равную 7. То есть все пространство по вертикали (так как вертикальная ориентация) условно делится на семь равных частей.

Первый TextView имеет вес 1, то есть из этих семи частей занимает только одну. Второй TextView имеет вес 3, то есть занимает три части из семи. И третий имеет вес 2. Итоговая сумма составляет 6. Но так как LinearLayout задает вес 7, то одна часть будет свободна от всех элементов.



Программное создание LinearLayout

Создание LinearLayout в коде MainActivity:

```
package com.example.eugene.layoutapp;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.LinearLayout;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        LinearLayout linearLayout = new LinearLayout(this);
        linearLayout.setOrientation(LinearLayout.VERTICAL);
        TextView textView1 = new TextView(this);
        textView1.setText("Hello");
```

```

        linearLayout.addView(textView1,                                new
        LinearLayout.LayoutParams
            (LinearLayout.LayoutParams.WRAP_CONTENT,
        LinearLayout.LayoutParams.WRAP_CONTENT));

        TextView textView2 = new TextView(this);
        textView2.setText("Android");
        linearLayout.addView(textView2,                                new
        LinearLayout.LayoutParams
            (LinearLayout.LayoutParams.WRAP_CONTENT,
        LinearLayout.LayoutParams.WRAP_CONTENT));

        setContentView(linearLayout);
    }
}

```

RelativeLayout

RelativeLayout представляет объект ViewGroup, который располагает дочерние элементы относительно позиции других дочерних элементов разметки или относительно области самой разметки RelativeLayout. Используя относительное позиционирование, мы можем установить элемент по правому краю или в центре или иным способом, который предоставляет данный контейнер. Для установки элемента в файле xml мы можем применять следующие атрибуты:

- android:layout_above: располагает элемент над элементом с указанным Id
- android:layout_below: располагает элемент под элементом с указанным Id
- android:layout_toLeftOf: располагается слева от элемента с указанным Id
- android:layout_toRightOf: располагается справа от элемента с указанным Id
- android:layout_alignBottom: выравнивает элемент по нижней границе другого элемента с указанным Id
- android:layout_alignLeft: выравнивает элемент по левой границе другого элемента с указанным Id
- android:layout_alignRight: выравнивает элемент по правой границе другого элемента с указанным Id
- android:layout_alignTop: выравнивает элемент по верхней границе другого элемента с указанным Id

android:layout_alignBaseline: выравнивает базовую линию элемента по базовой линии другого элемента с указанным Id

android:layout_alignParentBottom: если атрибут имеет значение true, то элемент прижимается к нижней границе контейнера

android:layout_alignParentRight: если атрибут имеет значение true, то элемент прижимается к правому краю контейнера

android:layout_alignParentLeft: если атрибут имеет значение true, то элемент прижимается к левому краю контейнера

android:layout_alignParentTop: если атрибут имеет значение true, то элемент прижимается к верхней границе контейнера

android:layout_centerInParent: если атрибут имеет значение true, то элемент располагается по центру родительского контейнера

android:layout_centerHorizontal: при значении true выравнивает элемент по центру по горизонтали

android:layout_centerVertical: при значении true выравнивает элемент по центру по вертикали

Например, позиционирование относительно контейнера RelativeLayout:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView android:text="Left Top"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:textSize="26sp"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true" />

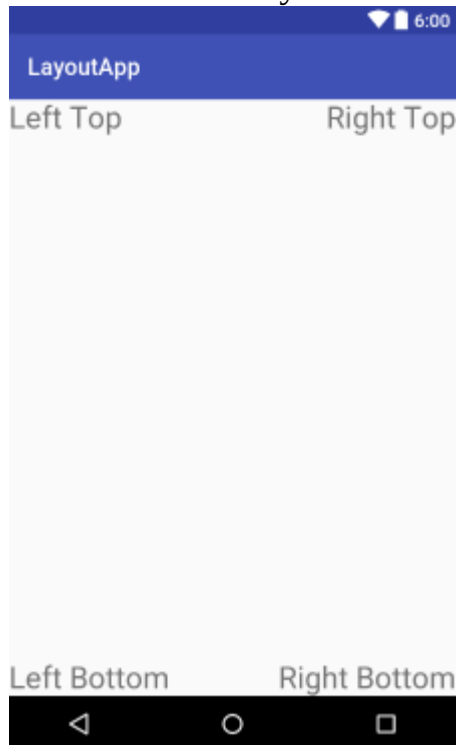
    <TextView android:text="Right Top"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:textSize="26sp"
        android:layout_alignParentRight="true"
        android:layout_alignParentTop="true" />

    <TextView android:text="Left Bottom"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:textSize="26sp"
        android:layout_alignParentLeft="true"
        android:layout_alignParentBottom="true" />
```

```

<TextView android:text="Right Bottom"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:textSize="26sp"
    android:layout_alignParentRight="true"
    android:layout_alignParentBottom="true" />
</RelativeLayout>

```



Для позиционирования относительно другого элемента, нам надо указать id этого элемента. Так, поместим на RelativeLayout текстовое поле и кнопку:

```

<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <EditText
        android:id="@+id/edit_message"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"/>

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Отправить"
        android:layout_alignRight="@id/edit_message"

```

```
        android:layout_below="@id/edit_message"  
    />  
</RelativeLayout>
```

В данном случае поле EditText располагается по центру в RelativeLayout, а кнопка помещается под EditText и выравнивается по его правой границе:



Создадим элемент RelativeLayout программно в коде MainActivity:

```
package com.example.eugene.layoutapp;  
  
import android.support.v7.app.AppCompatActivity;  
import android.os.Bundle;  
import android.widget.Button;  
import android.widget.EditText;  
import android.widget.RelativeLayout;  
  
public class MainActivity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
  
        RelativeLayout relativeLayout = new RelativeLayout(this);  
  
        EditText editText = new EditText(this);  
        editText.setId(editText.generateViewId());  
  
        Button button = new Button(this);  
        button.setText("Отправить");
```

```

        // устанавливаем параметры положения для EditText
        RelativeLayout.LayoutParams editTextParams = new
RelativeLayout.LayoutParams(
            RelativeLayout.LayoutParams.MATCH_PARENT,
            RelativeLayout.LayoutParams.WRAP_CONTENT
        );
        editTextParams.addRule(RelativeLayout.CENTER_IN_PARENT);
        relativeLayout.addView(editText, editTextParams);

        // устанавливаем параметры положения для Button
        RelativeLayout.LayoutParams buttonParams = new
RelativeLayout.LayoutParams(
            RelativeLayout.LayoutParams.WRAP_CONTENT,
            RelativeLayout.LayoutParams.WRAP_CONTENT
        );
        buttonParams.addRule(RelativeLayout.BELOW, editText.getId());
        buttonParams.addRule(RelativeLayout.ALIGN_RIGHT,
editText.getId());
        relativeLayout.addView(button, buttonParams);

        setContentView(relativeLayout);
    }
}

```

Чтобы задать положение элемента в контейнере, применяется класс `RelativeLayout.LayoutParams`. Через конструктор устанавливаются значения для для ширины и высоты. Например, у элемента `EditText` для ширины устанавливается значение `MATCH_PARENT`, а для высоты - `WRAP_CONTENT`.

С помощью метода `addRule()` мы можем добавлять дополнительные правила для позиционирования элемента. Этот метод в качестве параметра принимает числовую константу, которая представляет параметр позиционирования и которая аналогична атрибуту. Например, атрибуту `android:layout_centerInParent` будет соответствовать константа `CENTER_IN_PARENT`, а атрибуту `android:layout_alignRight` константа `ALIGN_RIGHT`.

Стоит отметить, что в целях упрощения кода для установки `id` у `EditText` вызывается метод `generateViewId()`, который позволяет программно сгенерировать `id` для элемента управления.

Затем установленный `id` передается в качестве второго параметра в метод `addRule` при установке правил для кнопки:

```
buttonParams.addRule(RelativeLayout.BELOW, editText.getId());
```

Тем самым мы указываем относительно какого элемента надо задать расположение.

ЛЕКЦИЯ №31-№32. БАЗОВЫЕ ЭЛЕМЕНТЫ УПРАВЛЕНИЯ.

TextView

Для простого вывода текста на экран предназначен элемент TextView. Он просто отображает текст без возможности его редактирования. Некоторые его основные атрибуты:

android:text: устанавливает текст элемента

android:textSize: устанавливает высоту текста, в качестве единиц измерения для указания высоты используются sp

android:background: задает фоновый цвет элемента в виде цвета в шестнадцатичной записи или в виде цветового ресурса

android:textColor: задает цвет текста

android:textAllCaps: при значении true делает все символы в тексте заглавными

android:textDirection: устанавливает направление текста. По умолчанию используется направление слева направо, но с помощью значения rtl можно установить направление справа налево

android:textAlignment: задает выравнивание текста. Может принимать следующие значения:

center: выравнивание по центру

textStart: по левому краю

textEnd: по правому краю

viewStart: по левому краю

viewEnd: по правому краю

android:fontFamily: устанавливает тип шрифта. Может принимать следующие значения:

monospace

serif

serif-monospace

sans-serif

sans-serif-condensed

sans-serif-smallcaps

sans-serif-light

casual

cursive

cursive

Например, определим три текстовых поля:

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<LinearLayout
```

```
xmlns:android="http://schemas.android.com/apk/res/android"
```

```
android:id="@+id/activity_main"
```

```
android:orientation="vertical"
```

```
android:layout_width="match_parent"
```

```
android:layout_height="match_parent">
```

```
<TextView
    android:layout_height="wrap_content"
    android:layout_width="match_parent"
    android:layout_margin="10dp"

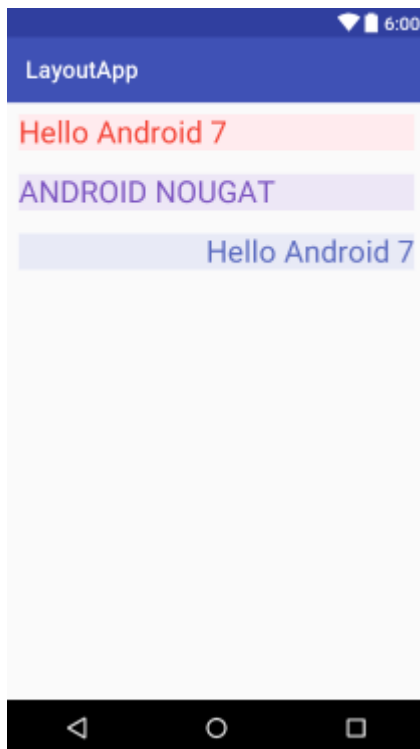
    android:text="Hello Android 7"
    android:fontFamily="sans-serif"
    android:textSize="26sp"
    android:background="#ffebee"
    android:textColor="#f44336" />

<TextView
    android:layout_height="wrap_content"
    android:layout_width="match_parent"
    android:layout_margin="10dp"

    android:text="Android Nougat"
    android:textAllCaps="true"
    android:textSize="26sp"
    android:background="#ede7f6"
    android:textColor="#7e57c2" />

<TextView
    android:layout_height="wrap_content"
    android:layout_width="match_parent"
    android:layout_margin="10dp"

    android:text="Hello Android 7"
    android:textAlignment="textEnd"
    android:textSize="26sp"
    android:background="#e8eaf6"
    android:textColor="#5c6bc0" />
</LinearLayout>
```

Установка элемента в коде тоже не отличается сложностью. Например, создадим элемент и выведем его на экран:

```
package com.example.eugene.layoutapp;
```

```
import android.support.v7.app.AppCompatActivity;  
import android.os.Bundle;  
import android.view.ViewGroup;  
import android.widget.LinearLayout;  
import android.widget.TextView;  
import android.graphics.Typeface;
```

```
public class MainActivity extends AppCompatActivity {
```

```
    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);
```

```
        LinearLayout linearLayout = new LinearLayout(this);
```

```
        TextView textView1 = new TextView(this);
```

```
        // установка фонового цвета
```

```
        textView1.setBackgroundColor(0xffe8eaf6);
```

```
        // установка цвета текста
```

```
        textView1.setTextColor(0xff5c6bc0);
```

```
        // делаем все буквы заглавными
```

```
        textView1.setAllCaps(true);
```

```
        // устанавливаем выравнивание текста по центру
```

```

        textView1.setTextAlignment(TextView.TEXT_ALIGNMENT_CENTER)
;
        // устанавливаем текста
        textView1.setText("Android Nougat 7");
        // установка шрифта
        textView1.setTypeface(Typeface.create("casual", Typeface.NORMAL));
        // устанавливаем высоту текста
        textView1.setTextSize(26);

        LinearLayout.LayoutParams layoutParams = new
        LinearLayout.LayoutParams
            (ViewGroup.LayoutParams.MATCH_PARENT,
            ViewGroup.LayoutParams.WRAP_CONTENT);
        // установка внешних отступов
        layoutParams.setMargins(20,20,20,20);
        // устанавливаем размеры
        textView1.setLayoutParams(layoutParams);
        linearLayout.addView(textView1);
        setContentView(linearLayout);
    }
}

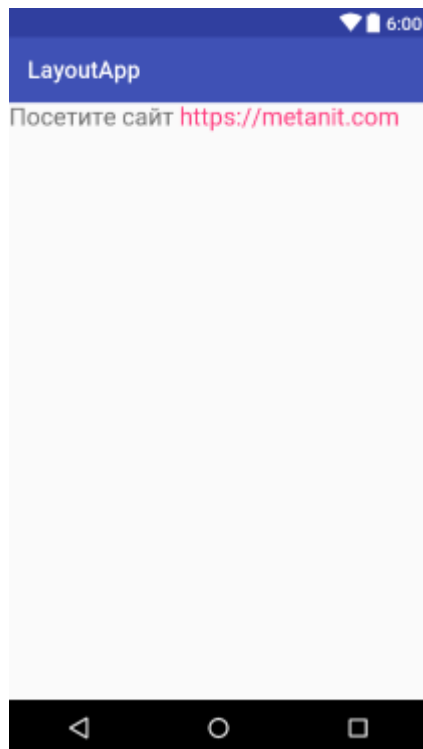
```

Иногда необходимо вывести на экран какую-нибудь ссылку, либо телефон, по нажатию на которые производилось бы определенное действие. Для этого в TextView определен атрибут android:autoLink:

```

<TextView android:id="@+id/display_message"
    android:text="Посетите сайт https://metanit.com"
    android:textSize="21sp"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:autoLink="web|email" />

```



`android:autoLink` может принимать несколько значений:

`none`: отключает все ссылки

`web`: включает все веб-ссылки

`email`: включает ссылки на электронные адреса

`phone`: включает ссылки на номера телефонов

`map`: включает ссылки на карту

`all`: включает все вышеперечисленные ссылки

То есть при настройке `android:autoLink="web"` если в тексте есть упоминание адреса url, то этот адрес будет выделяться, а при нажатии на него будет осуществлен переход к веб-браузеру, который откроет страницу по этому адресу. С помощью прямой черты мы можем объединять условия, как в данном случае: `android:autoLink="web|email"`

EditText

Элемент `EditText` является подклассом класса `TextView`. Он также представляет текстовое поле, но теперь уже с возможностью ввода и редактирования текста. Таким образом, в `EditText` мы можем использовать все те же возможности, что и в `TextView`.

Из тех атрибутов, что не рассматривались в теме про `TextView`, следует отметить атрибут `android:hint`. Он позволяет задать текст, который будет отображаться в качестве подсказки, если элемент `EditText` пуст. Кроме того, мы можем использовать атрибут `android:inputType`, который позволяет задать клавиатуру для ввода. В частности, среди его значений можно выделить следующие:

`text`: обычная клавиатура для ввода однострочного текста

`textMultiLine`: многострочное текстовое поле

`textEmailAddress`: обычная клавиатура, на которой присутствует символ `@`, ориентирована на ввод email

textUri: обычная клавиатура, на которой присутствует символ /, ориентирована на ввод интернет-адресов

textPassword: клавиатура для ввода пароля

textCapWords: при вводе первый введенный символ слова представляет заглавную букву, остальные - строчные

number: числовая клавиатура

phone: клавиатура в стиле обычного телефона

date: клавиатура для ввода даты

time: клавиатура для ввода времени

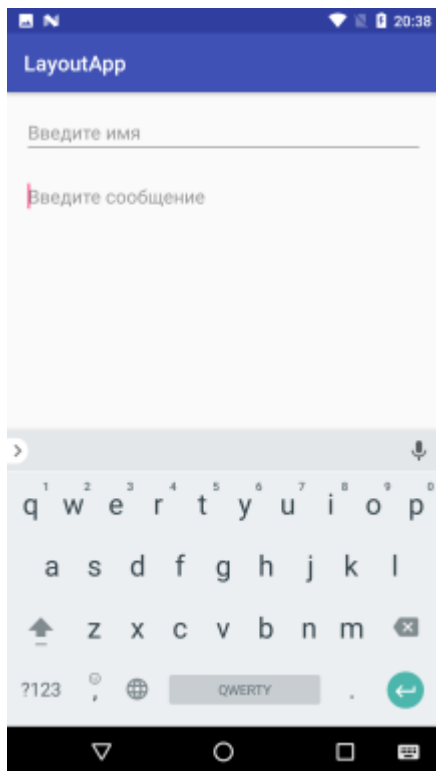
datetime: клавиатура для ввода даты и времени

Используем EditText:

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp">

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Введите имя" />
    <EditText
        android:layout_marginTop="16dp"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:hint="Введите сообщение"
        android:inputType="textMultiLine"
        android:gravity="top" />
</LinearLayout>
```

Первое поле здесь обычное однострочное, а второе - многострочное. Чтобы во втором поле текст выравнивался по верху, дополнительно устанавливается атрибут android:gravity="top".



Одной из возможностей элемента EditText также является возможность обработать введенные символы по мере ввода пользователя. Для этого определим в файле activity_main.xml следующую разметку:

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp">

    <TextView
        android:id="@+id/textView"
        android:textSize="34sp"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Введите имя"
        android:id="@+id/editText" />

</LinearLayout>
```

Предполагается, что введенные в EditText символы тут же будут отображаться в элементе TextView. И для этого также изменим код MainActivity:

```
package com.example.eugene.layoutapp;
```

```

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.textEditable;
import android.text.TextWatcher;
import android.widget.EditText;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_main);

        EditText editText = (EditText) findViewById(R.id.editText);

        editText.addTextChangedListener(new TextWatcher() {

            public void afterTextChanged(Editable s) {}

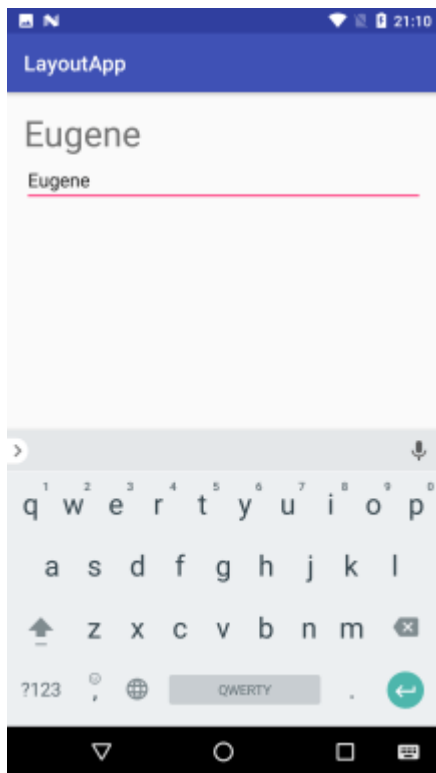
            public void beforeTextChanged(CharSequence s, int start,
                                           int count, int after) {}

            public void onTextChanged(CharSequence s, int start,
                                       int before, int count) {
                TextView textView = (TextView)
                findViewById(R.id.textView);
                textView.setText(s);
            }
        });
    }
}

```

С помощью метода `addTextChangedListener()` здесь к элементу `EditText` добавляется слушатель ввода текста - объект `TextWatcher`. Для его использования нам надо реализовать три метода, но в реальности нам хватит реализации метода `onTextChanged`, который вызывается при изменении текста. Введенный текст передается в этот метод в качестве параметра `CharSequence`. В самом методе просто передаем этот текст в элемент `TextView`.

В итоге при вводе в `EditText` все символы также будут отображаться в `TextView`:



Button

Одним из часто используемых элементов являются кнопки, которые представлены классом `android.widget.Button`. Ключевой особенностью кнопок является возможность взаимодействия с пользователем через нажатия.

Некоторые ключевые атрибуты, которые можно задать у кнопок:

`text`: задает текст на кнопке

`textColor`: задает цвет текста на кнопке

`background`: задает фоновый цвет кнопки

`textAllCaps`: при значении `true` устанавливает текст в верхнем регистре.

По умолчанию как раз и применяется значение `true`

`onClick`: задает обработчик нажатия кнопки

Итак, изменим код в `activity_main.xml` следующим образом:

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp">

    <TextView
        android:id="@+id/textView"
        android:textSize="34sp"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
    <EditText
        android:layout_width="match_parent"
```

```

        android:layout_height="wrap_content"
        android:hint="Введите имя"
        android:id="@+id/editText" />
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Click"
    android:onClick="sendMessage" />

```

```
</LinearLayout>
```

При помощи атрибута `android:onClick` можно задать метод в коде `java`, который будет обрабатывать нажатия кнопки. Так, в вышеприведенном примере это метод `sendMessage`. Теперь перейдем к коду `MainActivity` и пропишем в нем такой метод:

```

package com.example.eugene.layoutapp;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    // Обработка нажатия кнопки
    public void sendMessage(View view) {
        TextView textView = (TextView) findViewById(R.id.textView);
        EditText editText = (EditText) findViewById(R.id.editText);
        textView.setText("Добро пожаловать, " + editText.getText());
    }
}

```

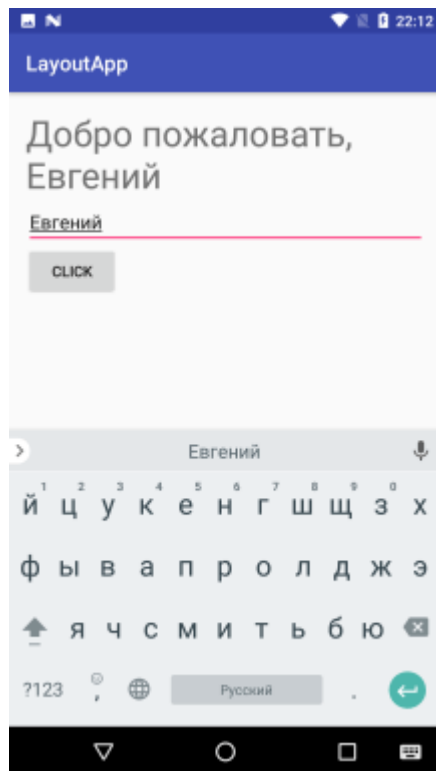
При создании метода обработки нажатия следует учитывать следующие моменты:

- Метод должен объявляться с модификатором `public`

- Должен возвращать значение `void`

- В качестве параметра принимать объект `View`. Этот объект `View` и представляет собой нажатую кнопку

- В данном случае после нажатия на кнопку в `TextView` выводится текст из `EditText`.



Аналогичный пример полностью в коде MainActivity:

```
package com.example.eugene.layoutapp;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.LinearLayout;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    EditText editText;
    TextView textView;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        LinearLayout linearLayout = new LinearLayout(this);
        linearLayout.setOrientation(LinearLayout.VERTICAL);
        textView = new TextView(this);
        textView.setLayoutParams(new LinearLayout.LayoutParams(
            LinearLayout.LayoutParams.MATCH_PARENT,
            LinearLayout.LayoutParams.WRAP_CONTENT
        ));
        linearLayout.addView(textView);
```

```

        editText = new EditText(this);
        editText.setHint("Введите имя");
        editText.setLayoutParams(new LinearLayout.LayoutParams(
            LinearLayout.LayoutParams.MATCH_PARENT,
            LinearLayout.LayoutParams.WRAP_CONTENT
        ));
        linearLayout.addView(editText);

        Button button = new Button(this);
        button.setText("CLICK");
        button.setLayoutParams(new LinearLayout.LayoutParams(
            LinearLayout.LayoutParams.WRAP_CONTENT,
            LinearLayout.LayoutParams.WRAP_CONTENT
        ));
        linearLayout.addView(button);

        button.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                // Обработка нажатия
                textView.setText("Добро пожаловать, " +
editText.getText());
            }
        });

        setContentView(linearLayout);
    }
}

```

При программном создании кнопки мы можем определить у нее слушатель нажатия `View.OnClickListener` и с помощью его метода `onClick` также обработать нажатие:

```

        button.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                // Обработка нажатия
            }
        });

```

Всплывающие окна. Toast

Для создания простых уведомлений в Android используется класс `Toast`. Фактически `Toast` представляет всплывающее окно с некоторым текстом, которое отображается в течение некоторого времени.

Объект `Toast` нельзя создать в коде разметки `xml`, например, в файл `activity_main.xml`. `Toast` можно использовать только в коде `java`.

Так, определим в файле разметки `activity_main.xml` кнопку:

```

<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/activity_main"

```

```
android:layout_width="match_parent"
android:layout_height="match_parent"
android:padding="16dp">
```

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Click"
    android:onClick="onClick" />
```

```
</RelativeLayout>
```

У кнопки установлен обработчик нажатия - метод `onClick`. Определим его в коде `MainActivity`:

```
package com.example.eugene.layoutapp;
```

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;
```

```
public class MainActivity extends AppCompatActivity {
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

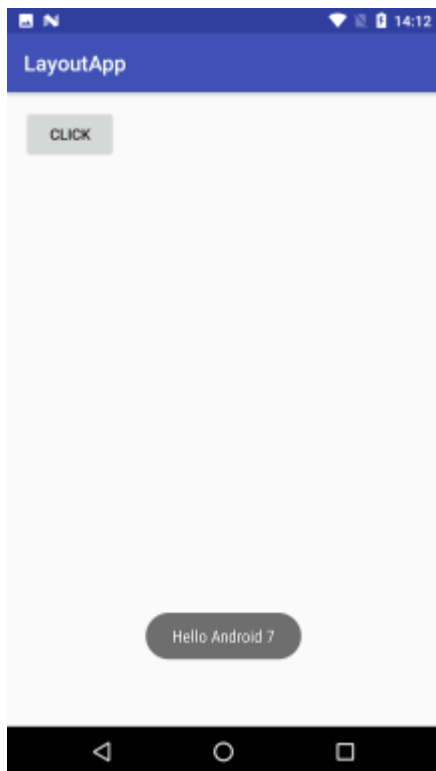
```
        setContentView(R.layout.activity_main);
    }
```

```
    public void onClick(View view){
        Toast toast = Toast.makeText(this, "Hello Android
7",Toast.LENGTH_LONG);
        toast.show();
    }
}
```

В обработчике отображается всплывающее окно. Для его создания применяется метод `Toast.makeText()`, в который передается три параметра: текущий контекст (текущий объект `activity`), отображаемый текст и время отображения окна.

В качестве времени показа окна мы можем использовать целочисленное значение - количество миллисекунд или встроенные константы `Toast.LENGTH_LONG` (3500 миллисекунд) и `Toast.LENGTH_SHORT` (2000 миллисекунд).

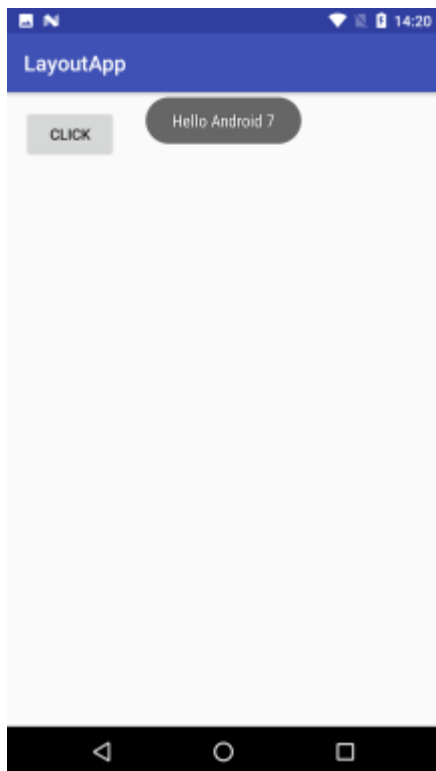
Для самого отображения окна вызывается метод `show()`:



По умолчанию окно отображается внизу интерфейса с центрированием по центру. Но мы можем кастомизировать позиционирование окна с помощью методов `setGravity()` и `setMargin()`. Так, изменим метод `onClick`:

```
public void onClick(View view){  
  
    Toast toast = Toast.makeText(this, "Hello Android  
7", Toast.LENGTH_LONG);  
    toast.setGravity(Gravity.TOP, 0, 160);  
    toast.show();  
}
```

Первый параметр метода `setGravity` указывает, в какой части контейнера надо позиционировать `Toast`, второй и третий параметр устанавливают отступы от этой позиции по горизонтали и вертикали соответственно:

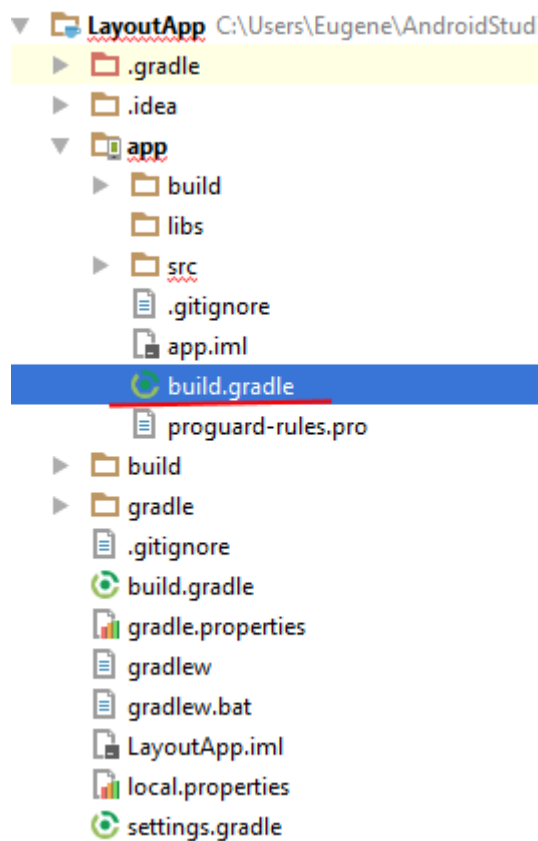


Метод `setMargin()` принимает два параметра: отступ от левой границы контейнера в процентах от ширины контейнера и отступ от верхней границы в процентах от длины контейнера.

Snackbar

Элемент `Snackbar` в некотором роде похож на `Toast`: он также позволяет выводить всплывающие сообщения, но теперь сообщения растягиваются по ширине экрана.

При создании проекта по типу `Empty Activity` по умолчанию функциональность `Snackbar` может быть недоступна, нам ее надо будет добавить. Для этого перейдем к файлу `build.gradle`, который находится внутри модуля `app`:



Он выглядит примерно так:

```
apply plugin: 'com.android.application'
```

```
android {
    compileSdkVersion 25
    buildToolsVersion "23.0.1"
    defaultConfig {
        applicationId "com.example.eugene.layoutapp"
        minSdkVersion 16
        targetSdkVersion 25
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner
        "android.support.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
            'proguard-rules.pro'
        }
    }
}

dependencies {
```

```

        compile fileTree(dir: 'libs', include: ['*.jar'])
        androidTestCompile('com.android.support.test.espresso:espresso-
core:2.2.2', {
            exclude group: 'com.android.support', module: 'support-
annotations'
        })
        compile 'com.android.support:appcompat-v7:25.1.0'
        testCompile 'junit:junit:4.12'
    }

```

В конец секции dependencies добавим нужный нам пакет:

```

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    androidTestCompile('com.android.support.test.espresso:espresso-
core:2.2.2', {
        exclude group: 'com.android.support', module: 'support-annotations'
    })
    compile 'com.android.support:appcompat-v7:25.1.0'
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:design:25.1.0'
}

```

Следует отметить, что для компиляции в данном случае применяется API 25 (compileSdkVersion 25), поэтому и у пакета версия начинается с этого числа: 25.1.0.

После изменения файла вверху в Android Studio появится сообщение о необходимости выполнить синхронизацию, и для этого нажмем на ссылку Sync Now:



После завершения синхронизации изменим файл activity_main.xml:

```

<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="CLICK"
        android:onClick="onClick"/>

</RelativeLayout>

```

Здесь определена кнопка, по нажатию на которую будет отображаться сообщение.

И также изменим класс MainActivity:

```

package com.example.eugene.layoutapp;

import android.os.Bundle;
import android.support.design.widget.Snackbar;
import android.support.v7.app.AppCompatActivity;
import android.view.View;

public class MainActivity extends AppCompatActivity {

```



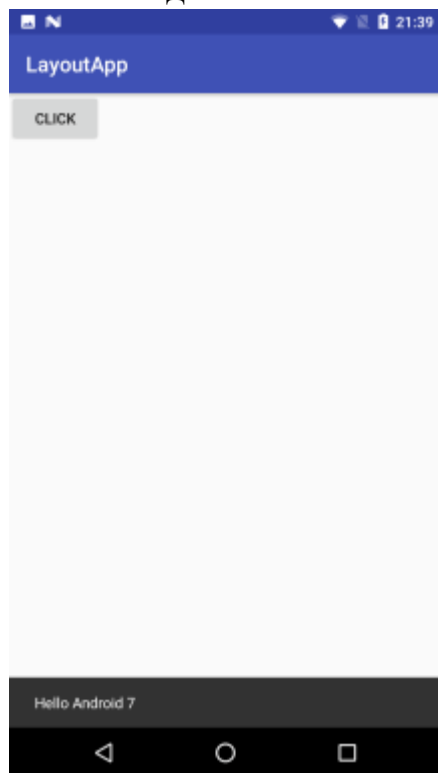
```

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
}
public void onClick(View view){
    Snackbar.make(view, "Hello Android 7",
Snackbar.LENGTH_LONG)
        .show();
}
}

```

Snackbar создается с помощью метода `make()`, в который передаются три параметра: объект `View`, к которому прикрепляется всплывающее сообщение, само сообщение в виде строки и параметр, который указывает, сколько будет отображаться сообщение. Последний параметр может принимать числовое значение - количество миллисекунд, либо одну из трех констант: `Snackbar.LENGTH_INDEFINITE` (отображение в течение неопределенного периода времени), `Snackbar.LENGTH_LONG` (долгое отображение) или `Snackbar.LENGTH_SHORT` (недолгое отображение).

После создания Snackbar отображается с помощью метода `show`:



При этом в отличие от `Toast` мы не можем повлиять на позицию сообщения, оно отображается внизу экрана и занимает всю нижнюю часть.

Checkbox

Элементы `Checkbox` представляют собой флажки, которые могут находиться в отмеченном и неотмеченном состоянии. Флажки позволяют производить множественный выбор из нескольких значений. Итак, определим в файле разметки `activity_main.xml` несколько флажков:

```

<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp">

    <TextView android:id="@+id/selection"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="26sp" />
    <CheckBox android:id="@+id/java"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Java"
        android:textSize="26sp"
        android:onClick="onCheckboxClicked"/>

    <CheckBox android:id="@+id/javascript"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="JavaScript"
        android:textSize="26sp"
        android:onClick="onCheckboxClicked"/>

```

```
</LinearLayout>
```

Атрибут `android:onClick`, как и в случае с простыми кнопками, позволяет задать обработчик нажатия на флажок. Определим обработчик нажатия в коде `MainActivity`:

```

package com.example.eugene.layoutapp;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.CheckBox;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}

```

```

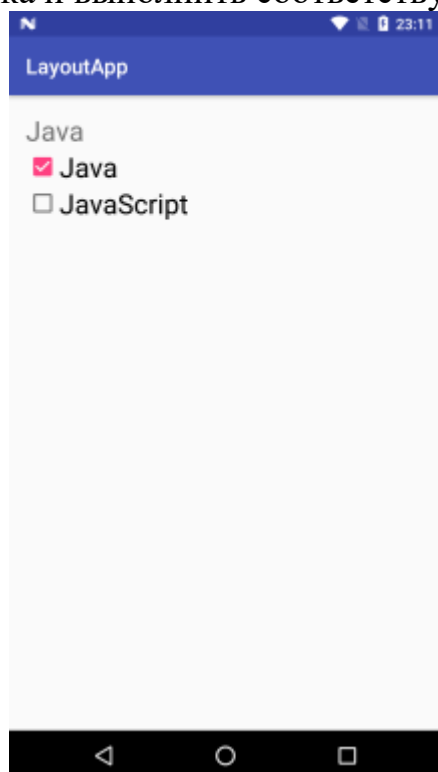
public void onCheckboxClicked(View view) {
    // Получаем флажок
    CheckBox language = (CheckBox) view;
    // Получаем, отмечен ли данный флажок
    boolean checked = language.isChecked();

    TextView selection = (TextView) findViewById(R.id.selection);
    // Смотрим, какой именно из флажков отмечен
    switch(view.getId()) {
        case R.id.java:
            if (checked){
                selection.setText("Java");
            }
            break;
        case R.id.javascript:
            if (checked)
                selection.setText("JavaScript");
            break;
    }
}
}

```

В качестве параметра в обработчик нажатия `onCheckboxClicked` передается нажатый флажок. С помощью метода `isChecked()` можно узнать, выделен ли флажок - в этом случае метод возвращает `true`.

С помощью конструкции `switch...case` можно получить `id` нажатого флажка и выполнить соответствующие действия.



Правда, если нам просто надо взять текст из выбранного флажка, то необязательно в данном случае использовать конструкцию switch, так как мы можем сократить весь код следующим образом:

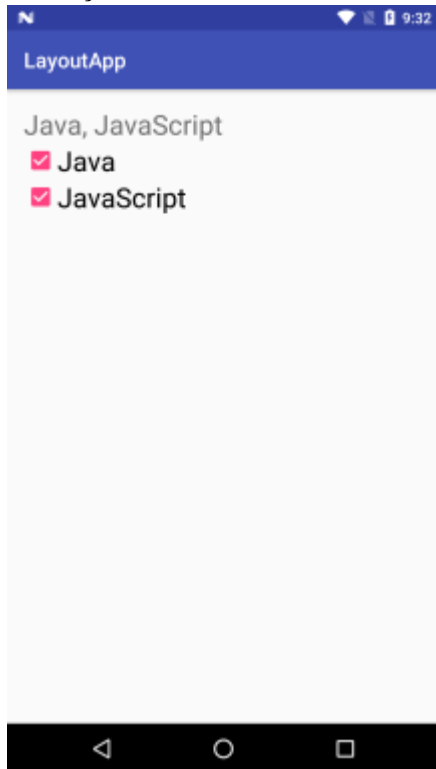
```
public void onCheckboxClicked(View view) {  
    // Получаем флажок  
    CheckBox language = (CheckBox) view;  
    // Получаем, отмечен ли данный флажок  
    TextView selection = (TextView) findViewById(R.id.selection);  
    if(language.isChecked())  
        selection.setText(language.getText());  
}
```

Но в данном случае у нас есть проблема: в текстовом поле отображается только один выделенный элемент. Изменим код MainActivity, чтобы отображать оба выделенных элемента:

```
package com.example.eugene.layoutapp;  
  
import android.support.v7.app.AppCompatActivity;  
import android.os.Bundle;  
import android.view.View;  
import android.widget.CheckBox;  
import android.widget.TextView;  
  
public class MainActivity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
  
        setContentView(R.layout.activity_main);  
    }  
  
    public void onCheckboxClicked(View view) {  
  
        // Получаем флажки  
        CheckBox java = (CheckBox) findViewById(R.id.java);  
        CheckBox javascript = (CheckBox)  
findViewById(R.id.javascript);  
        String selectedItems = "";  
        if(java.isChecked())  
            selectedItems += java.getText() + ", ";  
        if(javascript.isChecked())  
            selectedItems += javascript.getText();  
  
        TextView selection = (TextView) findViewById(R.id.selection);  
        selection.setText(selectedItems);  
    }  
}
```

}

}



OnCheckedChangeListener

С помощью слушателя `OnCheckedChangeListener` можно отслеживать изменения флажка. Этот слушатель срабатывает, когда мы устанавливаем или убираем отметку на флажке. Например, определим следующий `checkbox`:

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp">

    <CheckBox android:id="@+id/enabled"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:checked="true"
        android:text="Включить"
        android:textSize="26sp" />

</LinearLayout>
```

В коде `MainActivity` подключим обработчик изменения состояния:

```
package com.example.eugene.layoutapp;
```

```

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.CheckBox;
import android.widget.CompoundButton;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_main);

        CheckBox enableBox = (CheckBox)
findViewById(R.id.enabled);

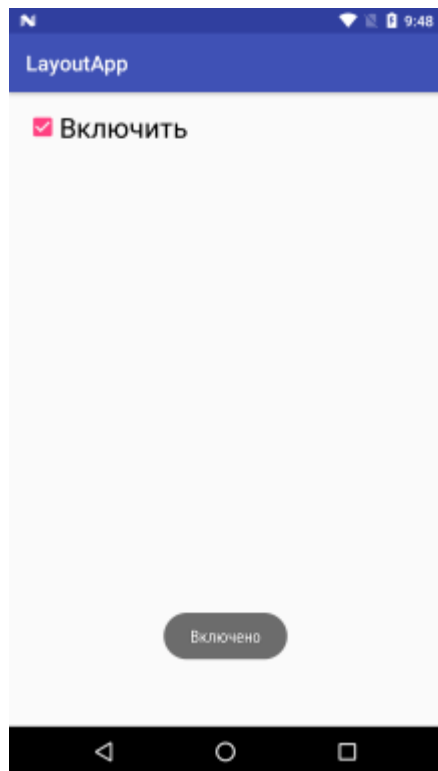
        enableBox.setOnCheckedChangeListener(new
CompoundButton.OnCheckedChangeListener() {
            public void onCheckedChanged(CompoundButton buttonView,
boolean isChecked) {

                if (isChecked) {
                    Toast.makeText(getApplicationContext(), "Включено",
Toast.LENGTH_SHORT).show();
                } else {
                    Toast.makeText(getApplicationContext(), "Выключено",
Toast.LENGTH_SHORT).show();
                }
            }
        });
    }
}

```

Слушатель `OnCheckedChangeListener` определен в базовом классе `CompoundButton` и определяет один метод - `onCheckedChanged`. Первый параметр этого метода `buttonView` - сам измененный флажок `CheckBox`. А второй параметр `isChecked` указывает, отмечен ли флажок.

При изменении состояния флажка будет выводиться во всплывающем окне соответствующее уведомление:



ToggleButton

ToggleButton подобно элементу CheckBox может пребывать в двух состояниях: отмеченном и неотмеченном, причем для каждого состояния мы можем отдельно установить свой текст. Например, определим следующий элемент ToggleButton:

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical" >

<ToggleButton
    android:id="@+id/toggle"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textOn="Включено"
    android:textOff="Выключено"
    android:onClick="onToggleClicked"/>
```

```
</LinearLayout>
```

Атрибуты android:textOn и android:textOff задают текст кнопки в отмеченном и неотмеченном состоянии соответственно. И также, как и для других кнопок, мы можем обработать нажатие на элемент с помощью события onClick. В этом случае определим в классе Activity обработчик события:

```
package com.example.eugene.layoutapp;
```

```

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;
import android.widget.ToggleButton;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

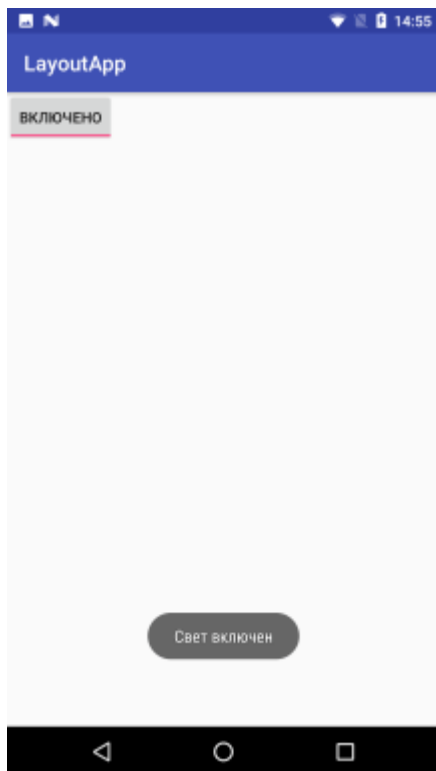
        setContentView(R.layout.activity_main);
    }

    public void onToggleClicked(View view) {

        // включена ли кнопка
        boolean on = ((ToggleButton) view).isChecked();

        if (on) {
            // действия если включена
            Toast.makeText(this, "Свет включен",
Toast.LENGTH_LONG).show();
        } else {
            // действия, если выключена
            Toast.makeText(this, "Свет выключен!",
Toast.LENGTH_LONG).show();
        }
    }
}

```

Создание элемента `ToggleButton` в коде java:

```
package com.example.eugene.layoutapp;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.LinearLayout;
import android.widget.Toast;
import android.widget.ToggleButton;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        LinearLayout linearLayout = new LinearLayout(this);
        ToggleButton toggleButton = new ToggleButton(this);
        toggleButton.setTextOff("Выключено");
        toggleButton.setTextOn("Включено");
        toggleButton.setText("Выключено");
        toggleButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                boolean on = ((ToggleButton) view).isChecked();

                if (on) {
```

```

        Toast.makeText(getApplicationContext(),
включен", Toast.LENGTH_LONG).show();
    } else {
        Toast.makeText(getApplicationContext(),
выключен!", Toast.LENGTH_LONG).show();
    }
}
});
linearLayout.addView(toggleButton,
LinearLayout.LayoutParams
    (LinearLayout.LayoutParams.WRAP_CONTENT,
LinearLayout.LayoutParams.WRAP_CONTENT));
setContentView(linearLayout);
}
}
RadioButton

```

Схожую с флажками функциональность предоставляют переключатели, которые представлены классом `RadioButton`. Но в отличие от флажков одновременно в группе переключателей мы можем выбрать только один переключатель.

Чтобы создать список переключателей для выбора, вначале надо создать объект `RadioGroup`, который будет включать в себя все переключатели:

```

<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <TextView android:id="@+id/selection"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="26sp"
    />

    <RadioGroup
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/radios"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical" >

        <RadioButton android:id="@+id/java"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Java"

```

```

        android:onClick="onRadioButtonClicked"/>
<RadioButton android:id="@+id/javascript"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="JavaScript"
        android:onClick="onRadioButtonClicked"/>
</RadioGroup>

```

```
</LinearLayout>
```

Поскольку класс `RadioGroup` является производным от `LinearLayout`, то мы также можем задать вертикальную или горизонтальную ориентацию списка, при том включив в него не только собственно переключатели, но и другие объекты, например, кнопку или `TextView`.

В классе `MainActivity` определим обработку выбора переключателей:

```

package com.example.eugene.layoutapp;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.RadioButton;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    public void onRadioButtonClicked(View view) {
        // если переключатель отмечен
        boolean checked = ((RadioButton) view).isChecked();
        TextView selection = (TextView) findViewById(R.id.selection);
        // Получаем нажатый переключатель
        switch(view.getId()) {
            case R.id.java:
                if (checked){
                    selection.setText("Выбран Java");
                }
                break;
            case R.id.javascript:
                if (checked){
                    selection.setText("Выбран JavaScript");
                }
            }
        }
    }
}

```

```

    }
    }
    break;
}
}
}

```



OnCheckedChangeListener

Кроме обработки нажатия на каждый отдельный переключатель мы можем в целом повесить на весь RadioGroup с его переключателями слушатель OnCheckedChangeListener и обрабатывать в нем нажатия. Для этого уберем из разметки у переключателей атрибуты android:onClick, а у элемента RadioGroup определим id:

```

<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <TextView android:id="@+id/selection"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="26sp"
    />

    <RadioGroup
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/radios"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"

```

```

        android:orientation="vertical">

        <RadioButton android:id="@+id/java"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Java"/>
        <RadioButton android:id="@+id/javascript"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="JavaScript"/>
    </RadioGroup>

```

```
</LinearLayout>
```

Далее в коде MainActivity повесим на объект RadioGroup слушатель OnCheckedChangeListener:

```
package com.example.eugene.layoutapp;
```

```

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.TextView;

```

```

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // получаем объект RadioGroup
        RadioGroup radGrp = (RadioGroup)findViewById(R.id.radios);
        // обработка переключения состояния переключателя
        radGrp.setOnCheckedChangeListener(new
RadioGroup.OnCheckedChangeListener() {
            @Override
            public void onCheckedChanged(RadioGroup arg0, int id) {
                TextView selection = (TextView)
findViewById(R.id.selection);
                switch(id) {
                    case R.id.java:
                        selection.setText("Выбран Java");
                        break;
                    case R.id.javascript:
                        selection.setText("Выбран JavaScript");

```

```

        break;
    default:
        break;
    }
    });
}
}

```

Слушатель `RadioGroup.OnCheckedChangeListener` определяет метод `onCheckedChanged()`, в который передается объект `RadioGroup` и `id` выделенного переключателя. Далее также мы можем проверить `id` и выполнить определенную обработку.

`atePicker` и хронометр

`DatePicker` и `TimePicker` представляет собой элементы для выбора даты и времени соответственно. Итак, создадим разметку для этих элементов:

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView android:id="@+id/dateDefault"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

    <DatePicker android:id="@+id/datePicker"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

    <TextView android:id="@+id/timeDefault"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

    <TimePicker android:id="@+id/timePicker"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</LinearLayout>

```

Теперь добавим в метод `onCreate` текущей `activity` код, который установит значения по умолчанию для этих элементов:

```

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
}

```

```

        TextView dateDefault =
(TextView)findViewById(R.id.dateDefault);
        TextView timeDefault =
(TextView)findViewById(R.id.timeDefault);

```

```

        DatePicker dp = (DatePicker)this.findViewById(R.id.datePicker);
        // Месяц начиная с нуля. Для отображения добавляем 1.
        dateDefault.setText("Дата по умолчанию " + dp.getDayOfMonth()
+ "/" +
        (dp.getMonth() + 1) + "/" + dp.getYear());

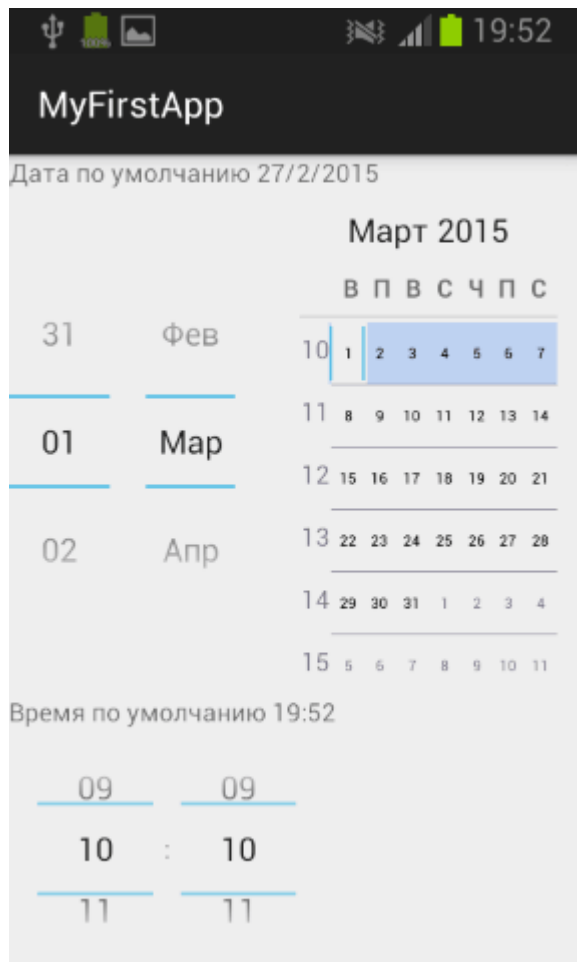
```

```

        dp.init(2015, 02, 01, null);
        TimePicker tp = (TimePicker)this.findViewById(R.id.timePicker);
        java.util.Formatter timeF = new java.util.Formatter();
        timeF.format("Время по умолчанию %d:%02d",
tp.getCurrentHour(),
        tp.getCurrentMinute());
        timeDefault.setText(timeF.toString());
        tp.setIs24HourView(true);
        tp.setCurrentHour(new Integer(10));
        tp.setCurrentMinute(new Integer(10));
    }

```

Используя метод `dp.init(2015, 02, 01, null);` устанавливаем дату по умолчанию - 1 марта, так как отсчет месяцев идет с нуля. А перед отображением времени, прибегаем к форматированию.



Ползунок SeekBar

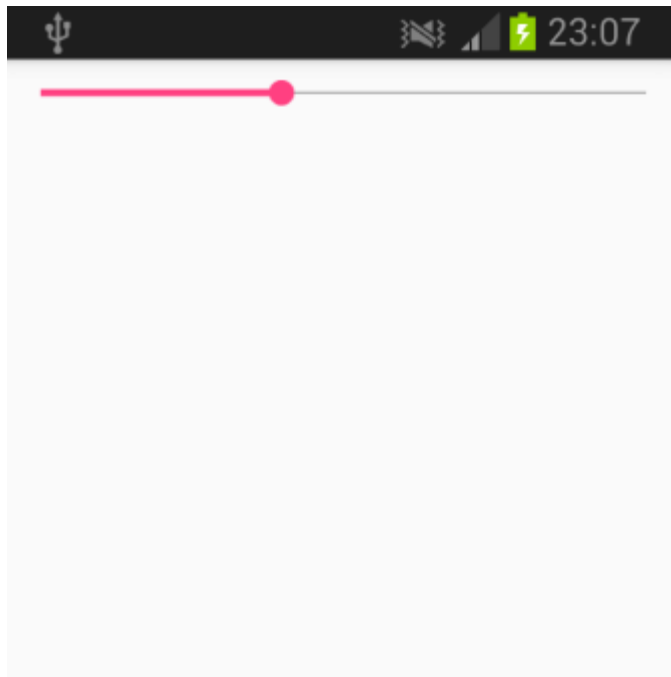
Элемент SeekBar выполняет роль ползунка, то есть шкалу делений, на которой мы можем менять текущую отметку. Определим SeekBar в разметке layout:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <SeekBar
        android:id="@+id/seekBar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:progress="20"
        android:max="50" />

</LinearLayout>
```

Атрибут android:progress позволяет задать текущее значение ползунка, а атрибут android:max - максимально возможное значение. В итоге мы получим следующий элемент:



Среди функционала элемента SeekBar следует отметить метод `setOnSeekBarChangeListener()`, который позволяет установить обработчики событий изменения значения ползунка. Воспользуемся этой функциональностью и определим в файле `layout` следующий код:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:id="@+id/txtView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textAlignment="center"
        android:layout_centerHorizontal="true"
        android:gravity="center_horizontal"
        android:textSize="30sp" />
    <SeekBar
        android:id="@+id/seekBar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:progress="20"
        android:max="50" />

</LinearLayout>
```

Здесь определен элемент `TextView`, который будет выводить текущее значение ползунка при его изменении.

И изменим код `MainActivity`:

```

import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.widget.SeekBar;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    TextView textView;
    SeekBar seekBar;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        seekBar = (SeekBar) findViewById(R.id.seekBar);
        textView = (TextView) findViewById(R.id.txtView);
        seekBar.setOnSeekBarChangeListener(new
SeekBar.OnSeekBarChangeListener() {
            @Override
            public void onProgressChanged(SeekBar seekBar, int progress,
boolean fromUser) {

                textView.setText(String.valueOf(progress));
            }

            @Override
            public void onStartTrackingTouch(SeekBar seekBar) {

            }

            @Override
            public void onStopTrackingTouch(SeekBar seekBar) {

            }
        });
    }
}

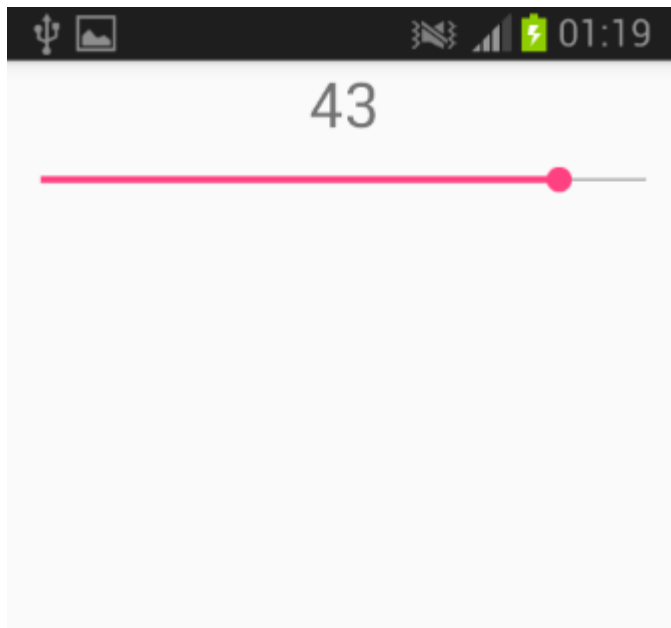
```

В метод `setOnSeekBarChangeListener()` передается объект `SeekBar.OnSeekBarChangeListener`, который позволяет установить три метода-обработчика:

`onProgressChanged`: срабатывает при перетаскивании ползунка по шкале. Передаваемый в метод параметр `progress` позволяет получить новое значение ползунка, которое в данном случае передается в `TextView` для отображения на экране

`onStartTrackingTouch`: срабатывает при начале перетаскивания ползунка по шкале

onStopTrackingTouch: срабатывает при завершении перетаскивания ползунка по шкале



Также мы можем получить текущее значение ползунка, используя метод `getProgress()`:

```
public void onProgressChanged(SeekBar seekBar, int progress, boolean
fromUser) {

    textView.setText(String.valueOf(seekBar.getProgress()));
}
```

ЛЕКЦИЯ №33-№34. РЕСУРСЫ В ANDROID-ПРИЛОЖЕНИЯХ.

Ресурс в приложении Android представляет собой файл, например, файл разметки интерфейса или некоторое значение, например, простую строку. То есть ресурсы представляют собой и файлы разметки, и отдельные строки, и звуковые файлы, файлы изображений и т.д. Все ресурсы находятся в проекте в каталоге `res`. Для различных типов ресурсов, определенных в проекте, в каталоге `res` создаются подкаталоги. Поддерживаемые подкаталоги:

- `animator/`: xml-файлы, определяющие анимацию свойств

- `anim/`: xml-файлы, определяющие tween-анимацию

- `color/`: xml-файлы, определяющие список цветов

- `drawable/`: Графические файлы (.png, .jpg, .gif)

- `mipmap/`: Графические файлы, используемые для иконок приложения под различные разрешения экранов

- `layout/`: xml-файлы, определяющие пользовательский интерфейс приложения

- `menu/`: xml-файлы, определяющие меню приложения

- `raw/`: различные файлы, которые сохраняются в исходном виде

- `values/`: xml-файлы, которые содержат различные используемые в приложении значения, например, ресурсы строк

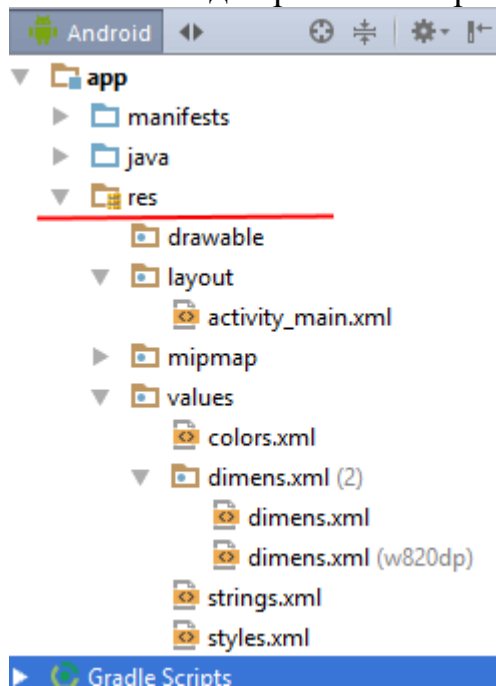
xml/: Произвольные xml-файлы

В общей сложности мы можем определить следующие типы ресурсов:

Ресурс	Каталог проекта	Файл	элемент в файле
Строки	/res/values/	strings.xml	<string>
Plurals	/res/values/	strings.xml	<plurals>
Массивы строк	/res/values/	strings.xml или arrays.xml	<string-array>
Логические значения Boolean	/res/values/	bools.xml	<bool>
Цвета	/res/values/	colors.xml	<color>
Список цветов	/res/color/	Произвольное название	<selector>
Размеры (Dimensions)	/res/values/	dimens.xml	<dimen>
Идентификаторы ID	/res/values/	ids.xml	<item>
Целые числа	/res/values/	integers.xml	<integer>
Массив целых чисел	/res/values/	integers.xml	<integer-array>
Графические файлы	/res/drawable/	Файлы с расширением jpg и png	-
Tween-анимация	/res/anim/	Файл xml с произвольным названием	<set>, <alpha>, <rotate>, <scale>, <translate>
Покадровая анимация	/res/drawable/	Файл xml с произвольным названием	<animation-list>
Анимация свойств	/res/animation/	Файл xml с произвольным названием	<set>, <objectAnimator>, <valueAnimator>
Меню	/res/menu/	Файл xml с произвольным названием	<menu>
XML-файлы	/res/xml/	Файл xml с произвольным названием	

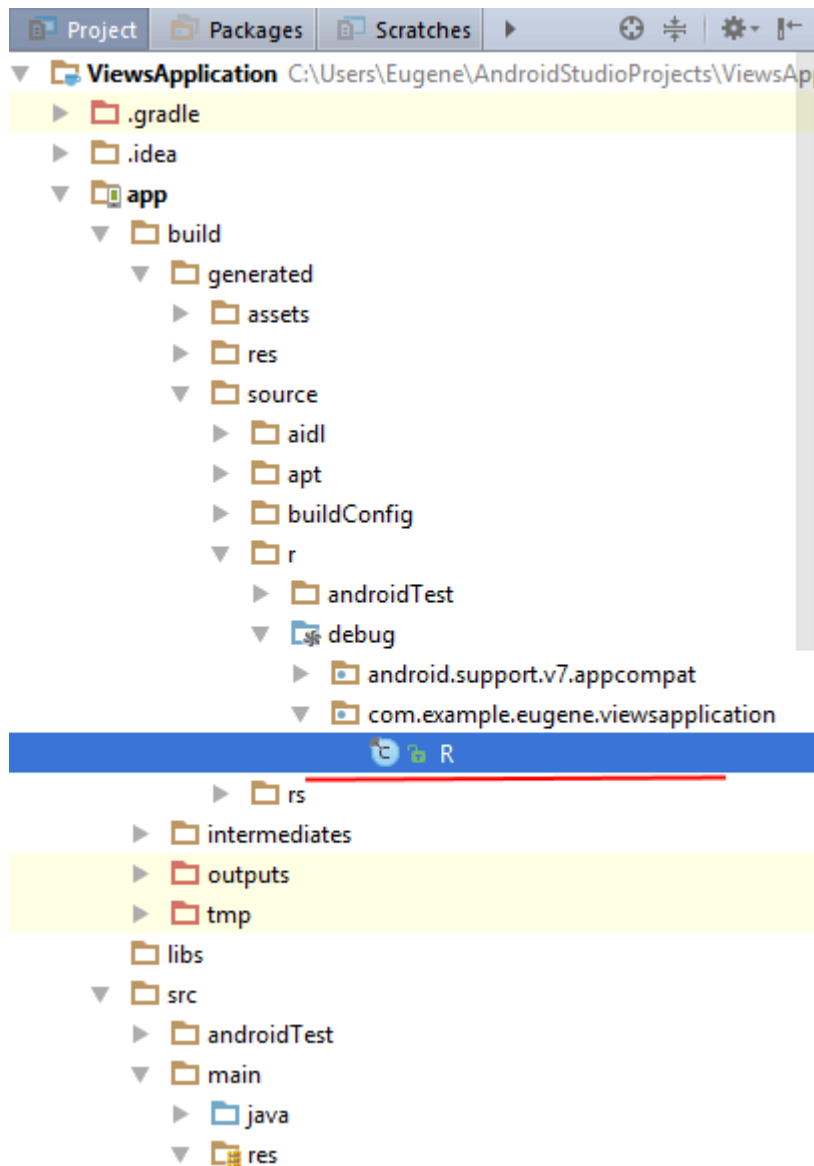
Бинарные и текстовые ресурсы	/res/raw/	Файлы мультимедиа (mp3, mp4), текстовые и другие файлы	
Разметка графического интерфейса	/res/layout/	Файл xml с произвольным названием	
Стили и темы	/res/values/	styles.xml, themes.xml	<style>

К примеру, если мы возьмем стандартный проект Android Studio, который создается по умолчанию, то там можем заметить наличие уже нескольких папок для различных ресурсов в каталоге res:



По умолчанию здесь есть каталоги не для всех типов ресурсов, которые использоваться в Android, однако при необходимости мы можем добавить в папку res нужный каталог, а в него затем поместить ресурс.

Когда происходит компиляция проекта сведения обо всех ресурсах добавляются в специальный файл R.java, который можно найти в проекте по пути app\build\generated\source\r\debug\[пакет_приложения]. Для его просмотра надо переключиться в полноценный вид проекта:



В этом файле будут храниться все определения ресурсов в виде числовых констант. К примеру, в проекте по умолчанию есть ресурс разметки интерфейса - файл `activity_main.xml` в папке `res/layout`. Для этого ресурса в классе `R` будет создаваться примерно следующая константа:

```
public final class R {
    .....
    public static final class layout {
        public static final int activity_main=0x7f030001;
    }
    .....
}
```

Применение ресурсов

Существует два способа доступа к ресурсам: в файле исходного кода и в файле `xml`.

Ссылка на ресурсы в коде

Тип ресурса в данной записи ссылается на одно из пространств (внутренних классов), определенных в файле `R.java`, которые имеют соответствующие им типы в `xml`:

`R.drawable` (ему соответствует тип в `xml` `drawable`)

R.id (id)
R.layout (layout)
R.string (string)
R.attr (attr)
R.plural (plurals)
R.array (string-array)

Например, для установки ресурса activity_main.xml в качестве графического интерфейса в коде MainActivity в методе onCreate() есть такая строка:

```
setContentView(R.layout.activity_main);
```

Через выражение R.layout.activity_main мы и ссылаемся на ресурс activity_main.xml, где layout - тип ресурса, а activity_main - имя ресурса.

Подобным образом мы можем получать другие ресурсы. Например, в файле res/values/strings.xml определен ресурс app_name:

```
<resources>  
    <string name="app_name">ViewsApplication</string>  
</resources>
```

Этот ресурс ссылается на строку. Чтобы получить ссылку на данный ресурс в коде java, мы можем использовать выражение R.string.app_name.

Доступ в файле xml

Нередко возникает необходимость сослаться на ресурс в файле xml, например, в файле, который определяет визуальный интерфейс, к примеру, в activity_main.xml. Ссылки на ресурсы в файлах xml имеют следующую формализованную форму: @[имя_пакета:]тип_ресурса/имя_ресурса

имя_пакета представляет имя пакета, в котором ресурс находится (указывать необязательно, если ресурс находится в том же пакете)

тип_ресурса представляет подкласс, определенный в классе R для типа ресурса

имя_ресурса имя файла ресурса без расширения или значение атрибута android:name в XML-элементе (для простых значений).

Например, мы хотим вывести в элемент TextView строку, которая определена в виде ресурса в файле strings.xml:

```
<TextView  
    android:id="@+id/welcome"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/app_name" />
```

В данном случае свойство text в качестве значения будет получать значение строкового ресурса app_name.

Метод getResources

Для получения ресурсов в классе Activity мы можем использовать метод getResources(), который возвращает объект android.content.res.Resources. Но чтобы получить сам ресурс, нам надо у полученного объекта Resources вызвать один из методов:

getString(): получает строку из файла strings.xml по числовому идентификатору

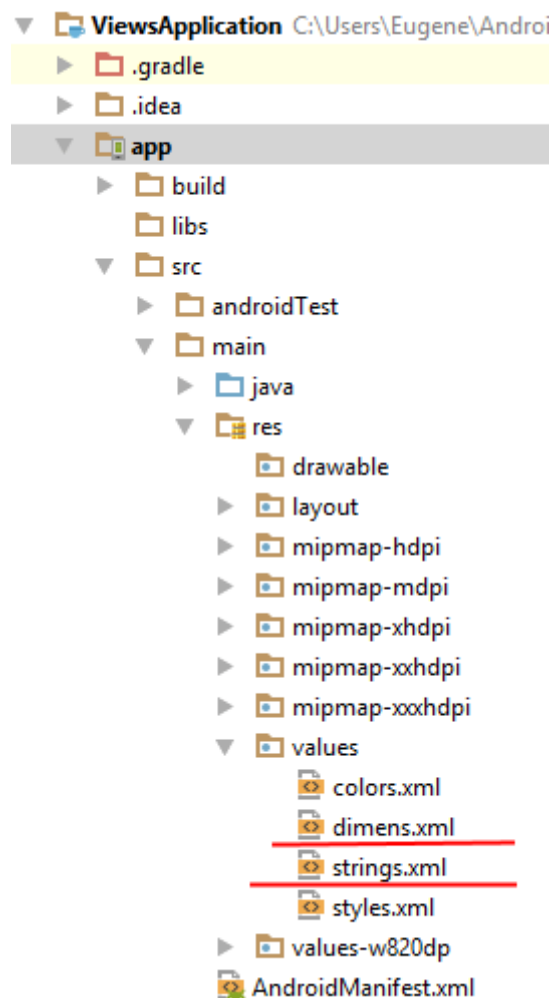
getDimension(): получает числовое значение - ресурс dimen

getDrawable(): получает графический файл

getBoolean(): получает значение boolean

Это только некоторые методы. Но вкратце рассмотрим их применение. Возьмем тот же файл res/values/strings.xml в качестве источника ресурсов. Пусть файл strings.xml выглядит так:

```
<resources>
    <string name="app_name">ViewsApplication</string>
</resources>
```



И изменим код MainActivity:

```
package com.example.eugene.viewsapplication;
```

```
import android.support.v7.app.AppCompatActivity;
```

```
import android.os.Bundle;
```

```
import android.widget.TextView;
```

```
public class MainActivity extends AppCompatActivity {
```

```
    protected void onCreate(Bundle savedInstanceState) {
```



```

super.onCreate(savedInstanceState);

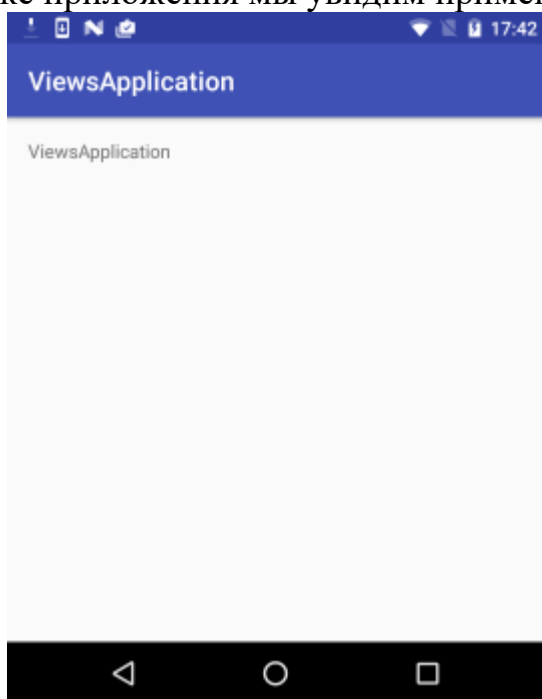
// получение ресурсов из файла values/strings.xml
String app_name = getResources().getString(R.string.app_name);

TextView textView = new TextView(this);
textView.setText(app_name);

setContentView(textView);
}
}

```

Здесь, используя метод `getResources()` получаем все ресурсы и затем используем их для установки значений свойств графических элементов. При запуске приложения мы увидим применение полученных ресурсов:



Ресурсы строк

Ресурсы строк - один из важных компонентов приложения. Мы используем их при выведении названия приложения, различного текста, например, текста кнопок и т.д.

XML-файлы, представляющие собой ресурсы строк, находятся в проекте в папке `res/values`. По умолчанию ресурсы строк находятся в файле `strings.xml`, который может выглядеть следующим образом:

```

<resources>
    <string name="app_name">ViewsApplication</string>
</resources>

```

В самом простом виде этот файл определяет один ресурс `"app_name"`, который устанавливает название приложения. Но естественно мы можем определить любые строковые ресурсы. Каждый отдельный ресурс определяется с помощью элемента `string`, а его атрибут `name` содержит название ресурса.

Для ресурсов строк в классе R определяется внутренний класс static final class string. Этот класс используется в качестве пространства для хранения идентификаторов ресурсов строк:

```
public static final class string {  
    public static final int app_name=0x7f040000;  
}
```

Константа app_name имеет тип не String, а int, а ее значение - числовой идентификатор ресурса.

Затем в приложении в файлах кода мы можем ссылаться на эти ресурсы:

```
R.string.app_name
```

А ОС Android сама сопоставит данные числовые идентификаторы с соответствующими ресурсами строк. Например:

```
String application_name = getResources().getString(R.string.app_name);
```

Либо в xml-файле:

```
@string/app_name
```

Например, изменим файл res/values/strings.xml следующим образом:

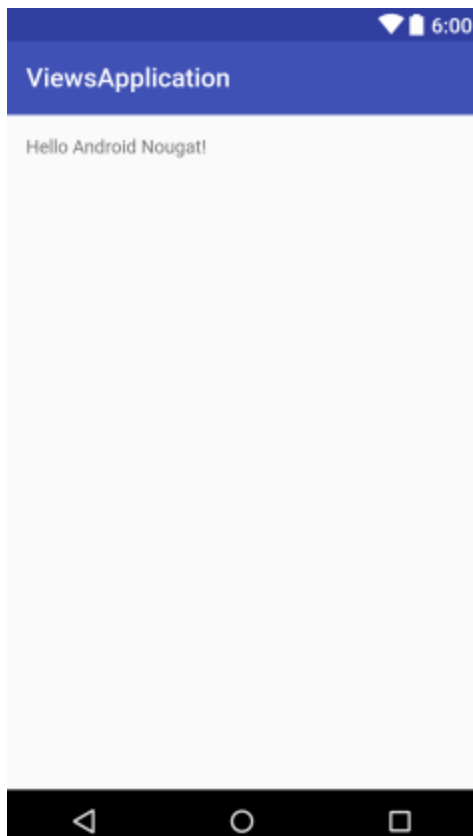
```
<resources>  
    <string name="app_name">ViewsApplication</string>  
    <string name="message">Hello Android Nougat!</string>  
</resources>
```

Здесь добавлен ресурс message со значением "Hello Android Nougat!".

Теперь используем ресурс в файле activity_main.xml:

```
<?xml version="1.0" encoding="utf-8"?>  
<RelativeLayout  
xmlns:android="http://schemas.android.com/apk/res/android"  
    android:id="@+id/activity_main"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:paddingBottom="@dimen/activity_vertical_margin"  
    android:paddingLeft="@dimen/activity_horizontal_margin"  
    android:paddingRight="@dimen/activity_horizontal_margin"  
    android:paddingTop="@dimen/activity_vertical_margin">  
  
    <TextView  
        android:id="@+id/welcome"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:text="@string/message" />  
</RelativeLayout>
```

С помощью выражения @string/message передаем атрибуту android:text значение из ресурса.



Аналогично мы могли бы использовать ресурс в коде Activity:

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

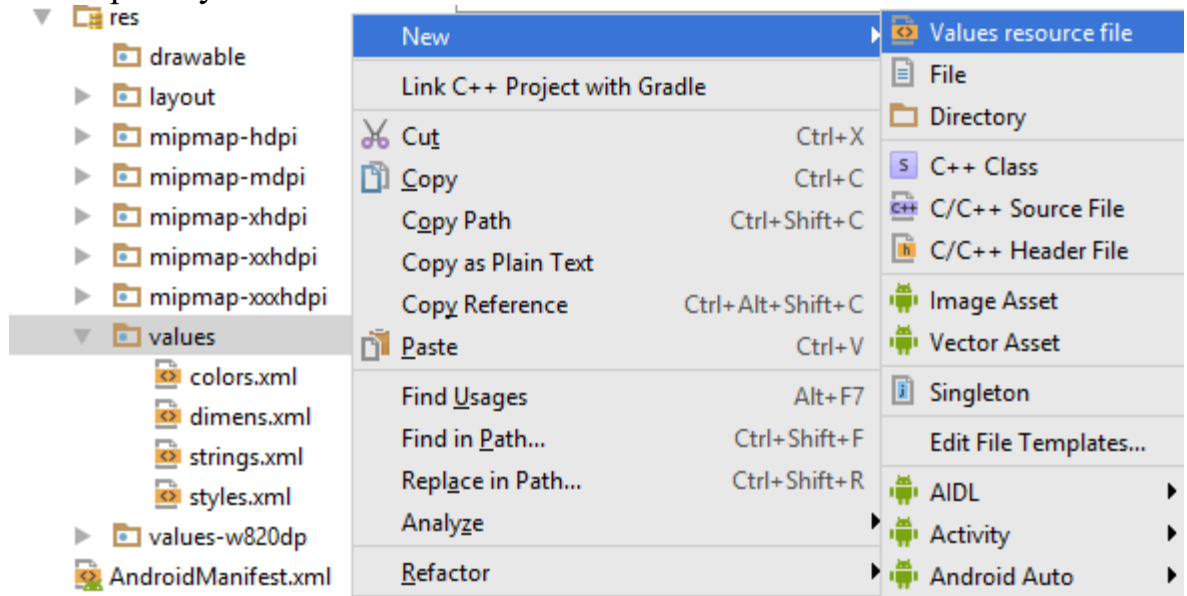
        // устанавливаем в качестве интерфейса файл second_layout.xml
        setContentView(R.layout.activity_main);

        // получаем элемент textView
        TextView textView = (TextView) findViewById(R.id.welcome);
        // получаем ресурс
        String message = getResources().getString(R.string.message);
        // переустанавливаем у него текст
        textView.setText(message);
    }
}
```

Хотя по умолчанию для ресурсов строк применяется файл strings.xml, но разработчики могут добавлять дополнительные файлы ресурсов в каталог проекта res/values. При этом достаточно соблюдать структуру файла: он

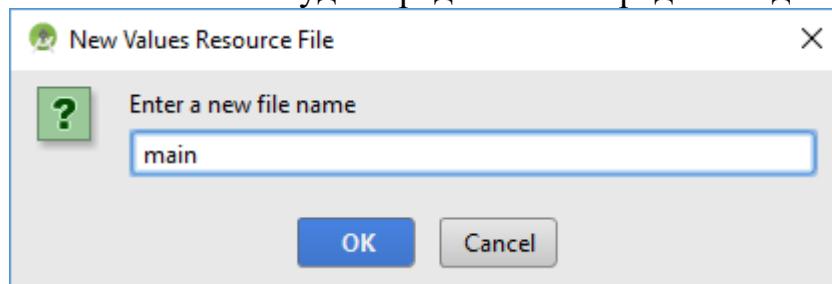
должен иметь корневой узел <resources> и иметь один или несколько элементов <string>.

Так, нажмем на папку res/values правой кнопкой мыши и в появившемся списке выберем пункт New -> Value Resource File:



Причем следует отметить, что данный тип файлов будет характерен для любого типа ресурсов, который добавляется в папку res/values.

После этого нам будет предложено определить для файла имя:



Назовем, к примеру, main, и в папку res/values будет добавлен новый файл main.xml. Определим в нем пару ресурсов:

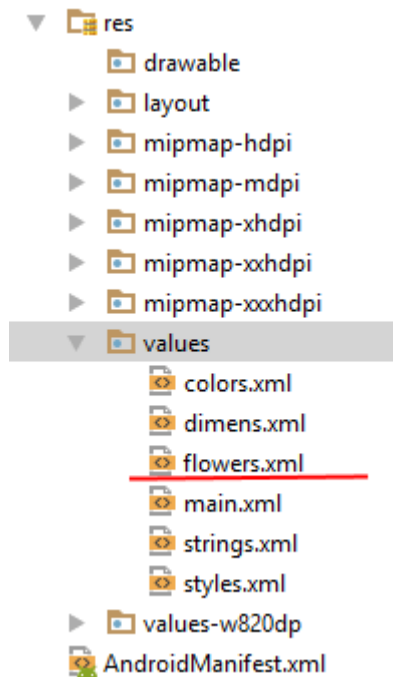
```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="welcome">Добро пожаловать</string>
    <string name="click_button">Нажмите на кнопку</string>
</resources>
```

И после этого мы сможем использовать определенные здесь ресурсы в коде Activity или в файле layout.

Ресурсы Plurals

Plurals представляют еще один вид набора строк. Он предназначен для описания количества элементов. Для чего это надо? К примеру, возьмем существительное: нередко оно изменяет окончание в зависимости от числительного, которое с ним употребляется: 1 цветок, 2 цветка, 5 цветков. Для подобных случаев и используется ресурс plurals.

Посмотрим на примере. Добавим в папку res/values новый ресурс. Назовем его flowers:



Изменим его содержимое следующим образом:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <plurals name="flowers">
        <item quantity="one">%d цветок</item>
        <item quantity="few">%d цветка</item>
        <item quantity="many">%d цветков</item>
    </plurals>
</resources>
```

Для задания ресурса используется элемент `<plurals>`, для которого существует атрибут `name`, получающий в качестве значения произвольное название, по которому потом ссылаются на данный ресурс.

Сами наборы строк вводятся дочерними элементами `<item>`. Этот элемент имеет атрибут `quantity`, который имеет значение, указывающее, когда эта строка используется. Данный атрибут может принимать следующие значения:

- zero: строка для количества в размере 0
- one: строка для количества в размере 1 (для русского языка - для задания всех количеств, оканчивающихся на 1, кроме 11)
- two: строка для количества в размере 2
- few: строка для небольшого количества
- many: строка для больших количеств
- other: все остальные случаи

Причем в данном случае многое зависит от конкретного языка. А система сама позволяет определить, какое значение брать для того или иного числа.

Использование данного ресурса возможно только в коде `java`. Поэтому изменим код `MainActivity`:

```
package com.example.eugene.viewsapplication;
```

```

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

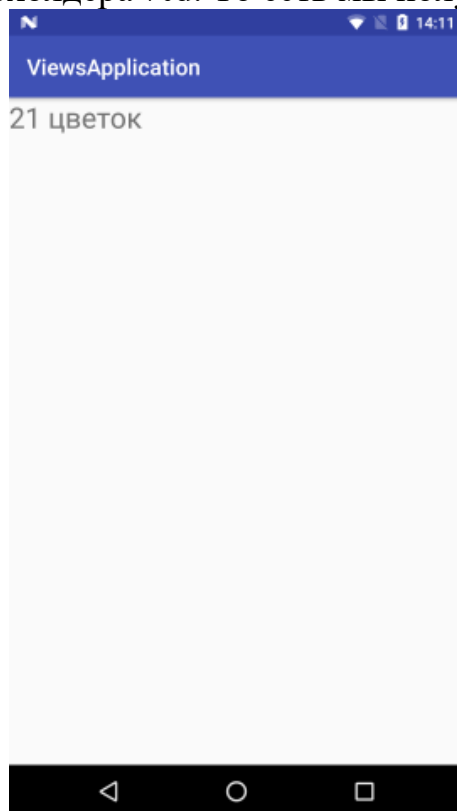
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        String rose = getResources().getQuantityString(R.plurals.flowers,
21, 21);

        TextView textView = new TextView(this);
        textView.setText(rose);
        textView.setTextSize(26);
        setContentView(textView);
    }
}

```

С помощью метода `getQuantityString` мы получаем значение ресурса. Первым параметром передаем идентификатор ресурса. Вторым параметром идет значение, для которого нужно найти нужную строку. Третий параметр представляет собой значение, которое будет вставляться на место плейсхолдера `%d`. То есть мы получаем строку для числа 21.



Ресурсы dimension

Определение размеров должно находиться в папке res/values в файле с любым произвольным именем. Общий синтаксис определения ресурса следующий:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="имя_ресурса">используемый_размер</dimen>
</resources>
```

Как и другие ресурсы, ресурс dimension определяется в корневом элементе <resources>. Тег <dimen> обозначает ресурс и в качестве значения принимает некоторое значение размера в одной из принятых единиц измерения (dp, sp, pt, px, mm, in).

Так, добавим в Android Studio в папку res/values новый файл, который назовем dimens.xml. Определим в нем следующее содержимое:

```
<resources>
    <dimen name="activity_horizontal_margin">16dp</dimen>
    <dimen name="activity_vertical_margin">16dp</dimen>
    <dimen name="text_size">16sp</dimen>
</resources>
```

Здесь определены два ресурса для отступов activity_horizontal_margin и activity_vertical_margin, которые хранят значение 16 dp, и ресурс text_size, который хранит высоту шрифта - 16sp. Названия ресурсов могут быть произвольными.

Используем ресурс в файле activity_main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin">

    <TextView
        android:textSize="@dimen/text_size"
        android:layout_width="wrap_content"
        android:text="Hello Android Nougat!" />
</RelativeLayout>
```

Ресурсы dimension используются для таких атрибутов визуальных элементов, которые в качестве значения требуют указание числового значения. Например, атрибут android:layout_height или android:textSize. Для получения ресурса в xml после "@dimen/" указывается имя ресурса.

Для получения ресурсов в коде java применяется метод getDimension() класса Resources:

```

package com.example.eugene.viewsapplication;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

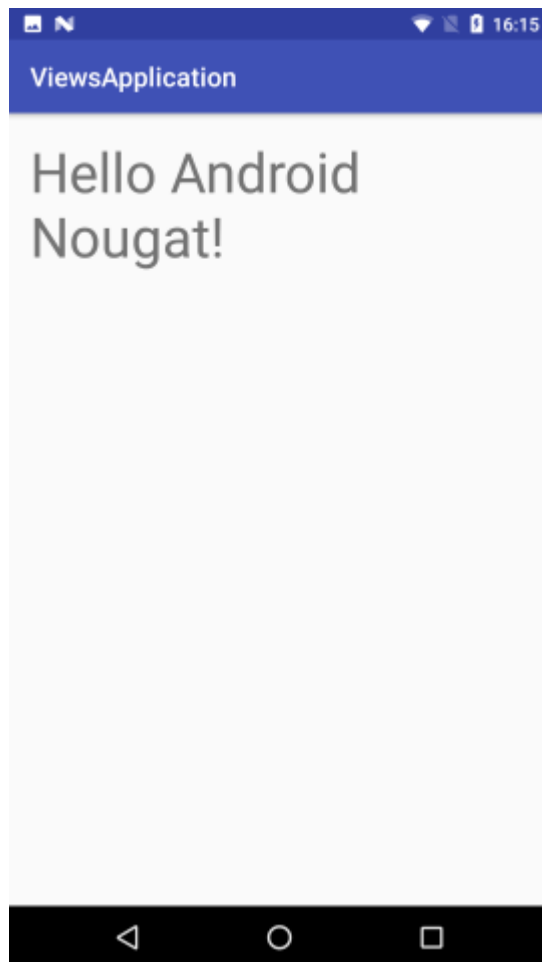
        float textSize = getResources().getDimension(R.dimen.text_size);
        int leftPadding =
(int)getResources().getDimension(R.dimen.activity_horizontal_margin);
        int topPadding =
(int)getResources().getDimension(R.dimen.activity_vertical_margin);

        TextView textView = new TextView(this);

        textView.setText("Hello Android Nougat!");
        textView.setTextSize(textSize);
        textView.setPadding(leftPadding, topPadding, leftPadding,
topPadding);

        setContentView(textView);
    }
}

```

Перевод из dip в пиксели

В случае выше мы использовали преимущественно единицы dip. Но что, если мы хотим динамически перейти от одних единиц измерения к другим, например, от dip к пикселям. Для этого мы можем применить класс `android.util.TypedValue`:

```
package com.example.eugene.viewsapplication;

package com.example.eugene.layoutapp;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.LinearLayout;
import android.widget.TextView;
import android.util.TypedValue;

public class MainActivity extends AppCompatActivity {

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // условное значение в dp
        int textViewHeight = 100;
```

```

LinearLayout linearLayout = new LinearLayout(this);
linearLayout.setOrientation(LinearLayout.VERTICAL);

TextView textView1 = new TextView(this);
textView1.setText("Hello Android Nougat!");
textView1.setTextSize(26);
textView1.setBackgroundColor(0xffc5cae9);

// преобразуем размер из dp в физические пиксели
int height =
(int)TypedValue.applyDimension(TypedValue.COMPLEX_UNIT_DIP,
    textViewHeight, getResources().getDisplayMetrics());
textView1.setHeight(height);

linearLayout.addView(textView1);

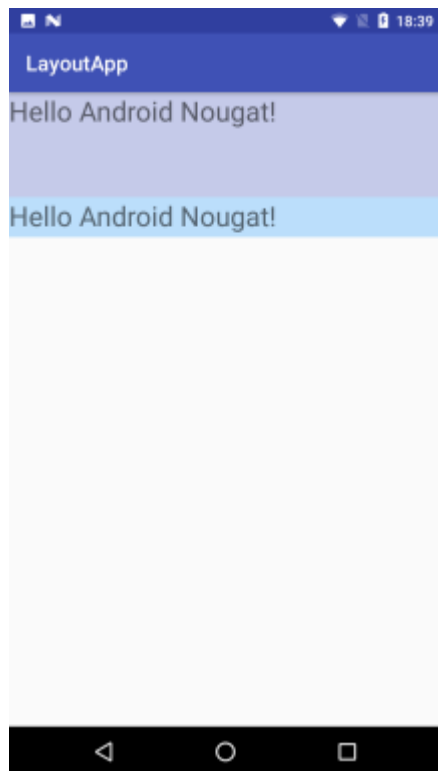
TextView textView2 = new TextView(this);
textView2.setText("Hello Android Nougat!");
textView2.setTextSize(26);
textView2.setBackgroundColor(0xffbbdefb);
textView2.setHeight(textViewHeight);
linearLayout.addView(textView2);

setContentView(linearLayout);
}
}

```

С помощью метода `TypedValue.applyDimension()` осуществляется перевод из одних единиц в другие. Первый параметр - тип исходных единиц измерения (в данном случае `dp`), второй параметр - количество для конвертации, а третий параметр - параметр `DisplayMetrics`, который позволяет произвести конвертацию.

В данном случае создается 2 элемента `TextView`. Для установки высоты для обоих элементов используется значение из переменной `textViewHeight`, однако в первом случае это значение подвергается преобразованию из `dp` в пиксели, и соответственно оба элемента будут иметь различную итоговую высоту:



Ресурсы Color и установка цвета

В приложении Android также можно определять ресурсы цветов (Color). Они должны храниться в файле по пути `res/values` и также, как и ресурсы строк, заключены в тег `<resources>`. Так, по умолчанию при создании самого простого проекта в папку `res/values` добавляется файл `colors.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#3F51B5</color>
    <color name="colorPrimaryDark">#303F9F</color>
    <color name="colorAccent">#FF4081</color>
</resources>
```

Цвет определяется с помощью элемента `<color>`. Его атрибут `name` устанавливает название цвета, которое будет использоваться в приложении, а шестнадцатеричное число - значение цвета.

Для задания цветовых ресурсов можно использовать следующие форматы:

- #RGB (#F00 - 12-битное значение)
- #ARGB (#8F00 - 12-битное значение с добавлением альфа-канала)
- #RRGGBB (#FF00FF - 24-битное значение)
- #AARRGGBB (#80FF00FF - 24-битное значение с добавлением альфа-канала)

Чтобы проще было ориентироваться в цветовых схемах и в том, какие цвета предпочтительнее использовать, на странице <https://material.io/guidelines/style/color.html#> можно найти большой список из цветов, рекомендованных компанией Google.

Изменим файл `colors.xml`, добавив еще один цвет:

```

<?xml version="1.0" encoding="utf-8"?>
<resources>
  <color name="colorPrimary">#3F51B5</color>
  <color name="colorPrimaryDark">#303F9F</color>
  <color name="colorAccent">#FF4081</color>
  <color name="textViewController">#EEEEEE</color>
</resources>

```

Применим цвета в файле activity_main.xml:

```

<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/activity_main"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingBottom="@dimen/activity_vertical_margin"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin">

  <TextView
    android:textSize="20dp"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"

    android:textColor="@color/colorPrimary"
    android:background="@color/textViewController"

    android:text="Hello Android Nougat!" />

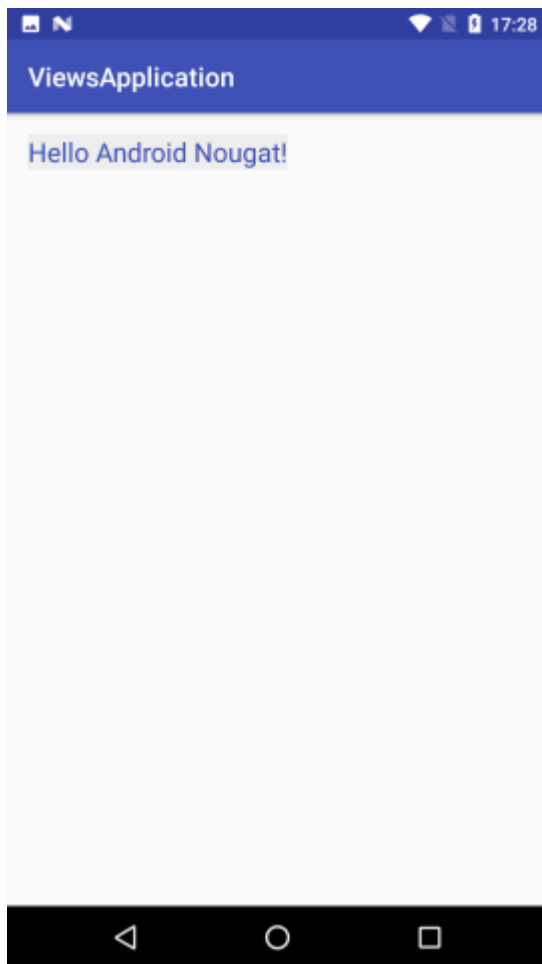
```

```

</RelativeLayout>

```

С помощью атрибута android:textColor устанавливается цвет текста в TextView, а атрибут android:background устанавливает фон TextView. В качестве значения они используют цвет, например, в том же шестнадцатеричном формате. Для получения самого цвета после "@color/" указывается имя ресурса.



Также можно использовать цветовые ресурсы в коде MainActivity:

```
package com.example.eugene.viewsapplication;
```

```
import android.support.v4.content.ContextCompat;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.TextView;
```

```
public class MainActivity extends AppCompatActivity {
```

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        int textColor = ContextCompat.getColor(this,
R.color.colorPrimary);
        int backgroundColor = ContextCompat.getColor(this,
R.color.textViewColor);
```

```
        TextView textView = new TextView(this);
        textView.setText("Hello Android Nougat!");
        textView.setTextSize(20);
        textView.setPadding(16, 16, 16, 16);
```

```

        textView.setTextColor(textColor);
        textView.setBackgroundColor(backgroundColor);

        setContentView(textView);
    }
}

```

Для получения цвета применяет метод `ContextCompat.getColor()`, который в качестве первого параметра принимает текущий объект `Activity`, а второй параметр - идентификатор цветового ресурса.

ЛЕКЦИЯ №35-№36. АКТИВНОСТИ И ИНТЕНТЫ.

Что такое Intent в Android?

Android Intent (возможен еще дословный вариант перевода, как «Намерение», однако я предпочитаю использовать слово в оригинале Intent или читать как «интент»). И так, Intent представляет собой структуру данных, содержащую описание операции, которую следует выполнить.

Одна из наиболее мощных функций Intent механизма в том, что вы можете асинхронно отправить сообщения другим Activity (активностям) или сервисам.

В этом уроке мы сделаем акцент на типах и обработке Android Intent (намерений).

Типы Android Intent

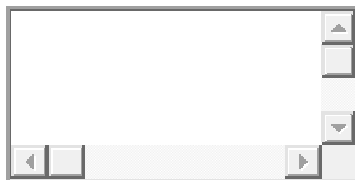
Явное намерение (Explicit intent) явно определяет компонент по имени. Чаще всего в наших приложениях мы будем использовать явное намерение для запуска какого-то компонента. Суть Explicit intent в том, что вы знаете имя класса Activity или Service (сервиса), который нужно запустить. При использовании явного намерения система немедленно запускает компонент приложения, указанный в объекте Intent.

Неявное намерения (Implicit intent) не определяет конкретный компонент, а просто заявляет о некотором действии, позволяя компонентам других приложений его обработать. При создании неявного намерения Android система находит все возможные компоненты, которые могут обрабатывать заявленное действие, и предлагает выбрать одно из найденного списка.

Как использовать Android Intent для запуска Activity?

Чтобы создать объект Intent и запустить с помощью него Activity, нужно в параметрах передать объект-Activity класса, который вызывает и имя Activity, которое нужно запустить:

Java



```
1      Intent i = new Intent(FromActivity.this, ToActivity.class);  
2      startActivity(i);
```

Запуск Activity под именем ToActivity.class будет происходить в таком порядке:

- Создание нового экземпляра класса ToActivity.class

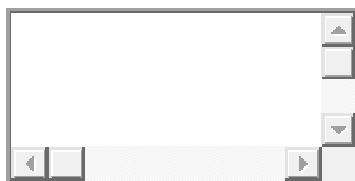
- Этот экземпляр помещается на вершину стека текущих операций

- Активности ToActivity.class появляется на экране.

Используем флаги в намерениях (Android Intent Flags)

Определенные в объекте Intent флаги играют роль метаданных. Они используются Android системой в качестве руководства по запуску и обработке этих самых намерений. Изменить поведение объектов Intent можно с помощью указания флагов (методом `addFlag()`) перед вызовом метода `startActivity(Intent)`, например:

Java



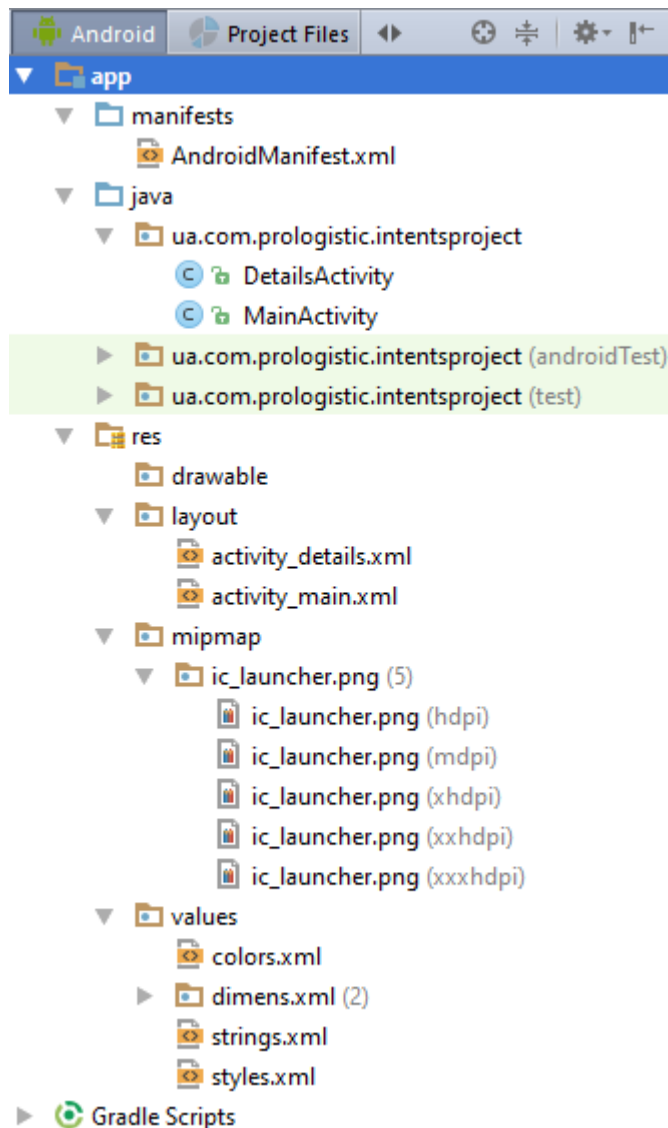
```
intent.addFlag(Intent.FLAG_ACTIVITY_SINGLE_TOP);
```

Это значит, что переданная в объект Intent активность не будет запущена, если в стеке работающих тасков уже есть такая Activity.

Работа с Intent на примере запуска одной активности из другой

Давайте создадим новый проект для работы с Intent. В данном случае нас устроит простой проект Hello Android, описанный в этом уроке. Работать мы будем в Android Studio. Если вы не знакомы с ней, то используйте любую удобную вам IDE.

После создания ваш проект должен примерно выглядеть так:

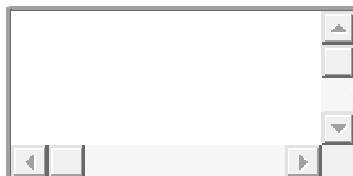


Этот проект состоит из двух Activity: MainActivity.java и DetailsActivity.java. Мы будем запускать активность DetailsActivity из MainActivity.

Для запуска активности нам нужна кнопка, по нажатию которой будет происходить создание намерения и запуск Activity. Если вы не знакомы с виджетом Button (кнопкой), пройдите на другой мой урок по [этой ссылке](#).

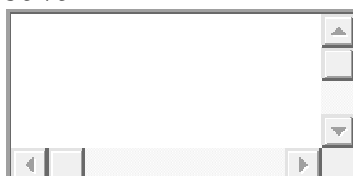
Код MainActivity будет такой:

Java



акет activity_main.xml будет таким:

Java




```

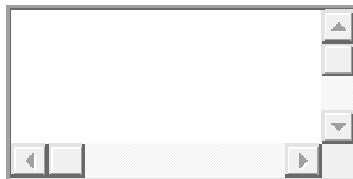
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="ua.com.prologistic.intentsproject.MainActivity">

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Запустить Activity"
        android:id="@+id/button"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true" />
</RelativeLayout>

```

Код DetailsActivity будет такой:

Java



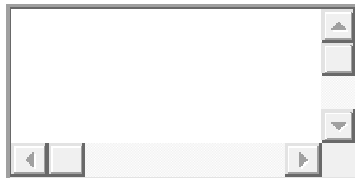
```
package ua.com.prologistic.intentsproject;
```

```
import android.os.Bundle;
import android.support.annotation.Nullable;
import android.support.v7.app.AppCompatActivity;
```

```
public class DetailsActivity extends AppCompatActivity {
    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_details);
    }
}
```

Макет activity_details.xml следующий:

Java

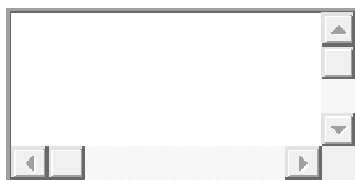


```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="ua.com.prologistic.intentsproject.DetailsActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Это вторая Activity"
        android:id="@+id/textView"
        android:layout_gravity="center_horizontal"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true" />
</RelativeLayout>
```

Файл конфигураций AndroidManifest.xml будет иметь такой вид:

Java



```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
    package="ua.com.prologistic.intentsproject">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
```

```

        <action android:name="android.intent.action.MAIN" />

        <category
android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <activity android:name=".DetailsActivity"/>
</application>

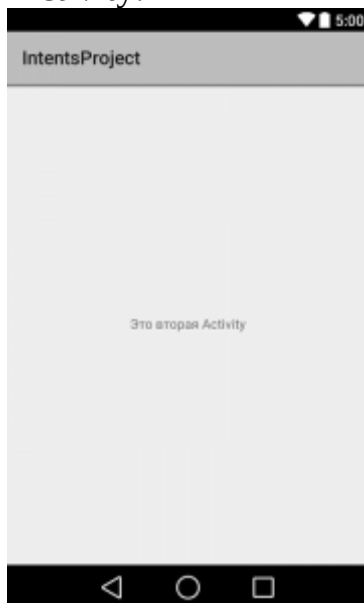
</manifest>

```

Создайте эти файлы и запустите приложение. При запуске оно будет выглядеть так:



После нажатия кнопки «Запустить Activity» у вас откроется новая Activity:



Вот и вся работа с Intent на Android. В этом уроке мы узнали, что такое Intent, зачем нужны намерения и как их использовать в своих приложениях.

ЛЕКЦИЯ №37-№38. ОБРАБОТКА ПОЛЬЗОВАТЕЛЬСКОГО ВВОДА. КАСАНИЯ, ВВОД ТЕКСТА.

В системе Android предусмотрено несколько способов перехвата событий взаимодействия пользователя с вашим приложением. Что касается событий в пользовательском интерфейсе, подход состоит в том, чтобы захватывать события из определенного представления объекта, с которым взаимодействует пользователь. Класс отображаемых объектов (View) содержит средства, позволяющие делать это.

В различных классах View, которые вы будете использовать для создания макета, вы заметите несколько общедоступных методов обратного вызова, полезных для работы с событиями пользовательского интерфейса. Эти методы вызываются платформой Android, когда с этим объектом выполняется соответствующее действие. Например, когда пользователь касается отображаемого объекта (кнопки), вызывается метод `onTouchEvent()` этого объекта. Однако для перехвата этого события вы должны наследовать класс и переопределить метод. Однако наследование каждого отображаемого объекта для обработки такого события было бы неудобно. Именно поэтому класс View также содержит набор вложенных интерфейсов с обратными вызовами, которые можно задать значительно проще. Эти интерфейсы, которые называются приемниками событий, и служат перехватчиками действий пользователя с вашим пользовательским интерфейсом.

Несмотря на то, что вы будете чаще использовать приемники событий для перехвата действий пользователя, может наступить момент, когда вам не захочется наследовать класс View, чтобы создать нестандартный компонент. Возможно, вы захотите наследовать класс Button, чтобы сделать нечто более необычное. В этом случае вы сможете определить поведение события по умолчанию для своего класса с помощью обработчиков событий класса.

Приемники событий

Приемник событий — это интерфейс в классе View, который содержит один метод обратного вызова. Эти методы будут вызываться платформой Android, когда в результате взаимодействия пользователя с объектом пользовательского интерфейса срабатывает отображаемый объект View, в котором зарегистрирован приемник.

Интерфейсы, включенные в приемник события, представляют собой следующие методы обратного вызова:

`onClick()`

Из объекта `View.OnClickListener`. Этот метод вызывается, когда пользователь касается элемента (в режиме касания), или переводит фокус на элемент с помощью клавиш перемещения или трекбола и нажимает соответствующую клавишу «ввода» или нажимает на трекбол.

`onLongClick()`

Из объекта `View.OnLongClickListener`. Этот метод вызывается, когда пользователь касается элемента и удерживает его (в режиме касания), или переводит фокус на элемент с помощью клавиш перемещения или трекбола и

нажимает и удерживает соответствующую клавишу «ввода» или трекбол (в течение одной секунды).

`onFocusChange()`

Из объекта `View.OnFocusChangeListener`. Этот метод вызывается, когда пользователь перемещается в элемент или из него с помощью клавиш перемещения или трекбола.

`onKey()`

Из объекта `View.OnKeyListener`. Этот метод вызывается, когда пользователь переносит фокус на элемент и нажимает или отпускает аппаратную клавишу на устройстве.

`onTouch()`

Из объекта `View.OnTouchListener`. Этот метод вызывается, когда пользователь выполняет действие, считающееся событием касания, например, нажимает, отпускает или выполняет любой жест на экране (в пределах границ элемента).

`onCreateContextMenu()`

Из объекта `View.OnCreateContextMenuListener`. Этот метод вызывается, когда создается контекстное меню (в результате длительного «длительного нажатия»). См. обсуждение контекстных меню в руководстве для разработчиков Меню.

Эти методы являются единственными составными частями соответствующих интерфейсов. Чтобы определить один из этих методов и обрабатывать события, реализуйте вложенный интерфейс в вашем процесс или определите его, как анонимный класс. Затем передайте экземпляр реализации в соответствующий метод `View.set...Listener()`. (Например, вызовите `setOnClickListener()` и передайте ему свою реализацию `OnClickListener`.)

В следующем примере показано, как зарегистрировать приемник события «по клику» (on-click) для кнопки.

```
// Create an anonymous implementation of OnClickListener
private OnClickListener mCorkyListener = new OnClickListener() {
    public void onClick(View v) {
        // do something when the button is clicked
    }
};

protected void onCreate(Bundle savedInstanceState) {
    ...
    // Capture our button from layout
    Button button = (Button)findViewById(R.id.corky);
    // Register the onClick listener with the implementation above
    button.setOnClickListener(mCorkyListener);
    ...
}
```

Возможно, еще удобнее реализовать `OnClickListener` как часть вашей операции. При этом исключается дополнительная нагрузка класса и выделение объекта. Например:

```
public class ExampleActivity extends Activity implements OnClickListener
{
    protected void onCreate(Bundle savedInstanceState) {
        ...
        Button button = (Button)findViewById(R.id.corky);
        button.setOnClickListener(this);
    }

    // Implement the OnClickListener callback
    public void onClick(View v) {
        // do something when the button is clicked
    }
    ...
}
```

Обратите внимание, что обратный вызов `onClick()` в примере выше не содержит возвращаемого значения, но некоторые другие методы приемника события должны возвращать логическое значение. Причина зависит от события. Для некоторых методов, которые возвращают значения, причина описана ниже:

`onLongClick()` — этот метод возвращает логическое значение, указывающее, что вы обработали это событие и его более не следует хранить. А именно, верните значение `true`, чтобы указать, что вы обработали событие и его следует остановить; верните значение `false`, если вы не обработали его и/или событие должно продолжаться для любых других приемников события `on-click`.

`onKey()` — этот метод возвращает логическое значение, указывающее, что вы обработали это событие и его более не следует хранить. А именно, верните значение `true`, чтобы указать, что вы обработали событие и его следует остановить; верните значение `false`, если вы не обработали его и/или событие должно продолжаться для любых других приемников события `on-click`.

`onTouch()` — этот метод возвращает логическое значение, указывающее, обработал ли ваш приемник это событие. Важно, что это событие может повлечь несколько действий, следующих друг за другом. Поэтому, если вы возвращаете ложь при приеме события нажатия, вы указываете, что вы не обработали событие и не интересуетесь последующими действиями в результате этого события. Соответственно, этот метод не будет вызываться для любых других действий в рамках этого события, таких как жесты или возможное действие отпускания.

Помните, что события аппаратных клавиш всегда попадают в отображаемый объект `View`, который находится в фокусе. Они отправляются, начиная с верха в иерархии отображаемых объектов, затем ниже, пока не

достигнут соответствующего назначения. Если ваш отображаемый объект (или дочерний объект вашего отображаемого объекта) находится в фокусе, вы можете видеть, как событие перемещается сквозь метод `dispatchKeyEvent()`. В качестве альтернативы к перехвату событий клавиш через отображаемый объект View можно также получать все события внутри вашей операции с помощью методов `onKeyDown()` и `onKeyUp()`.

Кроме того, размышляя о вводе текста для приложения, помните, что многие устройства поддерживают только программные методы ввода. Такие методы не обязательно основаны на нажатиях клавиш. Некоторые могут использовать голосовой ввод, рукописный ввод и т. д. Даже если метод ввода представлен интерфейсом, подобным клавиатуре, он обычно не приводит к запуску семейства событий `onKeyDown()`. Не следует создавать пользовательский интерфейс, для управления которым требуется нажимать определенные клавиши, кроме случаев, когда вы хотите ограничить использование приложения только устройствами с аппаратной клавиатурой. В частности, не полагайтесь на эти методы для подтверждения ввода, когда пользователь нажимает клавишу «Ввод». Вместо этого используйте такие действия, как `IME_ACTION_DONE`, чтобы подать методу ввода сигнал о том, какой реакции ожидает приложение, чтобы он мог соответственно изменить свой пользовательский интерфейс. Избегайте предположений о работе программного метода ввода и доверьте ему ввод уже отформатированного текста в ваше приложение.

Примечание. Система Android сначала будет вызывать обработчики событий, а затем соответствующие обработчики по умолчанию из определения класса. Поэтому возврат значения истина из этих приемников событий будет останавливать передачу события остальным приемникам событий, а также будет блокировать обратный вызов обработчика событий по умолчанию в отображаемом объекте. Поэтому проверьте, что вы хотите прервать событие, когда вы возвращаете значение true.

Обработчики событий

Если вы создаете нестандартный компонент из отображаемого объекта, вы сможете определить несколько методов обратного вызова, используемых как обработчики события по умолчанию. В документе Нестандартные компоненты можно узнать о некоторых общих обратных вызовах, используемых для обработки событий, включая следующие:

`onKeyDown(int, KeyEvent)` — вызывается при возникновении нового события клавиши.

`onKeyUp(int, KeyEvent)` — вызывается при возникновении события отпущения клавиши.

`onTrackballEvent(MotionEvent)` — вызывается при возникновении события перемещения трекбола.

`onTouchEvent(MotionEvent)` — вызывается при возникновении события жеста на сенсорном экране.

`onFocusChanged(boolean, int, Rect)` — вызывается, когда отображаемый объект получает или теряет фокус.

Существует несколько других методов, о которых следует знать и которые не являются частью класса `View`, но могут напрямую влиять на доступные вам способы обработки событий. Поэтому, управляя более сложными событиями внутри макета, учитывайте и другие методы:

`Activity.dispatchTouchEvent(MotionEvent)` — этот метод позволяет вашей операции `Activity` перехватывать все события касаний перед их отправкой в окно.

`ViewGroup.onInterceptTouchEvent(MotionEvent)` — этот метод позволяет объекту `ViewGroup` просматривать события перед их отправкой в дочерние отображаемые объекты.

`ViewParent.requestDisallowInterceptTouchEvent(boolean)` — вызовите этот метод в родительском отображаемом объекте, чтобы указать ему, что он не должен перехватывать события касания с помощью `onInterceptTouchEvent(MotionEvent)`.

Режим касания

Когда пользователь перемещается по пользовательскому интерфейсу с помощью клавиш перемещения или трекбола, необходимо передавать фокус действующим элементам (таким как кнопки), чтобы пользователь мог видеть, какой элемент будет принимать ввод. Однако, если устройство поддерживает сенсорный ввод, и пользователь начинает взаимодействовать с интерфейсом, прикасаясь к его элементам, исчезает необходимость выделять элементы или передавать фокус определенному отображаемому объекту. Следовательно, существует режим взаимодействия, который называется «режимом касания».

Как только пользователь касается экрана, устройство, поддерживающее сенсорный ввод, переходит в режим касания. Начиная с этого момента, фокус передается только тем отображаемым объектам, для которых `isFocusableInTouchMode()` имеет значение `true`, таким как виджеты редактирования текста. Другие отображаемые объекты, которых можно коснуться, например, кнопки, не будут получать фокус при касании. Нажатие будет просто запускать их приемники событий `on-click`.

В любой момент, когда пользователь нажимает клавишу перемещения или выполняет прокрутку трекболом, устройство выходит из режима касания и находит отображаемый объект, которому передается фокус. Теперь пользователь может возобновить взаимодействие с пользовательским интерфейсом без касания экрана.

Состояние режима касания поддерживается во всей системе (для всех окон и операций). Чтобы узнать текущее состояние, можно вызвать `isInTouchMode()` и посмотреть, находится ли устройство в режиме касания.

Фокус обработки

В ответ на пользовательский ввод система обрабатывает обычное перемещение фокуса. Сюда относится изменение фокуса, когда отображаемые объекты удаляются или скрываются, а также когда становятся доступными новые отображаемые объекты. Отображаемые объекты сообщают о своей готовности получить фокус с помощью метода `isFocusable()`. Чтобы изменить

способность объекта View получать фокус, вызовите `setFocusable()`. В режиме касания можно узнать способность отображаемого объекта View получать фокус с помощью метода `isFocusableInTouchMode()`. Для изменения вызовите `setFocusableInTouchMode()`.

Перемещение фокуса основано на алгоритме, который находит ближайший элемент в заданном направлении. В редких случаях алгоритм по умолчанию может не совпадать с поведением, которое предполагает разработчик. В этих ситуациях можно задать явное переопределение с помощью следующих атрибутов XML в файле макета: `nextFocusDown`, `nextFocusLeft`, `nextFocusRight` и `nextFocusUp`.

Добавьте один из этих атрибутов в отображаемый объект View, из которого выходит фокус. Задайте значение атрибута для идентификатора объекта View, в который следует передать фокус. Например:

```
<LinearLayout
    android:orientation="vertical"
    ...
    <Button
        android:id="@+id/top"
        android:nextFocusUp="@+id/bottom"
        ...
    <Button
        android:id="@+id/bottom"
        android:nextFocusDown="@+id/top"
        ...
</LinearLayout>
```

Обычно в этом вертикальном макете перемещение вверх из первой кнопки не должно приводить к перемещению, как и перемещение вниз из второй кнопки. Теперь, когда верхняя кнопка задает для нижней кнопки атрибут `nextFocusUp` (и наоборот), фокус будет перемещаться циклически сверху вниз и снизу вверх.

Если вы хотите объявить, что отображаемый объект в вашем пользовательском интерфейсе способен получать фокус (тогда как обычно он не может получать фокус), добавьте XML-атрибут `android:focusable` в объект View в объявлении макета. Установите для него значение `true`. Можно также объявить объект View, способным получать фокус в режиме касания с помощью `android:focusableInTouchMode`.

Чтобы запросить передачу фокуса определенному отображаемому объекту, вызовите `requestFocus()`.

Чтобы перехватывать события фокуса (получать уведомления, когда отображаемый объект получает или теряет фокус), используйте метод `onFocusChange()`.

ЛЕКЦИЯ №39-№40. ТИПЫ КОМПОНОВОК ГРАФИЧЕСКОГО ИНТЕРФЕЙСА.

Компоновка — это архитектура расположения элементов интерфейса пользователя для конкретного окна, представляющего Activity. Компоновка

определяет структуру расположения элементов в окне и содержит все элементы, которые отображаются пользователю программы. Это довольно важная тема в разработке мобильных приложений, поскольку проектирование пользовательского интерфейса для мобильных телефонов и планшетов сложнее, чем для настольных систем или для веб-страниц. Экраны мобильных телефонов имеют гораздо меньшие размеры, чем обычные мониторы. Кроме того, существует много разновидностей дисплеев для мобильных телефонов, отличающихся размерами, разрешением и плотностью пикселей.

Необходимо также учесть, что большинство экранов для мобильных телефонов сенсорные, причем они могут быть разного типа. Например, емкостный экран реагирует на касание пальцем, а для взаимодействия с резистивным экраном используется стилус. Поэтому важно правильно задавать компоновку и размеры элементов управления, чтобы пользователю было удобно управлять вашим приложением независимо от типа экрана.

Формирование графического интерфейса пользователя

В Android-приложении графический интерфейс пользователя формируется с использованием объектов View и ViewGroup. Объекты View — это основные модули для создания графического интерфейса пользователя на платформе Android. Класс View служит базовым для классов элементов управления, называемых виджетами, — текстовых полей, кнопок и т. д. Объект View является структурой, свойства которой сохраняют параметры компоновки и содержание для определенной прямоугольной области экрана. В графическом интерфейсе пользователя объект View является точкой взаимодействия пользователя и программы. Класс ViewGroup представляет собой контейнер, который служит ядром для подклассов, называемых компоновками (layouts). Эти классы формируют расположение элементов пользовательского интерфейса на форме и содержат дочерние элементы View или ViewGroup. При разработке пользовательского интерфейса для Android необходимо определить компоновку для каждого Activity в виде дерева, используя иерархии узлов View и ViewGroup.

Прорисовка графического интерфейса на экране

При запуске программы система Android получает ссылку на корневой узел дерева компоновки и использует ее для прорисовки графического интерфейса на экране мобильного устройства. Система также анализирует элементы дерева от вершины дерева иерархии, прорисовывая дочерние объекты View и ViewGroup и добавляя их родительским элементам. Для этого в методе onCreate() необходимо вызвать метод setContentView(), передав ему в качестве параметра ссылку на ресурс компоновки в следующем виде: R.layout.layout_file_name

Например, если компоновка находится в файле activity_main.xml, ее загрузка

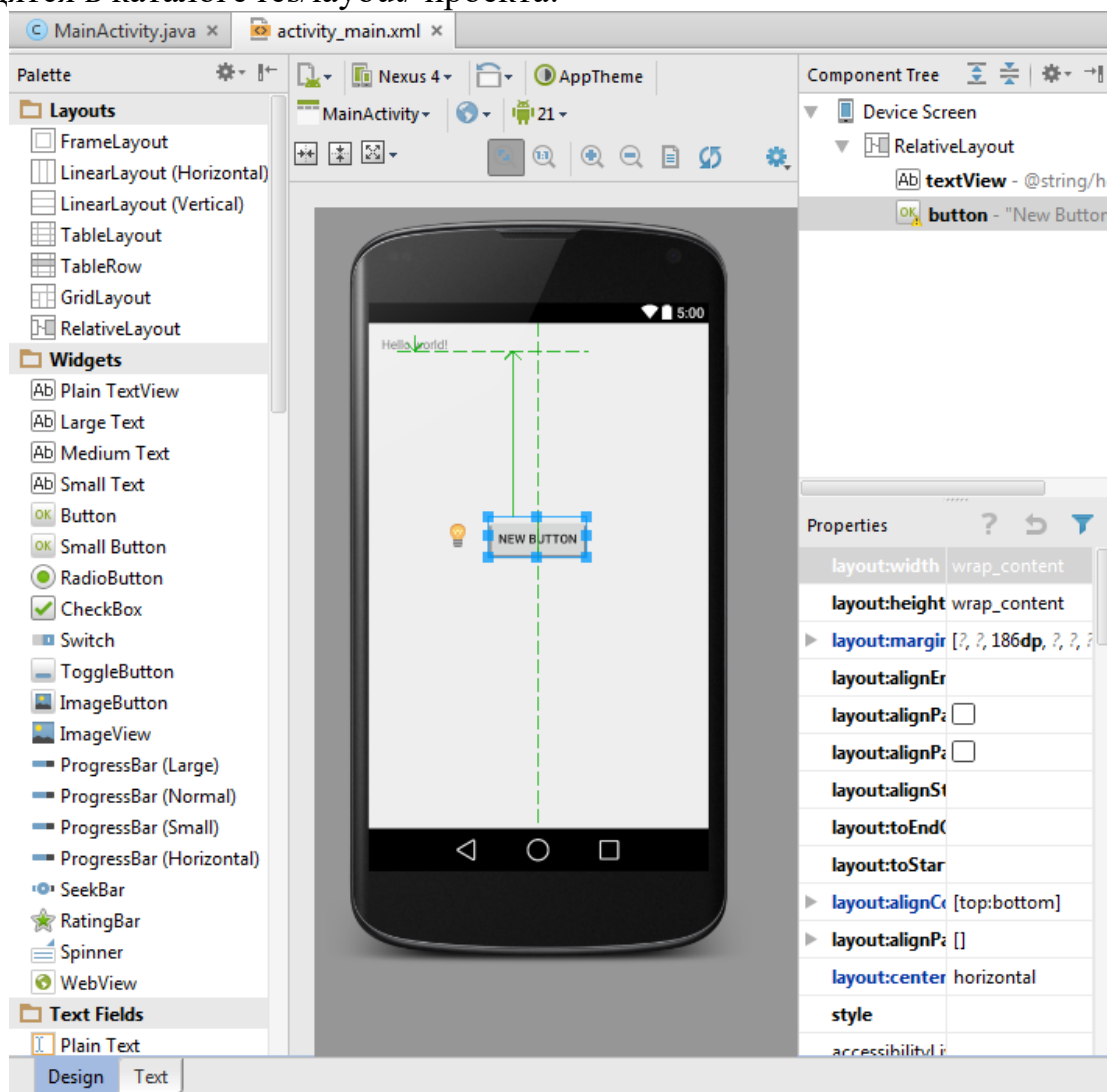
В методе onCreate() происходит так:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
}
```

Прорисовка элементов дерева компоновки всегда начинается с корневого узла. Затем последовательно прорисовываются дочерние объекты дерева компоновки. Это означает, что родители будут прорисовываться раньше, чем их дочерние объекты, т. е. по окончании процесса прорисовки элементов на экране родители будут находиться на заднем плане по отношению к дочерним узлам.

Layout Editor

Android Studio для разработки интерфейса предлагает визуальный редактор компоновки Layout Editor, применяемый для создания и предварительного просмотра создаваемых файлов компоновки, которые находятся в каталоге res/layout/ проекта:



Например, можно создавать XML-компоновки для различных ориентаций экрана мобильного устройства (portrait, landscape), размеров экрана, версии SDK и языков интерфейса. В разделе Outline отображается компоновка в виде дерева. Объекты View — листья дерева, объекты ViewGroup — ветви. В разделе Properties можно задавать свойства для элементов интерфейса.

Создание компоновки

Компоновку можно создавать двумя способами:

объявить элементы пользовательского интерфейса в XML-файле. Android обеспечивает прямой XML-словарь, который соответствует классам View и ViewGroup;

создать компоновку для окна в коде программы во время выполнения — инициализировать объекты Layout и дочерние объекты View, ViewGroup и управлять их свойствами программно.

При разработке пользовательского интерфейса можно применять каждый из этих методов в отдельности или оба сразу для объявления и управления пользовательским интерфейсом в приложении. Например, можно объявить заданную по умолчанию компоновку окна вашего приложения в XML-файле, включая экранные элементы, которые появятся в них, и их свойства, а затем добавить код в приложение, который во время выполнения изменит состояние объектов на экране, включая объявленные в XML-файле. Обычно, если интерфейс статический, используется XML-файлы. Преимущество объявления пользовательского интерфейса в XML-файле состоит в том, что это дает возможность отделить дизайн приложения от программного кода, который управляет поведением приложения. Ваше описание пользовательского интерфейса является внешним по отношению к программному коду. Это означает, что вы можете изменять пользовательский интерфейс в файле компоновки без необходимости изменения вашего программного кода.

Используя XML-компоновку, можно быстро проектировать пользовательский интерфейс тем же самым способом, которым вы создаете веб-страницы в HTML, — с рядом вложенных элементов. Каждый файл компоновки должен содержать только один корневой элемент, который должен быть объектом View или ViewGroup. К корневому элементу, вы можете добавить дополнительные объекты компоновки как дочерние элементы.

Например, вот этот XML-файл компоновки приложения нашего тестового приложения:

```
activity_main.xml
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
```

```
android:layout_height="match_parent"
android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
android:paddingBottom="@dimen/activity_vertical_margin"
tools:context=".MainActivity">
```

```
<TextView
    android:text="@string/hello_world"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```

```
</RelativeLayout>
```

В этой XML-компоновке есть корневой элемент `<RelativeLayout>` и только один дочерний элемент `TextView` (текстовое поле), которые имеют следующие атрибуты:

`xmlns:android` — объявление пространства имен XML, которое сообщает среде Android, что вы ссылаетесь на общие атрибуты, определенные в пространстве имен Android. В каждом файле компоновки у корневого элемента должен быть этот атрибут со значением `"http://schemas.android.com/apk/res/android";`

`android:layout_width` — атрибут определяет, сколько из доступной ширины экрана должен использовать этот объект View (или ViewGroup). В нашем случае он — единственный объект, таким образом, можно растянуть его на весь экран, которому в данном случае соответствует значение `fill_parent`;

`android:layout_height` — аналогичен атрибуту `android:layout_width` за исключением того, что он ссылается на доступную высоту экрана.

группа

`android:padding...` — задают отступы вложенных элементов от границ контейнера.

Элементы управления в Android могут иметь множество различных атрибутов, которые определяют внешний вид и поведение данного элемента и различаются в зависимости от типа элемента. Мы их будем постепенно изучать при рассмотрении элементов управления в следующих статьях.

Типы компоновок

Для создания окон существует несколько стандартных типов компоновок, которые вы можете использовать в разрабатываемых приложениях:

FrameLayout;
LinearLayout;

TableLayout;
GridLayout;
RelativeLayout.

Каждый из этих типов компоновки предлагает уникальный набор параметров, которые используются, чтобы определить позиции дочерних элементов и структуру компоновки на экране. В зависимости от требований, предъявляемых к пользовательскому интерфейсу, выбирается наиболее подходящий тип компоновки. В следующих уроках мы рассмотрим все варианты компоновок и их использование:

Компоновка LinearLayout

Компоновка LinearLayout наиболее простая в использовании и применяется чаще всего. Это компоновка с последовательным линейным расположением дочерних элементов. Несмотря на то, что мастер по умолчанию создает шаблон с относительной компоновкой RelativeLayout, во многих случаях при создании пользовательского интерфейса лучше использовать LinearLayout.

Ориентация дочерних компонентов

Компоновка LinearLayout выравнивает все дочерние объекты View в одном направлении — вертикально или горизонтально, в зависимости от того, как определен атрибут ориентации android:orientation:
android:orientation="horizontal"

или

android:orientation="vertical"

Все дочерние элементы помещаются в стек один за другим, так что вертикальный список объектов View будет иметь только один дочерний элемент в строке независимо от того, насколько широким он является. Горизонтальное расположение списка обеспечивает размещение элементов в одну строку с высотой, равной высоте самого высокого дочернего элемента списка.

Поменяйте код в файле res/layout/activity_main.xml: создайте корневой элемент LinearLayout с тремя дочерними кнопками:

```
activity_main.xml
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal">

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
```

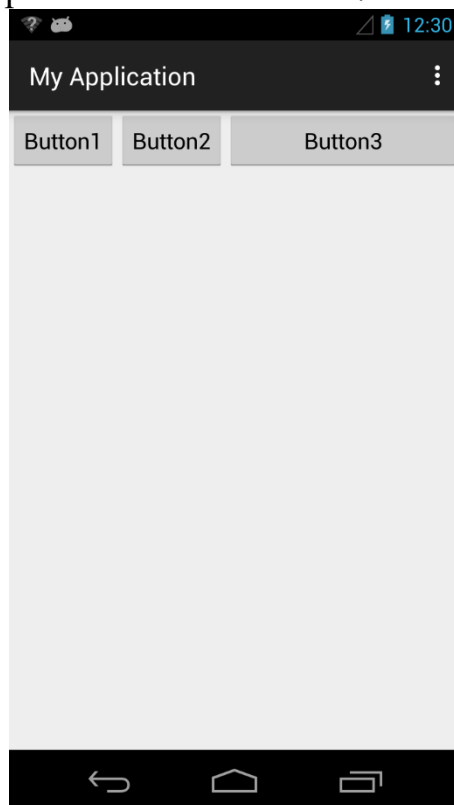
```

        android:layout_height="wrap_content"
        android:text="Button1"/>
    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button2"/>
    <Button
        android:id="@+id/button3"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:text="Button3"/>

</LinearLayout>

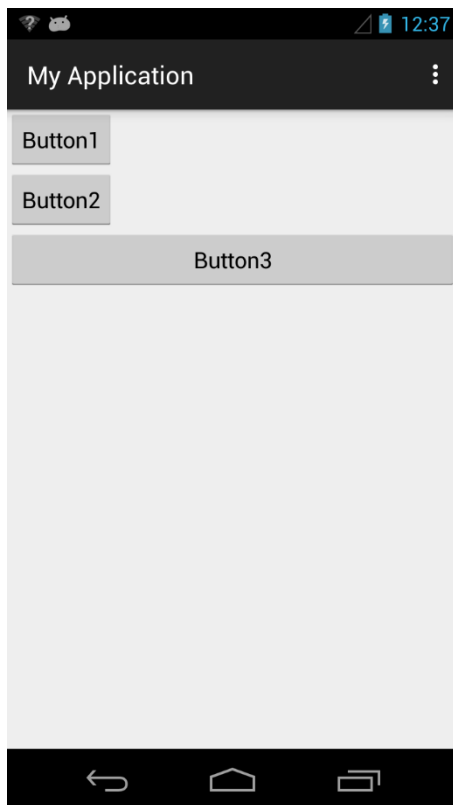
```

Обратите внимание, что у первых двух кнопок атрибуту `android:layout_width` присвоено значение `wrap_content`, а у третьей кнопки — `fill_parent`, т. е. последняя кнопка заполнит оставшееся свободное пространство в компоновке.



В результате получится линейное горизонтальное размещение дочерних элементов. Если изменить в корневом элементе значение атрибута `android:layout_height`:

`android:orientation="vertical"` элементы в контейнере расположатся вертикально. Внешний вид экрана для компоновки `LinearLayout` с горизонтальной и вертикальной ориентациями элементов:



Вес элементов

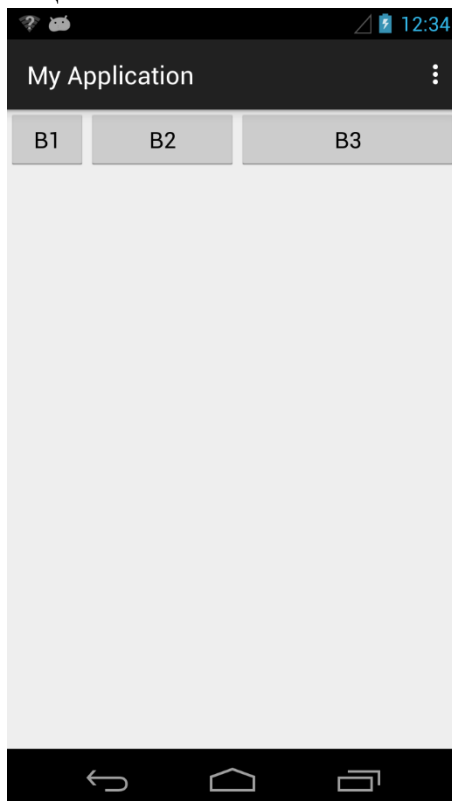
Компоновка `LinearLayout` также поддерживает атрибут `android:layout_weight`, который назначает индивидуальный вес для дочернего элемента. Данный атрибут определяет "важность" объекта `View` и позволяет этому элементу расширяться, чтобы заполнить любое оставшееся пространство в родительском объекте `View`. Заданный по умолчанию вес является нулевым. Например, если есть три текстовых поля, и двум из них объявлен вес со значением 1, в то время как другому не дается никакого веса (0), третье текстовое поле без веса не будет расширяться и займет область, определяемую размером текста, отображаемого этим полем. Другие два расширятся одинаково, чтобы заполнить остаток пространства, не занятого третьим полем. Если третьему полю присвоить вес 2 (вместо 0), это поле будет объявлено как "более важное", чем два других, так что третье поле получит 50% общего пространства, в то время как первые два получают по 25% общего пространства. Далее в файле `res/layout/main.xml` создайте корневой элемент `LinearLayout` с тремя дочерними элементами `Button`:

```
activity_main.xml
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
```



```
<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="B1"
    android:layout_weight="0"/>
<Button
    android:id="@+id/button2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="B2"
    android:layout_weight="1"/>
<Button
    android:id="@+id/button3"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:text="B3"
    android:layout_weight="2"/>
</LinearLayout>
```

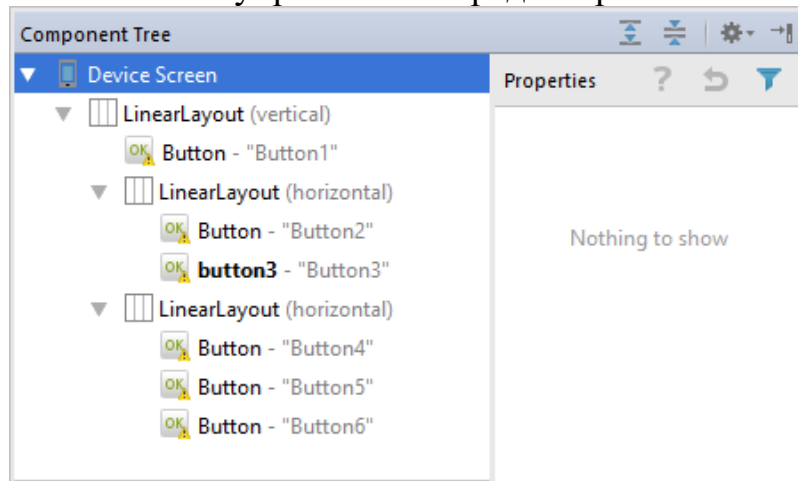
Внешний вид экрана и влияние атрибута `android:layout_weight` будет следующим:



Обратите внимание, как различные XML-атрибуты определяют поведение элемента. Попробуйте поэкспериментировать с разными значениями `layout_weight` для дочерних элементов, чтобы увидеть, как будет распределяться доступное пространство для элементов.

Вложенные компоновки

Компоновки могут быть и вложенными. При проектировании окон с многочисленными элементами управления для заданного расположения элементов часто задаются вложенные компоновки, которые являются контейнерами для элементов управления. Например, для корневой компоновки `LinearLayout` с атрибутом `orientation="vertical"` мы можем задать простой дочерний элемент `Button` и еще два контейнера `LinearLayout` с атрибутом `orientation="horizontal"`, которые, в свою очередь, содержат дочерние элементы управления. В редакторе внешний вид будет таким:



Код файла `main.xml` в результате должен получиться таким:

```
activity_main.xml
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

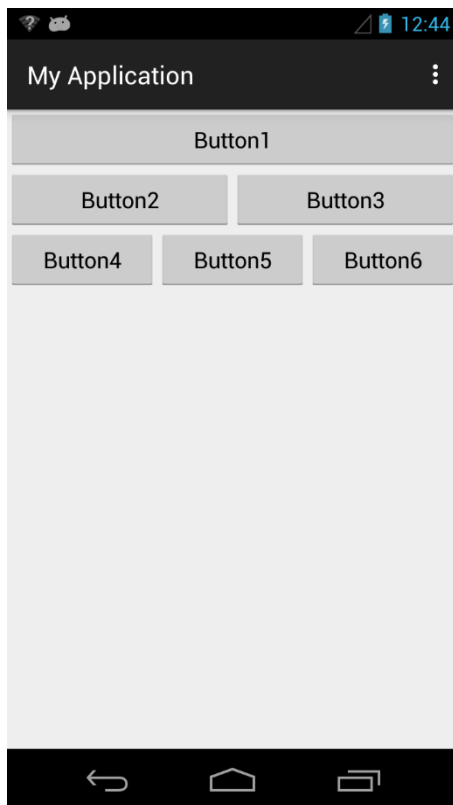
    <Button
        android:text="Button1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"/>

    <LinearLayout
        android:orientation="horizontal"
        android:layout_height="wrap_content"
        android:layout_width="match_parent">
        <Button
            android:text="Button2"
            android:layout_width="wrap_content"
            android:layout_weight="1"
            android:layout_height="wrap_content"/>
        <Button android:text="Button3"
```

```
        android:id="@+id/button3"
        android:layout_weight="1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    </LinearLayout>
```

```
<LinearLayout
    android:orientation="horizontal"
    android:layout_height="wrap_content"
    android:layout_width="match_parent">
    <Button
        android:text="Button4"
        android:layout_weight="1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <Button
        android:text="Button5"
        android:layout_weight="1"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"/>
    <Button
        android:text="Button6"
        android:layout_weight="1"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"/>
</LinearLayout>
</LinearLayout>
```

Внешний вид экрана с вложенными компоновками будет следующим:



Такое применение вложенных компоновок позволяет строить гибкие и легко перенастраиваемые окна и является самым распространенным способом при создании пользовательского интерфейса для Android-приложений.

Компоновка FrameLayout

Компоновка FrameLayout является самой простой. Это в основном пустое пространство на экране, которое можно заполнить только единственным дочерним объектом View или ViewGroup. Все дочерние элементы FrameLayout прикрепляются к левому верхнему углу экрана и будут накладываться друг на друга. Вот пример простейшей компоновки FrameLayout с одним дочерним элементом Button:

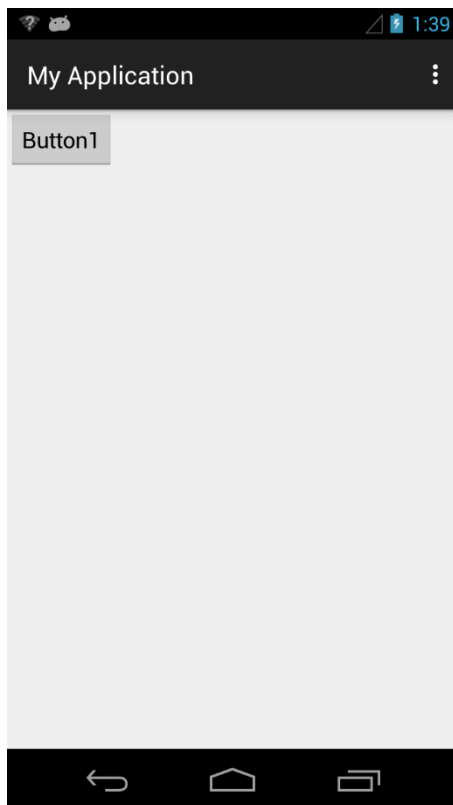
activity_main.xml

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_height="wrap_content"
    android:layout_width="fill_parent">
```

```
    <Button
        android:text="Button1"
        android:id="@+id/Button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
```

```
</FrameLayout>
```

Внешний вид экрана с компоновкой FrameLayout должен получиться таким:



В компоновке `FrameLayout` нельзя определить различное местоположение для дочернего объекта `View`. Последующие дочерние объекты `View` будут просто рисоваться поверх предыдущих, частично или полностью затеняя их, если находящийся сверху объект непрозрачен, поэтому единственный дочерний элемент для `FrameLayout` обычно растянут до размеров родительского контейнера и имеет атрибуты `layout_width="fill_parent"` и `layout_height="fill_parent"`.

Добавим в файл `activity_main` еще один элемент `Button`:

`activity_main.xml`

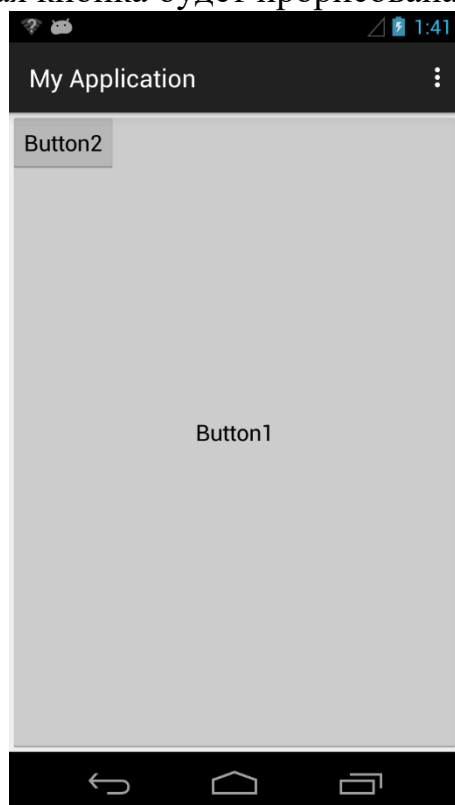
```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
```

```
    <Button
        android:text="Button1"
        android:id="@+id/Button1"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"/>
```

```
    <Button
        android:text="Button2"
        android:id="@+id/Button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
```

```
</FrameLayout>
```

Вторая кнопка будет прорисована поверх первой и будет затенять ее:



Компоновка `FrameLayout` применяется довольно редко, т. к. не позволяет создавать сложные окна с множеством элементов. В практических приложениях эту компоновку обычно используют для создания оверлеев. Например, если у вас в окне выводится изображение, занимающее весь экран (это может быть карта, загруженная с сервиса Google Map, или картинка с видеокамеры), можно сверху на изображении расположить элементы управления (в дочернем контейнере, если их несколько), скажем, кнопки для управления камерой и рамку видоискателя, а также выводить индикацию времени съемки и другую полезную информацию.

Компоновка `TableLayout`

Компоновка `TableLayout` позиционирует свои дочерние элементы в строки и столбцы. `TableLayout` не отображает линии обрамления для их строк, столбцов или ячеек. Кроме того, `TableLayout` может иметь строки с разным количеством ячеек. При формировании компоновки таблицы некоторые ячейки при необходимости можно оставлять пустыми. При создании компоновки для строк используются объекты `TableRow`, которые являются дочерними классами `TableLayout` (каждый `TableRow` определяет единственную строку в таблице). Строка может совсем не иметь ячеек или иметь одну и более ячеек, которые являются контейнерами для других объектов `View` или `ViewGroup`. Ячейка может также быть объектом `ViewGroup` (например, допускается вложить другой `TableLayout` или `LinearLayout` как ячейку).

Для примера с использованием компоновки `TableLayout` можно создать окно, похожее на наборную панель телефона с 12 кнопками. В окне `Layout Editor` создайте `TableLayout` с четырьмя дочерними `TableRow` и двенадцатью кнопками, по три кнопки в каждой строке. Для каждого элемента `TableRow` на вкладке `Properties` задайте свойства:

Layout height — `wrap_content`;
Layout width — `wrap_content`;
Gravity — `center`.

Свойство `gravity` задает выравнивание дочерних элементов в контейнере, в данном случае — по центру. Для каждой кнопки на вкладке `Properties` задайте свойства:

Layout height — `wrap_content`;
Layout width — `70dp`. Это фиксированная ширина элемента в `dp` (`density-independent pixels`) — независимые от плотности экрана пиксели. Подробнее о них поговорим в следующих уроках.

Надписи на кнопках сделайте так, как на телефонной клавиатуре (1, 2, 3, 4, 5, 6, 7, 8, 9, *, 0, #).

Файл компоновки должен получиться таким:

```
activity_main.xml
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/TableLayout01"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="center">
    <TableRow
        android:id="@+id/TableRow01"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:gravity="center">
        <Button
            android:id="@+id/Button01"
            android:layout_height="wrap_content"
            android:text="1"
            android:layout_width="70dp"/>
        <Button
            android:id="@+id/Button02"
            android:layout_height="wrap_content"
            android:text="2"
```

```

        android:layout_width="70dp"/>
    <Button
        android:id="@+id/Button03"
        android:layout_height="wrap_content"
        android:text="3"
        android:layout_width="70dp"/>
</TableRow>
<TableRow
    android:id="@+id/TableRow02"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:gravity="center">
    <Button
        android:id="@+id/Button04"
        android:layout_height="wrap_content"
        android:layout_width="70dp"
        android:text="4"/>
    <Button
        android:id="@+id/Button05"
        android:layout_height="wrap_content"
        android:layout_width="70dp"
        android:text="5"/>
    <Button
        android:id="@+id/Button06"
        android:layout_height="wrap_content"
        android:layout_width="70dp"
        android:text="6"/>
</TableRow>
<TableRow
    android:id="@+id/TableRow03"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:gravity="center">
    <Button
        android:id="@+id/Button07"
        android:layout_height="wrap_content"
        android:layout_width="70dp"
        android:text="7"/>
    <Button
        android:id="@+id/Button08"
        android:layout_height="wrap_content"
        android:layout_width="70dp"
        android:text="8"/>
    <Button
        android:id="@+id/Button09"

```

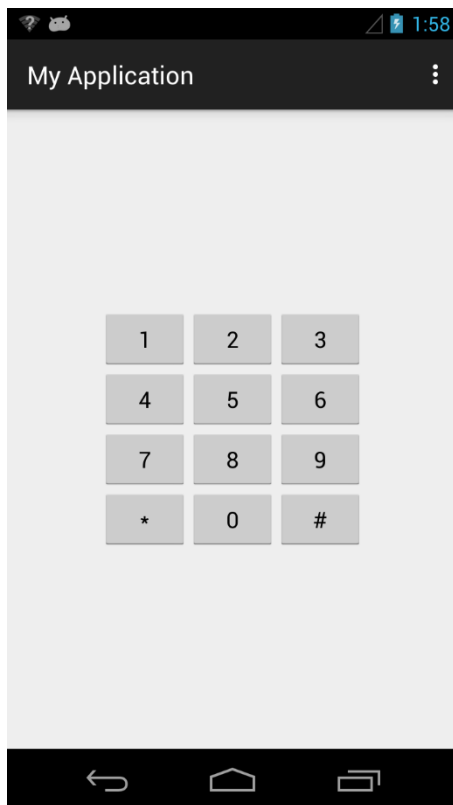


```

        android:layout_height="wrap_content"
        android:layout_width="70dp"
        android:text="9"/>
</TableRow>
<TableRow
    android:id="@+id/TableRow04"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:gravity="center">
    <Button
        android:id="@+id/Button10"
        android:layout_height="wrap_content"
        android:layout_width="70dp"
        android:text="*" />
    <Button
        android:id="@+id/Button11"
        android:layout_height="wrap_content"
        android:layout_width="70dp"
        android:text="0" />
    <Button
        android:id="@+id/Button12"
        android:layout_height="wrap_content"
        android:layout_width="70dp"
        android:text="#" />
</TableRow>
</TableLayout>

```

В результате внешний вид экрана должен получиться в виде телефонной клавиатуры:



Компоновка `TableLayout` на практике применяется довольно редко, обычно вместо нее используют сочетание компоновок `LinearLayout` или `GridLayout`, которая будет рассмотрена далее. Компоновку `TableLayout` удобно использовать, если расположение элементов представлено в виде "таблицы", как в нашем примере. Кроме того, существует компоновка `GridLayout`, которая внешне аналогична `TableLayout`, но имеет дополнительные возможности.

Компоновка `GridLayout`

Компоновка `GridLayout` (сетка) — это относительно новая компоновка, появилась в версии 4.0 (API Level 14). Она похожа на `TableLayout`, но позволяет более гибко работать с размерами ячеек и их расположением. В отличие от `TableLayout`, она не фиксирует дочерние элементы в `TableRow`, а использует атрибуты `android:columnCount` и `android:rowCount` для задания количества дочерних элементов соответственно в столбце или в строке сетки. Кроме того, в `GridLayout` можно объединять ячейки по горизонтали, используя атрибут `android:layout_columnSpan`, задавая им количество ячеек, которые должны быть объединены. Для примера используем уже созданный набор кнопок из предыдущего урока, только поместим их в `GridLayout` и уберем элементы `TableRow`. Для кнопки "0" зададим атрибут `android:layout_columnSpan="3"`, а для кнопки "*" — `android:layout_columnSpan="2"`.

`activity_main.xml`

```
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
```

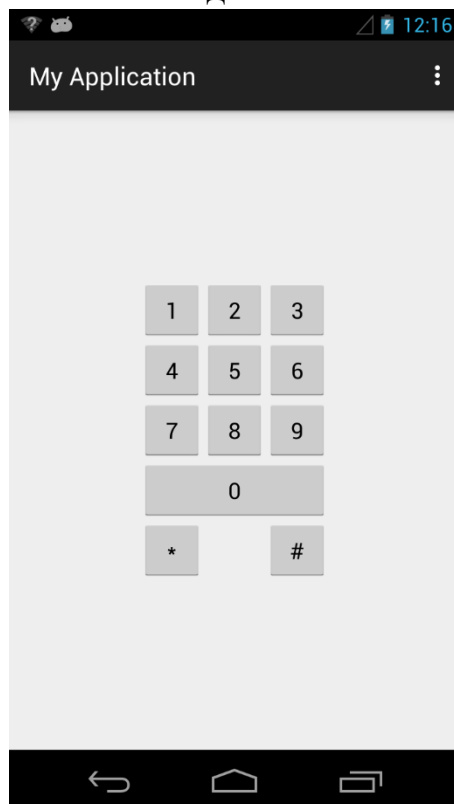
```
android:columnCount="3" >
<Button
    android:id="@+id/Button01"
    android:layout_height="wrap_content"
    android:layout_width="50dp"
    android:text="1"/>
<Button
    android:id="@+id/Button02"
    android:layout_height="wrap_content"
    android:layout_width="50dp"
    android:text="2"/>
<Button
    android:id="@+id/Button03"
    android:layout_height="wrap_content"
    android:layout_width="50dp"
    android:text="3"/>
<Button
    android:id="@+id/Button04"
    android:layout_height="wrap_content"
    android:layout_width="50dp"
    android:text="4"/>
<Button
    android:id="@+id/Button05"
    android:layout_height="wrap_content"
    android:layout_width="50dp"
    android:text="5"/>
<Button
    android:id="@+id/Button06"
    android:layout_height="wrap_content"
    android:layout_width="50dp"
    android:text="6"/>
<Button
    android:id="@+id/Button07"
    android:layout_height="wrap_content"
    android:layout_width="50dp"
    android:text="7"/>
<Button
    android:id="@+id/Button08"
    android:layout_height="wrap_content"
    android:layout_width="50dp"
    android:text="8"/>
<Button
    android:id="@+id/Button09"
    android:layout_height="wrap_content"
    android:layout_width="50dp"
```

```

        android:text="9"/>
<Button
    android:id="@+id/Button11"
    android:layout_height="wrap_content"
    android:layout_width="150dp"
    android:layout_columnSpan="3"
    android:text="0"/>
<Button
    android:id="@+id/Button10"
    android:layout_height="wrap_content"
    android:layout_width="50dp"
    android:layout_columnSpan="2"
    android:text="*"/>
<Button
    android:id="@+id/Button12"
    android:layout_height="wrap_content"
    android:layout_width="50dp"
    android:text="#"/>
</GridLayout>

```

Внешний вид компоновки на смартфоне:



Обратите внимание, что объединение ячеек автоматически не растягивает элемент, находящийся в них, как в случае с кнопкой "*". Размеры элемента в ячейке надо задавать отдельно. Для кнопки "0", которая растянута на 3 ячейки, кроме атрибута `android:layout_columnSpan` мы изменили ширину кнопки

Компоновка RelativeLayout

Компоновка RelativeLayout позволяет дочерним объектам определять свою позицию относительно родительского объекта или относительно соседних дочерних элементов (по идентификатору элемента). В RelativeLayout дочерние элементы расположены так, что если первый элемент расположен по центру экрана, другие элементы, выровненные относительно первого элемента, будут выровнены относительно центра экрана. При таком расположении, при объявлении компоновки в XML-файле, элемент, на который будут ссылаться для позиционирования другие объекты, должен быть объявлен раньше, чем другие элементы, которые обращаются к нему по его идентификатору. Если в программном коде мы не работаем с некоторыми элементами пользовательского интерфейса, создавать идентификаторы для них необязательно, однако определение идентификаторов для объектов важно при создании RelativeLayout. В компоновке RelativeLayout расположение элемента может определяться относительно другого элемента, на который ссылаются через его уникальный идентификатор: `android:layout_toLeftOf="@id/TextView1"`

Давайте теперь создадим пример окна с корневой компоновкой RelativeLayout, в котором будут семь элементов Button. В файле компоновки `activity_main.xml` напишите следующий код:

```
activity_main.xml
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_height="fill_parent"
    android:layout_width="fill_parent">

    <Button
        android:id="@+id/button_center"
        android:text="Center"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerVertical="true"
        android:layout_centerInParent="true"/>
    <Button
        android:id="@+id/button_bottom"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bottom"
        android:layout_centerHorizontal="true"
        android:layout_alignParentBottom="true"/>
    <Button
        android:id="@+id/button_top"
        android:layout_width="wrap_content"
```

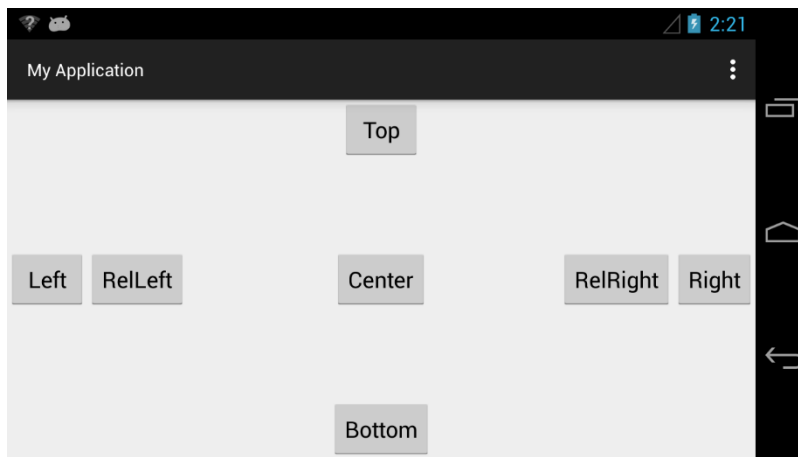
```

        android:layout_height="wrap_content"
        android:text="Top"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"/>
<Button
    android:id="@+id/button_left"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Left"
    android:layout_alignParentLeft="true"
    android:layout_centerVertical="true"/>
<Button
    android:id="@+id/button_right"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Right"
    android:layout_alignParentRight="true"
    android:layout_centerVertical="true"/>
<Button
    android:id="@+id/button_rel_right"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_toLeftOf="@id/button_right"
    android:layout_alignTop="@id/button_right"
    android:text="RelRight"/>
<Button
    android:id="@+id/button_rel_left"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_toRightOf="@id/button_left"
    android:layout_alignTop="@id/button_left"
    android:text="RelLeft"/>

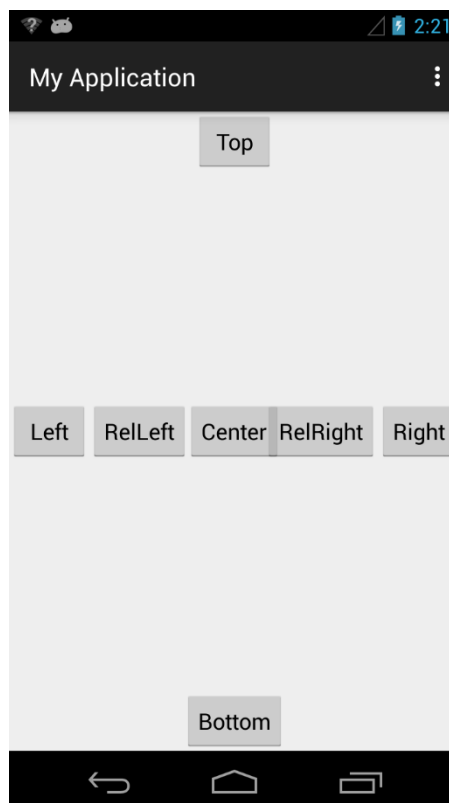
```

</RelativeLayout>

Кстати, обратите внимание, что атрибуты, которые обращаются к идентификаторам относительных элементов (например, `layout_toLeftOf`), используют синтаксис относительного ресурса `@id/id`. Внешний вид экрана с компоновкой `RelativeLayout` должен получиться таким:



Тип компоновки RelativeLayout применяется не так часто, как, например, LinearLayout, несмотря на то, что мастер создания проекта ее использует ее в качестве компоновки по умолчанию. Тем более что такое задание расположения элементов зависит от разрешения и ориентации экрана мобильного устройства. Если вы будете использовать RelativeLayout в собственных приложениях, всегда проверяйте внешний вид окна для различных разрешений и ориентации экрана, поскольку возможно наложение элементов друг на друга. В нашем случае, например, в вертикальной ориентации экрана кнопки Center и RelRight немного накладываются друг на друга:

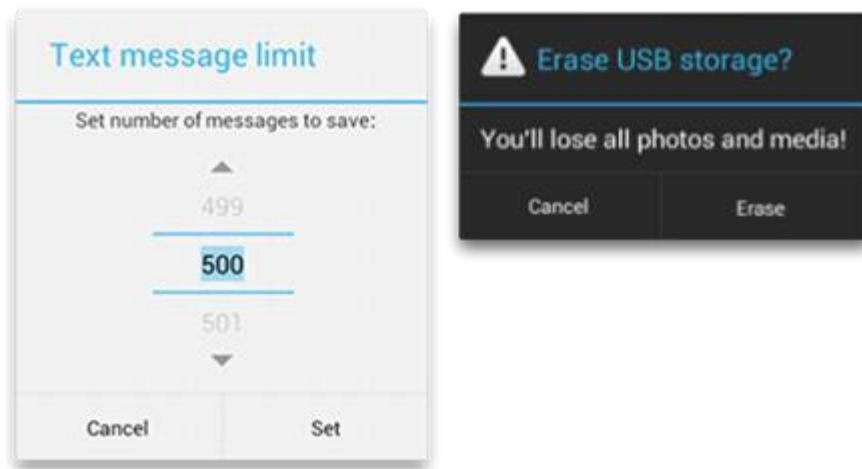


ЛЕКЦИЯ №41. ДИАЛОГОВЫЕ ОКНА.

Диалоговое окно — это небольшое окно, которое предлагает пользователю принять решение или ввести дополнительную информацию. Диалоговое окно не заполняет весь экран и, как правило, используется при

модальных событиях, для продолжения которых требуется действие пользователя.

Дизайн диалогового окна



Класс Dialog — это базовый класс для создания диалоговых окон, но реализовывать напрямую класс Dialog не рекомендуется. Вместо этого следует использовать один из следующих подклассов:

AlertDialog

Диалоговое окно, в котором могут отображаться заголовок, кнопки вплоть до трех штук, список из выбираемых элементов либо пользовательский макет.

DatePickerDialog или TimePickerDialog

Диалоговое окно с предопределенным пользовательским интерфейсом, с помощью которого пользователь указывает значения даты или времени.

Следует избегать ProgressDialog

В Android имеется другой класс диалоговых окон под названием ProgressDialog, которые отображают диалоговое окно с индикатором выполнения процесса. Тем не менее, если необходимо отобразить прогресс загрузки или неопределенный процесс, рекомендуется следовать инструкциям по дизайну Выполнение & Операции, чтобы использовать ProgressBar в вашем макете.

Эти классы определяют стиль и структуру вашего диалогового окна, однако следует использовать DialogFragment в качестве контейнера вашего диалогового окна. Класс DialogFragment предоставляет все функции, необходимые для создания диалогового окна и управления его внешним видом, вместо вызова методов к объекту Dialog.

Использование DialogFragment для управления диалоговым окном обеспечивает корректную обработку событий жизненного цикла, таких как нажатие пользователем кнопки Назад или поворот экрана. С помощью класса DialogFragment также происходит повторное использование пользовательского интерфейса диалогового окна в качестве встраиваемого компонента в пользовательский интерфейс более высокого уровня — подобно традиционному классу Fragment (например, когда необходимо различное

отображение пользовательского интерфейса диалогового окна на больших и маленьких экранах).

В следующих разделах руководства описано использование DialogFragment в сочетании с объектом AlertDialog. Если необходимо создать элемент выбора даты или времени, обратитесь к руководству Элементы выбора.

Примечание: Поскольку класс DialogFragment изначально включен в Android 3.0 (уровень API 11), в настоящем документе описывается использование класса DialogFragment, предоставляемого в Библиотеке поддержки. После добавления этой библиотеки в приложение появится возможность использовать DialogFragment и множество других API на устройствах, работающих на Android 1.6 и выше. Если минимальная версия вашего приложения поддерживает уровень API 11 и выше, можно использовать фреймворк-версию DialogFragment, но следует иметь в виду, что данный документ ссылается на API со вспомогательными библиотеками. При использовании вспомогательной библиотеки необходимо импортировать класс `android.support.v4.app.DialogFragment`, а не `android.app.DialogFragment`.

Создание фрагмента диалогового окна

Вы можете реализовать широкие возможности дизайна диалоговых окон, включая создание пользовательских макетов, а также пути, описанные в руководстве по дизайну Диалоговых окон — путем расширения DialogFragment и создания AlertDialog в методе обратного вызова onCreateDialog().

Например, имеется базовый класс AlertDialog, управляемый в рамках DialogFragment:

```
public class FireMissilesDialogFragment extends DialogFragment {
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        // Use the Builder class for convenient dialog construction
        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
        builder.setMessage(R.string.dialog_fire_missiles)
            .setPositiveButton(R.string.fire, new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int id) {
                    // FIRE ZE MISSILES!
                }
            })
            .setNegativeButton(R.string.cancel, new
DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int id) {
                    // User cancelled the dialog
                }
            });
        // Create the AlertDialog object and return it
        return builder.create();
    }
}
```

}

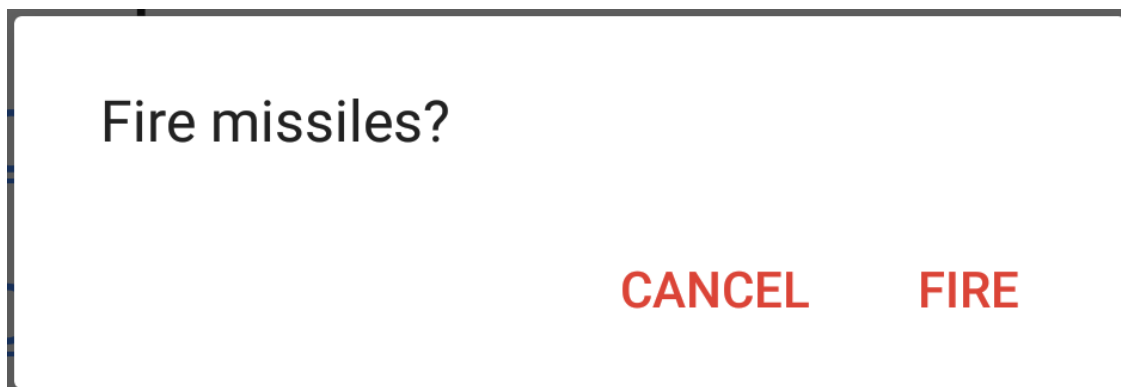


Рисунок 1. Диалоговое окно с сообщением и двумя кнопками действия.

Итак, после создания экземпляра этого класса и вызова `show()` к этому объекту появляется диалоговое окно, показанное на рисунке 1.

В следующем разделе предоставлена более подробная информация об использовании API `AlertDialog.Builder` для создания диалоговых окон.

В зависимости от сложности создаваемого диалогового окна возможно реализовать множество других методов обратного вызова в `DialogFragment`, включая все базовые методы жизненных циклов фрагментов.

Создание диалогового окна оповещения

С помощью класса `AlertDialog` создается многообразие решений касательно внешнего вида диалогового окна, и зачастую этого класса вполне достаточно. Как показано на рис. 2, диалоговое окно состоит из трех областей:

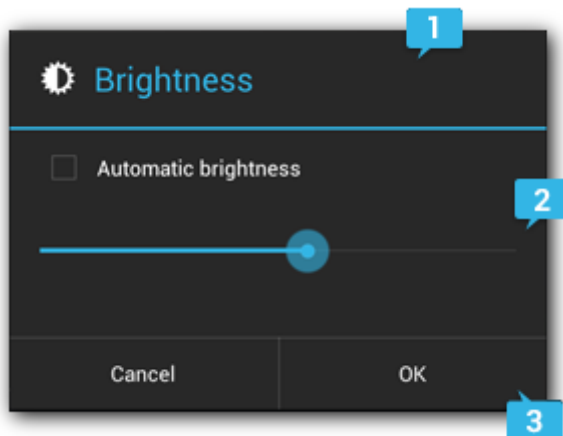


Рисунок 2. Макет диалогового окна.

Заголовок

Это дополнительная возможность, которая используется только в случае, если область содержимого занята подробным сообщением, списком или пользовательским макетом. Если необходимо отобразить простое сообщение или вопрос (как, например, в диалоге на рисунке 1), заголовок не нужен.

Область содержимого

Здесь может отображаться сообщение, список или другой пользовательский макет.

Кнопки действия

В диалоговом окне не должно содержаться более трех кнопок действия.

Класс AlertDialog.Builder предоставляет API, с помощью которых можно создавать AlertDialog с этими видами содержимого, включая пользовательский макет.

Создание AlertDialog:

```
// 1. Instantiate an AlertDialog.Builder with its constructor
AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());

// 2. Chain together various setter methods to set the dialog characteristics
builder.setMessage(R.string.dialog_message)
                                   .setTitle(R.string.dialog_title);

// 3. Get the AlertDialog from create()
AlertDialog dialog = builder.create();
```

В следующих главах показано, как определять различные атрибуты диалоговых окон с помощью класса AlertDialog.Builder.

Добавление кнопок

Для добавления кнопок, изображенных на рисунке 2, вызывайте методы setPositiveButton() и setNegativeButton():

```
AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
// Add the buttons
builder.setPositiveButton(R.string.ok, new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int id) {
        // User clicked OK button
    }
});
builder.setNegativeButton(R.string.cancel, new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int id) {
        // User cancelled the dialog
    }
});
// Set other dialog properties
...

// Create the AlertDialog
AlertDialog dialog = builder.create();
```

Методы set...Button() предполагают заголовок для кнопки (реализуемый через строковый ресурс) и DialogInterface.OnClickListener, который определяет действие, следующее за нажатием кнопки пользователем.

Реализована возможность добавлять три различных вида кнопок действий:

Положительные

Используются для подтверждения и продолжения действия (кнопка «ОК»).

Отрицательные

Используются для отмены действия.

Нейтральные

Используются в случаях, когда пользователь может не желать продолжить действие, но при этом необязательно хочет его отменить. Появляется между положительными и отрицательными кнопками. Примером такого действия может быть «Напомнить позже».

Можно добавлять только одну кнопку каждого вида в AlertDialog. Это означает, что нельзя использовать более одной «положительной» кнопки.

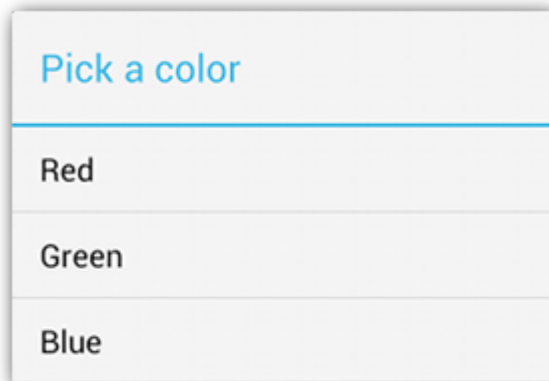


Рисунок 3. Диалоговое окно с заголовком и списком.

Добавление списка

В API AlertDialog реализована возможность использования трех видов списков:

Традиционный список с выбором одного варианта

Интерактивный список с выбором одного варианта (переключатели)

Интерактивный список с выбором нескольких вариантов (флажки)

Для создания списка с выбором одного варианта, как на рисунке 3, используйте метод setItems():

```
@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
    builder.setTitle(R.string.pick_color)
        .setItems(R.array.colors_array, new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int which) {
                // The 'which' argument contains the index position
                // of the selected item
            }
        });
    return builder.create();
}
```

Поскольку список отображается в области содержимого диалогового окна, диалоговое окно не может показать одновременно сообщение и список,

поэтому необходимо задать заголовок диалогового окна с помощью `setTitle()`. Для указания элементов списка необходимо вызвать `setItems()`, передающий указатель. В качестве другого варианта можно указать список с помощью `setAdapter()`. Наполнение списка динамическими данными (например, из базы данных) происходит с помощью `ListAdapter`.

Если вы предпочтете реализовать список с помощью `ListAdapter`, рекомендуется использовать `Loader`, чтобы содержимое загружалось асинхронно. Подробно этот процесс описан далее в руководстве по Созданию макетов с помощью адаптера и загрузчиков.

Примечание: По умолчанию нажатие по элементу списка отменяет диалоговое окно, за исключением случаев, когда используется один из следующих интерактивных списков.

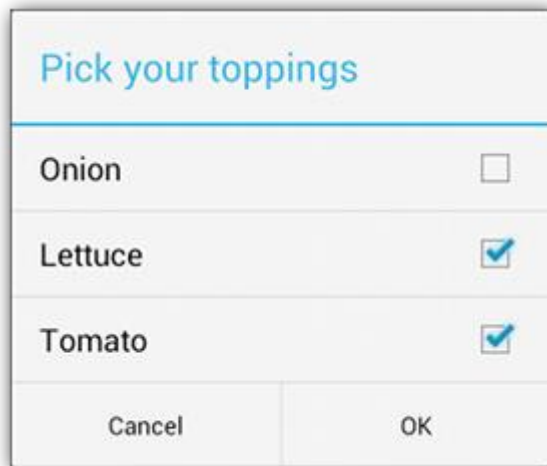


Рисунок 4. Список с несколькими вариантами ответов.

Добавление интерактивного списка с одним или несколькими вариантами ответов

Для добавления списка с несколькими вариантами ответов (флажки) или списка с одним вариантом ответа (переключатели) используйте методы `setMultiChoiceItems()` или `setSingleChoiceItems()` соответственно.

Например, таким образом можно создать список с несколькими вариантами ответов, как на рисунке 4, который сохраняет выбранные элементы в `ArrayList`:

```
@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    mSelectedItems = new ArrayList(); // Where we track the selected items
    AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
    // Set the dialog title
    builder.setTitle(R.string.pick_toppings)
    // Specify the list array, the items to be selected by default (null for none),
    // and the listener through which to receive callbacks when items are selected
    .setMultiChoiceItems(R.array.toppings, null,
        new DialogInterface.OnMultiChoiceClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which,
```

```

        boolean isChecked) {
    if (isChecked) {
        // If the user checked the item, add it to the selected items
        mSelectedItems.add(which);
    } else if (mSelectedItems.contains(which)) {
        // Else, if the item is already in the array, remove it
        mSelectedItems.remove(Integer.valueOf(which));
    }
}
})
// Set the action buttons
.setPositiveButton(R.string.ok, new DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialog, int id) {
        // User clicked OK, so save the mSelectedItems results somewhere
        // or return them to the component that opened the dialog
        ...
    }
})
.setNegativeButton(R.string.cancel, new DialogInterface.OnClickListener()
{
    @Override
    public void onClick(DialogInterface dialog, int id) {
        ...
    }
});

return builder.create();
}

```

Несмотря на то, что и традиционный список, и список с переключателями предполагают действие по выбору одного элемента, вам необходимо использовать [setSingleChoiceItems\(\)](#), чтобы сохранить выбор пользователя. Это значит, что при повторном открытии диалогового окна будет отображаться текущий выбор пользователя, а затем создается список с переключателями.

Создание пользовательского макета

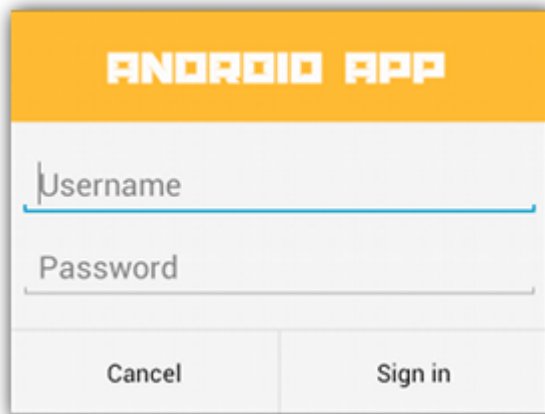


Рисунок 5. Пользовательский макет диалогового окна.

Если в диалоговом окне необходим пользовательский макет, нужно создать макет и добавить его в AlertDialog путем вызова setView() в объекте AlertDialog.Builder.

По умолчанию пользовательский макет заполняет окно диалога, при этом все равно можно использовать методы AlertDialog.Builder для добавления кнопок и заголовка.

В качестве примера на рисунке 5 приведен файл макета для диалогового окна.

```
res/layout/dialog_signin.xml
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
    <ImageView
        android:src="@drawable/header_logo"
        android:layout_width="match_parent"
        android:layout_height="64dp"
        android:scaleType="center"
        android:background="#FFFFBB33"
        android:contentDescription="@string/app_name" />
    <EditText
        android:id="@+id/username"
        android:inputType="textEmailAddress"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="16dp"
        android:layout_marginLeft="4dp"
        android:layout_marginRight="4dp"
        android:layout_marginBottom="4dp"
        android:hint="@string/username" />
    <EditText
        android:id="@+id/password"
```

```

        android:inputType="textPassword"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="4dp"
        android:layout_marginLeft="4dp"
        android:layout_marginRight="4dp"
        android:layout_marginBottom="16dp"
        android:fontFamily="sans-serif"
        android:hint="@string/password"/>
</LinearLayout>

```

Совет. По умолчанию при настройке элемента EditText для типа ввода "textPassword" используется семейство шрифтов фиксированной ширины, поэтому необходимо изменить семейство шрифтов на "sans-serif", чтобы в обоих текстовых полях использовались одинаковые стили шрифта.

Для применения макета в вашем DialogFragment вам понадобится LayoutInflater с getLayoutInflater() и вызов inflate(), где первым параметром будет являться ID ресурса макета, а вторым параметром — исходный вид макета. Затем можно вызвать setView() для размещения макета в диалоговом окне.

```

@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
    // Get the layout inflater
    LayoutInflater inflater = getActivity().getLayoutInflater();

    // Inflate and set the layout for the dialog
    // Pass null as the parent view because its going in the dialog layout
    builder.setView(inflater.inflate(R.layout.dialog_signin, null))
    // Add action buttons
    .setPositiveButton(R.string.signin, new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int id) {
            // sign in the user ...
        }
    })
    .setNegativeButton(R.string.cancel, new DialogInterface.OnClickListener() {
        {
            public void onClick(DialogInterface dialog, int id) {
                LoginDialogFragment.this.getDialog().cancel();
            }
        }
    });
    return builder.create();
}

```


Совет. Если необходимо пользовательское диалоговое окно, можно отображать Activity в качестве диалога вместо API Dialog. Нужно создать операцию и установить тему Theme.Holo.Dialog в элементе манифеста <операция>:

```
<activity android:theme="@android:style/Theme.Holo.Dialog" >
```

Готово. Операция теперь отображается в диалоговом окне, а не в полноэкранный режиме.

Передача событий обратно в основное диалоговое приложение

Когда пользователь нажимает одну из кнопок действий в диалоговом окне, либо выбирает элемент из списка, DialogFragment может самостоятельно произвести необходимое действие, однако зачастую вам может понадобиться доставить информацию о событии операции или фрагменту, которые открыли диалоговое окно. Для этого необходимо определить интерфейс метода для каждого типа события нажатия. Затем этот интерфейс применяется в основном компоненте приложения, которое получает информацию о событиях из диалогового окна.

Например, DialogFragment определяет интерфейс, по который доставляет события обратно в основной компонент операции:

```
public class NoticeDialogFragment extends DialogFragment {
```

```
/* The activity that creates an instance of this dialog fragment must
 * implement this interface in order to receive event callbacks.
 * Each method passes the DialogFragment in case the host needs to query it. */
```

```
public interface NoticeDialogListener {
    public void onDialogPositiveClick(DialogFragment dialog);
    public void onDialogNegativeClick(DialogFragment dialog);
}
```

```
// Use this instance of the interface to deliver action events
NoticeDialogListener mListener;
```

```
// Override the Fragment.onAttach() method to instantiate the
NoticeDialogListener
```

```
@Override
```

```
public void onAttach(Activity activity) {
```

```
    super.onAttach(activity);
```

```
    // Verify that the host activity implements the callback interface
```

```
    try {
```

```
        // Instantiate the NoticeDialogListener so we can send events to the host
```

```
        mListener = (NoticeDialogListener) activity;
```

```
    } catch (ClassCastException e) {
```

```
        // The activity doesn't implement the interface, throw exception
```

```
        throw new ClassCastException(activity.toString()
```

```
            + " must implement NoticeDialogListener");
```

```

    }
}
...
}

```

Операция, выполняемая диалоговым окном, создает экземпляр диалогового окна с помощью конструктора фрагментов диалогового окна и получает события диалога с помощью реализации интерфейса `NoticeDialogListener`:

```

public class MainActivity extends FragmentActivity
    implements NoticeDialogFragment.NoticeDialogListener{
    ...

    public void showNoticeDialog() {
        // Create an instance of the dialog fragment and show it
        DialogFragment dialog = new NoticeDialogFragment();
        dialog.show(getSupportFragmentManager(), "NoticeDialogFragment");
    }

    // The dialog fragment receives a reference to this Activity through the
    // Fragment.onAttach() callback, which it uses to call the following methods
    // defined by the NoticeDialogFragment.NoticeDialogListener interface
    @Override
    public void onDialogPositiveClick(DialogFragment dialog) {
        // User touched the dialog's positive button
        ...
    }

    @Override
    public void onDialogNegativeClick(DialogFragment dialog) {
        // User touched the dialog's negative button
        ...
    }
}

```

Поскольку выполняемая операция реализуется через `NoticeDialogListener` с помощью метода обратного вызова `onAttach()`, во фрагменте диалога могут использоваться методы обратного вызова интерфейса для доставки событий нажатий к операциям:

```

public class NoticeDialogFragment extends DialogFragment {
    ...

    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        // Build the dialog and set up the button click handlers
    }
}

```

```

AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
builder.setMessage(R.string.dialog_fire_missiles)
    .setPositiveButton(R.string.fire, new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int id) {
            // Send the positive button event back to the host activity
            mListener.onDialogPositiveClick(NoticeDialogFragment.this);
        }
    })
    .setNegativeButton(R.string.cancel, new
DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int id) {
            // Send the negative button event back to the host activity
            mListener.onDialogNegativeClick(NoticeDialogFragment.this);
        }
    });
return builder.create();
}
}

```

Отображение диалогового окна

Для отображения диалогового окна необходимо создать экземпляр DialogFragment и вызвать show(), передавая FragmentManager и наименование тега для фрагмента диалога.

Можно получить FragmentManager путем вызова getSupportFragmentManager() из FragmentActivity или getFragmentManager() из Fragment. Пример:

```

public void confirmFireMissiles() {
    DialogFragment newFragment = new FireMissilesDialogFragment();
    newFragment.show(getSupportFragmentManager(), "missiles");
}

```

Второй аргумент, "missiles", — это уникальное наименование тега, которое система использует для сохранения и восстановления состояния фрагмента, когда это необходимо. С помощью этого тега также можно управлять фрагментом путем вызова findFragmentByTag().

Отображение диалогового окна в полноэкранном режиме или в виде встроенного фрагмента

Вам может понадобиться макет пользовательского интерфейса, в котором в некоторых ситуациях часть пользовательского интерфейса должна появляться как диалоговое окно, отображаемое в полноэкранном режиме либо в виде встроенного фрагмента (возможно, в зависимости от того, маленький или большой экран у устройства). С помощью класса DialogFragment обеспечивается гибкость решения, поскольку он может вести себя как встраиваемый Fragment.

Тем не менее, в этом случае нельзя использовать AlertDialog.Builder или другие объекты Dialog для построения диалогового окна. Если необходимо сделать DialogFragment встраиваемым, нужно определить пользовательский интерфейс диалогового окна в макете методом обратного вызова onCreateView().

В качестве примера приведен DialogFragment, который появляется либо в виде диалогового окна, либо в виде встраиваемого фрагмента (используя макет с наименованием `purchase_items.xml`):

```
public class CustomDialogFragment extends DialogFragment {
    /** The system calls this to get the DialogFragment's layout, regardless
        of whether it's being displayed as a dialog or an embedded fragment. */
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        // Inflate the layout to use as dialog or embedded fragment
        return inflater.inflate(R.layout.purchase_items, container, false);
    }

    /** The system calls this only when creating the layout in a dialog. */
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        // The only reason you might override this method when using
        onCreateView() is
        // to modify any dialog characteristics. For example, the dialog includes a
        // title by default, but your custom layout might not need it. So here you can
        // remove the dialog title, but you must call the superclass to get the Dialog.
        Dialog dialog = super.onCreateDialog(savedInstanceState);
        dialog.requestWindowFeature(Window.FEATURE_NO_TITLE);
        return dialog;
    }
}
```

Приведен пример кода, реализующего принятие решения об отображении фрагмента в качестве диалогового окна или полноэкранного пользовательского интерфейса на основе размера экрана:

```
public void showDialog() {
    FragmentManager fragmentManager = getSupportFragmentManager();
    CustomDialogFragment newFragment = new CustomDialogFragment();

    if (mIsLargeLayout) {
        // The device is using a large layout, so show the fragment as a dialog
        newFragment.show(fragmentManager, "dialog");
    } else {
        // The device is smaller, so show the fragment fullscreen
        FragmentTransaction transaction = fragmentManager.beginTransaction();
```

```

        // For a little polish, specify a transition animation
        transaction.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_OP
EN);
        // To make it fullscreen, use the 'content' root view as the container
        // for the fragment, which is always the root view for the activity
        transaction.add(android.R.id.content, newFragment)
            .addToBackStack(null).commit();
    }
}

```

Подробные сведения о выполнении операций с фрагментами приведены в руководстве Фрагменты.

В приведенном примере `mIsLargeLayout` булеан указывает, должно ли текущее устройство использовать большой макет приложения (и отображать фрагмент как диалоговое окно, а не в полноэкранном режиме). Лучшим способом установить такой вид булеана является объявление значения булевой переменной с альтернативным значением для других размеров экранов. В качестве примера приведены два варианта булевых ресурсов для различных размеров экранов:

```

res/values/bools.xml
<!-- Default boolean values -->
<resources>
    <bool name="large_layout">false</bool>
</resources>

```

```

res/values-large/bools.xml
<!-- Large screen boolean values -->
<resources>
    <bool name="large_layout">true</bool>
</resources>

```

Затем можно инициализировать значение `mIsLargeLayout` в течение выполнения метода операции `onCreate()`:

```

    boolean mIsLargeLayout;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        mIsLargeLayout = getResources().getBoolean(R.bool.large_layout);
    }

```

Отображение операции в качестве диалога на больших экранах

Вместо отображения диалогового окна в полноэкранном режиме на экранах малого размера можно отображать Activity в качестве диалогового окна на экранах большого размера. Выбор зависит от дизайна приложения, но отображение операции в качестве диалогового окна имеет смысл, когда приложение предназначено для использования на малых экранах, и необходимо улучшить взаимодействие с ним на планшетах, показывая кратковременные операции в качестве диалогового окна.

Для отображения операции в качестве диалогового окна только на больших экранах необходимо применить тему Theme.Holo.DialogWhenLarge к элементу манифеста <операция>:

```
<activity android:theme="@android:style/Theme.Holo.DialogWhenLarge" >
```

Подробная информация о темах операций приведена в руководстве Стили и темы.

Заккрытие диалогового окна

Когда пользователь нажимает кнопки, созданные с помощью AlertDialog.Builder, система закрывает диалоговое окно самостоятельно.

Система также закрывает диалоговое окно, когда пользователь нажимает на элемент списка в диалоговом окне, за исключением списков с переключателями или флажками. В иных случаях можно вручную закрыть диалоговое окно путем вызова dismiss() в DialogFragment.

В случае, если необходимо произвести определенные действия после закрытия диалогового окна, можно реализовать метод onDismiss() в DialogFragment.

Также можно отменить диалоговое окно. Это особое событие, возникающее, когда пользователь покинул диалоговое окно, не завершив задачу. Так происходит, когда пользователь нажимает кнопку Назад, касается экрана за областью диалогового окна, либо когда задано cancel() в Dialog (например, в качестве отклика на нажатие кнопки «Отмена» в диалоговом окне).

Как показано в примере выше, можно ответить на событие отмены с помощью onCancel() в классе DialogFragment.

Примечание: Система вызывает onDismiss() при каждом событии, которое вызывается методом обратного вызова onCancel(). Тем не менее, при вызове Dialog.dismiss() или DialogFragment.dismiss(), система вызывает onDismiss(), а не onCancel(). Поэтому в общих случаях вызов dismiss() производится при нажатии пользователем положительной кнопки в диалоговом окне, а после диалоговое окно закрывается.

ЛЕКЦИЯ №42-№43. МНОГОПОТОЧНЫЕ ПРИЛОЖЕНИЯ В ANDROID.

Многопоточность — это принцип построения программы, при котором несколько блоков могут выполняться одновременно и не мешать друг другу. Предположим, нам необходимо, чтобы какие-то операции выполнялись

одновременно.

Например, у нас существует два объекта, выполняющие конкретные операции независимо друг от друга. Но они используют один и тот же метод, который добавляет к статической переменной число, передаваемое методу в параметре. Однако если эти операции будут выполняться одновременно, то непонятно будет, что запишется в этой переменной, ведь мы одновременно прибавляем к исходному значению переменной два разных числа.

Возникнет странная ситуация. Для того чтобы такого не происходило, для этого метода необходимо установить так называемый признак блокировки. Если какой-либо объект начинает выполнять метод, то происходит блокировка этого метода, и никто больше не может его выполнить, пока данный объект не закончит работу с ним. Все другие объекты, которые должны выполнить этот метод, будут находиться в состоянии ожидания.

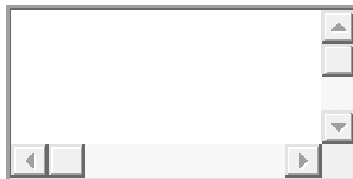
Создание потоков на основе класса Thread

Чтобы создать свой поток, необходимо создать дочерний класс класса Thread. Давайте создадим такой класс и затем разберем весь текст программы (листинг 5.7).

Листинг

5.7.

Создание потока на основе класса Thread



```
public class MyThread extends Thread{
    private int seconds;

    public MyThread(int seconds) {
        this.seconds = seconds;
    }

    public void run() {
        try {
            for (int i = 0; i < this.seconds; i++) {
                Thread.sleep(1000);
                System.out.println("Идет секунда: " + i);
            }
        } catch (InterruptedException e) {
            System.out.println("У нас проблемы с потоком");
        }
    }

    public static void main(String[] args) {
        MyThread myThread = new MyThread(5);
    }
}
```

```

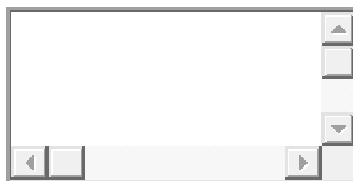
        myThread.start();
        new MyThread(15);
    }
}

```

Вначале мы создаем дочерний класс для класса Thread, затем — конструктор с параметром, который установит количество секунд выполнения потока. После этого мы определяем метод run, который начинает действовать при запуске потока. Чтобы его вызвать, необходимо вызвать метод start () — «стартовать поток».

У класса Thread есть метод sleep, который позволяет приостановить выполнение потока на заданное количество миллисекунд (внимание: отсчет идет в миллисекундах). Чтобы использовать данный метод, необходимо поместить его в блок try, а блок catch сделать обрабатывающим исключение InterruptedException. Затем мы запускаем два потока, которые будут выполняться одновременно. Чтобы их различить, можно переделать код так (листинг 5.8).

Листинг 5.8.
Создание двух потоков, которые будут выполняться в классе Thread одновременно



```

public class MyThread extends Thread{
    private int seconds;
    private static int numbers = 0;
    private int number;

    public static int setNumber() {
        return ++numbers;
    }

    public MyThread(int seconds) {
        this.seconds = seconds;
        number = MyThread.setNumber();
    }

    public void run() {
        try {
            for (int i = 0; i < this.seconds; i++) {
                Thread.sleep(1000);
            }
        } catch (InterruptedException e) {
            // Handle exception
        }
    }
}

```



```

        System.out.println("Идет секунда: " + i + " выполнения
потока под номером " + this.number);
    }
    } catch (InterruptedException e) {
        System.out.println("У нас проблемы с потоком");
    }
}

public static void main(String[] args) {
    MyThread myThread = new MyThread(5);
    myThread.start();
    new MyThread(15).start();
}
}

```

Теперь мы каждому потоку присвоили свой уникальный номер.

Использование интерфейса Runnable

Можно создавать классы потоков, реализуя интерфейс Runnable. Это бывает порой значительно удобнее. Например, код из предыдущего примера можно переписать так (листинг 5.9).

Листинг

5.9.

Создание классов потоков с использованием интерфейса Runnable

```

public class MyThread implements Runnable
{
    private int seconds;
    private static int numbers = 0;
    private int number;
    public static int setNumber ()
    {
        return ++numbers;
    }
    public MyThread (int seconds)
    {
        this.seconds = seconds;
        number = MyThread.setNumber();
    }
    public void run ()
    {
        try
        {
            for (int i = 0; i < this.seconds; i++)
            {
                Thread.sleep (1000);
                System.out.println ("Идет секунда: " + i + " выполнения потока
                под номером " + this.number);
            }
        }
        catch (InterruptedException e)
        {
            System.out.println ("У нас проблемы с потоком");
        }
    }
    public static void main (String args[])
    {
        Runnable thread1 = new MyThread (5);
        new Thread(thread1).start();
        new Thread(new MyThread (15)).start();
    }
}

```

В некоторых случаях так представляется удобней: обычно у подобного создания потоков более короткая запись. И в том и в другом случае потоку можно присвоить имя. Обычно оно задается в конструкторе в качестве последнего параметра.

Windows Phone

Начну с обзора того, как многопоточность в целом работает в XAML-приложениях.

Проще говоря, поток можно считать меньшей единицей выполнения приложения. Каждому приложению принадлежит минимум один поток, который называется основным (main thread). Этот поток запускается операционной системой, когда вызывается метод main приложения; это происходит при старте приложения. Заметьте, что более-менее одно и то же происходит на всех поддерживаемых платформах — как в WPF, работающей на мощных компьютерах, так и на устройствах под управлением Windows Phone с ограниченными вычислительными ресурсами.

При вызове какого-либо метода операция добавляется в очередь. Каждая операция выполняется последовательно, в соответствии с порядком, в котором она была добавлена в очередь (хотя на порядок выполнения операций можно повлиять, назначив им приоритеты). Объект, отвечающий за управление очередью, называется диспетчером потока (thread's dispatcher). Этот объект

является экземпляром класса Dispatcher в WPF, Silverlight и Windows Phone. В Windows 8 объект диспетчера называется CoreDispatcher и использует несколько иной API.

При необходимости приложение может запускать новые потоки — как явным образом в коде, так и неявным, с помощью некоторых библиотек или ОС. В основном цель запуска нового потока — выполнение какой-либо операции (или ожидание результата этой операции) без блокирования остальной части приложения. Это может делаться для операции, требующей интенсивных вычислений, ввода-вывода и т. д. Многопоточность современных приложений постоянно расширяется из-за роста требований к качественной UX. Поскольку приложения становятся более сложными, увеличивается количество запускаемых ими потоков. Хороший пример такой тенденции — инфраструктура Windows Runtime, применяемая в приложениях Windows Store. В этих современных клиентских приложениях очень распространены асинхронные операции (т. е. операции, выполняемые фоновыми потоками). Так, обращение к каждому файлу в Windows 8 теперь является асинхронной операцией. Вот как в WPF считывается файл (синхронно):

```
C#Копировать
public string ReadFile(FileInfo file)
{
    using (var reader = new StreamReader(file.FullName))
    {
        return reader.ReadToEnd();
    }
}
```

А это эквивалентная асинхронная операция в Windows 8:

```
C#Копировать
public async Task<string> ReadFile(IStorageFile file)
{
    var content = await FileIO.ReadTextAsync(file);
    return content;
}
```

Обратите внимание на присутствие ключевых слов await и async в версии для Windows 8. Они предназначены для того, чтобы избежать использования обратных вызовов в асинхронных операциях и чтобы код было легче читать. Здесь они нужны, поскольку файловая операция асинхронная. В WPF-версии, напротив, эта операция синхронная, что создает риск блокировки основного потока, если считываемый файл окажется длинным. Это может вызвать сделать анимации прерывистыми или привести к тому, что UI перестанет обновляться, т. е. ухудшить UX.

Аналогично длительные операции в вашем приложении должны выноситься в фоновые потоки, если есть риск того, что они будут мешать обновлению UI. Например, в WPF, Silverlight и Windows Phone код на рис. 1 инициирует фоновую операцию, которая выполняется в длительном

цикле. В каждом цикле поток ненадолго переводится в спящее состояние, чтобы дать время другим потокам обработать свои операции.

Рис. 1. Асинхронная операция в Microsoft .NET Framework

C#Копировать

```
public void DoSomethingAsynchronous()
{
    var loopIndex = 0;
    ThreadPool.QueueUserWorkItem(
        o =>
        {
            // Это фоновая операция!
            while (_condition)
            {
                // Здесь что-то делаем
                // ...
                // Засыпаем ненадолго
                Thread.Sleep(500);
            }
        });
}
```

Взаимодействие потоков

Когда одному из потоков нужно взаимодействовать с другим, следует соблюдать некоторые меры предосторожности. Например, я модифицирую код с рис. 1 так, чтобы он отображал сообщение о состоянии в каждом цикле. Для этого я просто добавлю строку кода в цикл while, которая устанавливает свойство Text элемента управления StatusTextBlock, находящегося в XAML:

C#Копировать

```
while (_condition)
{
    // Здесь что-то делаем
    // Уведомляем пользователя
    StatusTextBlock.Text = string.Format(
        "Loop # {0}", loopIndex++);
    // Ненадолго засыпаем
    Thread.Sleep(500);
}
```

Приложение SimpleMultiThreading в коде, сопутствующем этой статье, демонстрирует этот пример. Если вы запускаете его, используя кнопку Start (crashes the app), то в приложении действительно происходит крах. Что же случилось? Когда объект создается, его владельцем становится тот поток, в котором был вызван метод конструктора. В случае UI-элементов объекты создаются анализатором XAML (XAML parser) при загрузке XAML-документа. Все это происходит в основном потоке. В итоге все UI-элементы принадлежат основному потоку, который также зачастую называют UI-потоком. Когда фоновый поток в предыдущем коде пытается модифицировать

свойство Text элемента StatusTextBlock, обнаруживается недопустимое обращение между потоками. Как следствие, генерируется исключение. Это можно увидеть, выполняя данный код в отладчике. На рис. 2 показан диалог исключения. Обратите внимание на сообщение «Additional information», которое указывает корень проблемы.

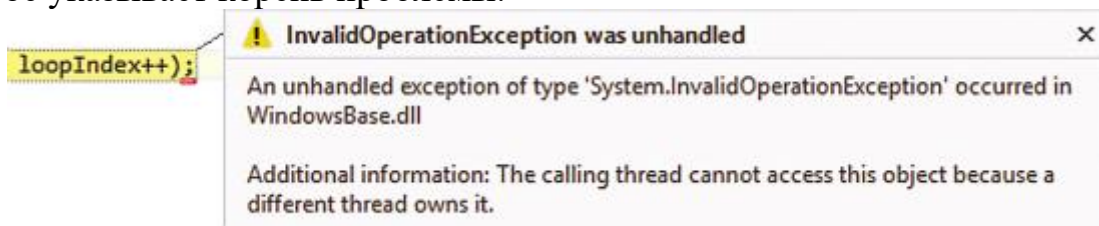


Рис. 2. Диалог исключения, вызванного недопустимым обращением из одного потока в другой

Чтобы этот код работал, фоновый поток должен ставить операция в очередь в основном потоке через его диспетчер. К счастью, каждый FrameworkElement также является DispatcherObject, как иллюстрирует иерархия .NET-классов на рис. 3. Каждый DispatcherObject предоставляет свойство Dispatcher, которое дает доступ к своему диспетчеру. Таким образом, данный код можно модифицировать, как показано на рис. 4.

▲ Inheritance Hierarchy

```
System.Object
System.Windows.Threading.DispatcherObject
System.Windows.DependencyObject
System.Windows.Media.Visual
System.Windows.UIElement
System.Windows.FrameworkElement
System.Windows.Controls.Control
System.Windows.Controls.ContentControl
System.Windows.Window
System.Windows.Controls.Ribbon.RibbonWindow
System.Windows.Navigation.NavigationWindow
```

Рис. 3. Иерархия классов

Рис. 4. Диспетчеризация вызова UI-потоку

C#Копировать

```
while (_condition)
```

```
{
```

```
    // Здесь что-то делаем
```

```
    Dispatcher.BeginInvoke(
```

```
        (Action)(() =>
```

```
        {
```

```
            // Уведомляем пользователя
```

```
            StatusTextBlock.Text = string.Format(
```

```
                "Loop # {0}", loopIndex++);
```

```
        });
```

```
    // Ненадолго засыпаем
```

```
    Thread.Sleep(500);
```

```
}
```

Диспетчеризация в MVVM-приложениях

Когда фоновая операция запускается из ViewModel, ситуация несколько меняется. Обычно ViewModel не наследует от DispatcherObject. ViewModel — это РОСО-объекты (Plain Old CLR Objects), которые реализуют интерфейс INotifyPropertyChanged. Например, на рис. 5 показан ViewModel, производный от класса ViewModelBase из MVVM Light. В истинном стиле MVVM я добавляю наблюдаемое свойство Status, генерирующее событие PropertyChanged. Затем из кода фонового потока я пытаюсь записать в это свойство информационное сообщение.

Рис. 5. Обновление связанного свойства в ViewModel

C#Копировать

```
public class MainViewModel : ViewModelBase
{
    public const string StatusPropertyName = "Status";
    private bool _condition = true;
    private RelayCommand _startSuccessCommand;
    private string _status;
    public RelayCommand StartSuccessCommand
    {
        get
        {
            return _startSuccessCommand
            ?? (_startSuccessCommand = new RelayCommand(
                () =>
                {
                    var loopIndex = 0;
                    ThreadPool.QueueUserWorkItem(
                        o =>
                        {
                            // Это фоновая операция!
                            while (_condition)
                            {
                                // Здесь что-то делаем
                                DispatcherHelper.CheckBeginInvokeOnUI(
                                    () =>
                                    {
                                        // Направляем обратно в основной поток
                                        Status = string.Format(
                                            "Loop # {0}", loopIndex++);
                                    });
                                // Ненадолго засыпаем
                                Thread.Sleep(500);
                            }
                        });
                }));
        }
    }
}
```

```

    }
    public string Status
    {
        get
        {
            return _status;
        }
        set
        {
            Set(StatusPropertyName, ref _status, value);
        }
    }
}

```

Если запустить этот код в Windows Phone или Silverlight, он прекрасно работает, пока я не попытаюсь связать свойство Status с TextBlock в XAML UI. Выполнение операции вновь обрушит приложение. Как и раньше, едва фоновый поток попытается обратиться к элементу, принадлежащему другому потоку, возникнет исключение. Это происходит, даже если доступ осуществляется через механизм связывания с данными.

Заметьте, что в WPF дело обстоит иначе и код, приведенный на рис. 5, работает, даже если свойство Status связано с TextBlock. Это вызвано тем, что WPF автоматически направляет событие PropertyChanged в основной поток в отличие от всех остальных инфраструктур XAML. Во всех других инфраструктурах требуется специальное решение по диспетчеризации. По сути, требуется система, которая при необходимости диспетчеризует вызов. Чтобы код ViewModel можно было использовать и в WPF, и в других инфраструктурах и не заботиться о диспетчеризации, вы должны располагать объектом, который делал бы это автоматически.

Поскольку ViewModel является POCO, у него нет доступа к свойству Dispatcher, поэтому мне нужен другой способ обращения к основному потоку и постановки операции в очередь. Для этого предназначен компонент DispatcherHelper в MVVM Light. Фактически этот класс хранит Dispatcher основного потока в статическом свойстве и предоставляет несколько вспомогательных методов для доступа к нему удобным и единым способом. Чтобы класс мог работать, его нужно инициализировать в основном потоке. В идеале, это должно происходить на самом раннем этапе жизненного цикла приложения, чтобы его функциональность была доступна с момента запуска приложения. Как правило, DispatcherHelper в приложении MVVM Light инициализируется в App.xaml.cs — файле, определяющем стартовый класс приложения. В Windows Phone вы вызываете DispatcherHelper.Initialize в методе InitializePhoneApplication сразу после создания основной рамки окна приложения. В WPF этот класс инициализируется в конструкторе App. В Windows 8 вы вызываете метод Initialize в OnLaunched сразу после активизации окна.

По окончании вызова метода `DispatcherHelper.Initialize` свойство `UIDispatcher` класса `DispatcherHelper` содержит ссылку на диспетчер основного потока. Напрямую это свойство используют относительно редко, но при необходимости такое возможно. Однако лучше задействовать метод `CheckBeginInvokeOnUi`. Этот метод принимает делегат как параметр. Обычно вы используете лямбда-выражение, показанное на рис. 6, но вместо него можно подставить именованный метод.

Рис. 6. Использование `DispatcherHelper` для предотвращения краха

C#Копировать

```
while (_condition)
{
    // Здесь что-то делаем
    DispatcherHelper.CheckBeginInvokeOnUI(
        () =>
        {
            // Направляем обратно в основной поток
            Status = string.Format("Loop # {0}", loopIndex++);
        });
    // Ненадолго засыпаем
    Thread.Sleep(500);
}
```

Как и предполагает название, этот метод сначала выполняет проверку. Если код, вызвавший этот метод, уже выполняется в основном потоке, никакой диспетчеризации не требуется. В таком случае делегат выполняется немедленно — прямо в основном потоке. Однако, если код, вызвавший этот метод, находится в фоновом потоке, происходит диспетчеризация.

Поскольку метод выполняет проверку перед диспетчеризацией, вызывающий может быть уверен, что код всегда будет использовать оптимальный вызов. Это особенно полезно, когда вы пишете кросс-платформенный код, где многопоточность может работать с небольшими различиями на разных платформах. В данном случае код `ViewModel` на рис. 6 можно использовать где угодно без модификации строки, в которой задается свойство `Status`.

Кроме того, `DispatcherHelper` абстрагирует различия в API диспетчера между платформами XAML. В Windows 8 основные члены `CoreDispatcher` — это метод `RunAsync` и свойство `HasThreadAccess`. Однако в других инфраструктурах XAML используются методы `BeginInvoke` и `CheckAccess` соответственно. Применяя `DispatcherHelper`, вам не придется заботиться об этих различиях, и сделать код общим будет проще.

Диспетчеризация на практике: датчики

Я проиллюстрирую применение `DispatcherHelper`, создав приложение Windows Phone для датчика компаса.

В сопутствующем этой статье коде есть набросок приложения под названием «CompassSample – Start». Откройте это приложение в Visual Studio и вы увидите, что доступ из `MainViewModel` к датчику компаса

инкапсулирован в сервисе `SensorService`, который является реализацией интерфейса `ISensorService`. Эти два элемента вы найдете в папке `Model`.

`MainViewModel` получает ссылку на `ISensorService` в своем конструкторе и регистрируется на каждое изменение в показаниях компаса, вызывая метод `SensorService.RegisterForHeading`. Этот метод требует обратного вызова, который будет выполняться всякий раз, когда датчик сообщает об изменении в направлении устройства под управлением Windows Phone. В `MainViewModel` замените конструктор по умолчанию следующим кодом:

```
C#Копировать
sensorService.RegisterForHeading(
    heading =>
    {
        Heading = string.Format("{0:N1}°", heading);
        Debug.WriteLine(Heading);
    });
```

К сожалению, в эмуляторе Windows Phone нет возможности симитировать аппаратный компас. Для проверки кода придется запустить приложение на физическом устройстве. Подключите это устройство и запустите код в режиме отладки, нажав F5. Наблюдайте за консольным окном `Output` в `Visual Studio`. Вы увидите вывод `Compass`. Если вы подвигаете устройство, то сможете найти север и наблюдать, как будет постоянно обновляться значение.

Затем я свяжу `TextBlock` в XAML со свойством `Heading` в `MainViewModel`. Откройте `MainPage.xaml` и найдите `TextBlock` в `ContentPanel`. Замените «`Nothing yet`» в свойстве `Text` на «`{Binding Heading}`». Если вы снова запустите приложение в режиме отладки, приложение рухнет с сообщением об ошибке, похожим на то, что вы видели ранее. И вновь это исключение из-за некорректного взаимодействия между потоками.

Ошибка возникает потому, что датчик компаса обрабатывается фоновым потоком. Когда запускается обратный вызов, он также выполняется в фоновом потоке, как и аксессор `set` свойства `Heading`. Поскольку `TextBlock` принадлежит основному потоку, генерируется исключение. Здесь тоже нужно создать «безопасную зону» — позаботиться о диспетчеризации операций в основной поток. Для этого откройте класс `SensorService`. Событие `CurrentValueChanged` обрабатывается методом `CompassCurrentValueChanged`; как раз в нем выполняется метод обратного вызова. Замените этот код показанным ниже, где используется `DispatcherHelper`:

```
C#Копировать
void CompassCurrentValueChanged(object sender,
    SensorReadingEventArgs<CompassReading> e)
{
    if (_orientationCallback != null)
    {
        DispatcherHelper.CheckBeginInvokeOnUI(
            () => _orientationCallback(e.SensorReading.TrueHeading));
    }
}
```

```
}  
}
```

Теперь нужно инициализировать `DispatcherHelper`. Для этого откройте `App.xaml.cs` и найдите метод `InitializePhoneApplication`. В самый конец этого метода добавьте выражение `DispatcherHelper.Initialize();`. Теперь запуск кода даст ожидаемый результат, корректно отображая направление устройства `Windows Phone`.

Заметьте, что в `Windows Phone` не все датчики генерируют свои события в фоновом потоке. Например, датчик `GeoCoordinateWatcher`, используемый для отслеживания геопозиционирования устройства, возвращает показания в основной поток для вашего удобства. Используя `DispatcherHelper`, вы не должны заботиться об этом и можете всегда единообразно инициировать обратный вызов основного потока.

ЛЕКЦИЯ №44-№45. ПРОЦЕССЫ В ANDROID. ОБЪЕКТЫ ACTIVITY.

Когда компонент приложения запускается при отсутствии других работающих компонентов, система `Android` запускает новый процесс `Linux` для приложения с одним потоком выполнения. По умолчанию все компоненты одного приложения работают в одном процессе и потоке (называется «главным потоком»). Если компонент приложения запускается при наличии процесса для этого приложения (так как существует другой компонент из приложения), тогда компонент запускается в этом процессе и использует тот же поток выполнения. Однако можно организовать выполнение других компонентов приложения в отдельных процессах и создавать дополнительный поток для любого процесса.

В этом документе обсуждается работа процессов и потоков в приложении `Android`.

Процессы

По умолчанию все компоненты одного приложения работают в одном процессе, и большинство приложений не должно менять это поведение. Однако, если необходимо контролировать, к какому процессу принадлежат определенный компонент, можно сделать это в файле манифеста.

Запись манифеста для каждого типа элементов компонента — `<activity>`, `<service>`, `<receiver>` и `<provider>` — поддерживает атрибут `android:process`, позволяющий задавать процесс, в котором следует выполнять этот компонент. Можно установить этот атрибут так, чтобы каждый компонент выполнялся в собственном процессе, или так, чтобы только некоторые компоненты совместно использовали один процесс. Можно также настроить процесс `android:process` так, чтобы компоненты разных приложений выполнялись в одном процессе, при условии что приложения совместно используют один идентификатор пользователя `Linux` и выполняют вход с одним сертификатом.

Элемент <application> также поддерживает атрибут `android:process`, позволяющий задать значение по умолчанию, которое применяется ко всем компонентам.

Android может остановить процесс в некоторой точке, когда не хватает памяти и она необходима другим процессам, которые обслуживают пользователя в данный момент. Работа компонентов приложения, работающих в этом процессе, последовательно останавливается. Процесс для этих компонентов запускается повторно, когда для них появляется работа.

Принимая решение о прерывании процессов, система Android взвешивает их относительную важность для пользователя. Например, более вероятно выключение процессов, содержащих действия, которые не отображаются на экране, по сравнению с процессом, содержащим видимые действия. Следовательно, решение о прерывании процесса зависит от состояния компонентов, работающих в этом процессе. Ниже обсуждаются правила, на основании которых принимается решение о выборе прерываемых процессов.

Жизненный цикл процесса

Система Android пытается сохранять процесс приложения как можно дольше, но в конечном счете вынуждена удалять старые процессы, чтобы восстановить память для новых или более важных процессов. Чтобы определить, какие процессы сохранить, а какие удалить, система помещает каждый процесс в «иерархию важности» на основе компонентов, выполняющихся в процессе, и состояния этих компонентов. Процессы с самым низким уровнем важности исключаются в первую очередь, затем исключаются процессы следующего уровня важности и т. д., насколько это необходимо для восстановления ресурсов системы.

В иерархии важности предусмотрено пять уровней. В следующем списке представлены различные типы процессов в порядке важности (первый процесс является наиболее важным и удаляется в последнюю очередь):

Процесс переднего плана

Процесс, необходимый для текущей деятельности пользователя. Процесс считается процессом переднего плана, если выполняется любое из следующих условий:

- Он содержит действие Activity, с которым взаимодействует пользователь (вызван метод Activity onResume()).

- Он содержит службу Service, связанную с действием, с которым взаимодействует пользователь.

- Он содержит службу Service, которая выполняется "на переднем плане", — службу, которая называется startForeground().

- Он содержит службу Service, которая выполняет один из обратных вызовов жизненного цикла (onCreate(), onStart() или onDestroy()).

- Он содержит ресивер BroadcastReceiver, который выполняет метод onReceive().

Обычно одновременно работает лишь несколько процессов переднего плана. Они уничтожаются только в крайнем случае, если памяти остается так

мало, что они не могут продолжать совместную работу. Обычно в этот момент устройство достигло состояния разбиения памяти на страницы, поэтому для того, чтобы пользовательский интерфейс откликался на действия пользователя, необходимо удаление некоторых процессов на переднем плане.

Видимые процессы

Процессы, которые не содержат компонентов переднего плана, но могут влиять на отображение на экране. Процесс считается видимым, если выполняется любое из следующих условий:

Он содержит действие Activity, которое не находится на переднем плане, но видно пользователю (вызван метод onPause()). Например, это может происходить, если действие на переднем плане запустило диалоговое окно, которое позволяет видеть предыдущее действие позади него.

Он содержит службу Service, связанную с видимым действием или действием переднего плана.

Видимый процесс считается исключительно важным, его следует удалять только в случае, если требуется сохранить работу всех процессов переднего плана.

Служебный процесс

Процесс, который выполняет службу, запущенную с помощью метода startService(), и не попадает ни в одну из двух категорий более высокого уровня. Хотя служебные процессы не связаны непосредственно с тем, что видит пользователь, они обычно выполняют важные для пользователя действия (например, воспроизводят музыку в фоновом режиме или загружают данные в сеть), поэтому система сохраняет их выполнение, если памяти достаточно для их работы наряду со всеми видимыми процессами и процессами переднего плана.

Фоновый процесс

Процесс, содержащий действия, которые не видны пользователю в настоящее время (вызван метод onStop() действия). Эти процессы не оказывают непосредственного воздействия на работу пользователя, и система может удалить их в любой момент, чтобы освободить память для процессов переднего плана, видимых или служебных процессов. Обычно выполняется множество фоновых процессов, поэтому они хранятся в списке LRU (недавно использованные), чтобы процессы, содержащие самые недавние действия, которые видел пользователь, удалялись в последнюю очередь. Если для действия правильно реализованы методы жизненного цикла, и действие сохраняет текущее состояние, удаление процесса этого действия не оказывает видимого воздействия на работу пользователя, так как когда пользователь возвращается к этому действию, оно восстанавливает все элементы своего видимого состояния. Информацию о сохранении и восстановлении состояния см. в документе Действия.

Пустой процесс

Процесс, не содержащий никаких компонентов активного приложения. Единственная причина сохранять процесс такого типа — это кэширование, которое улучшает время следующего запуска компонента в этом процессе.

Система часто удаляет эти процессы для равномерного распределения всех системных ресурсов между кэшем процесса и кэшем базового ядра.

Система Android относит процесс к максимально высокому уровню на основе важности компонентов, активных в процессе в текущее время. Например, если процесс содержит служебное и видимое действие, процесс считается видимым, а не служебным процессом.

Кроме того, уровень процесса может быть повышен, поскольку имеются другие процессы, зависящие от него. Например, процесс, обслуживающий другой процесс, не может иметь уровень ниже уровня обслуживаемого процесса. Например, если поставщик контента в процессе А обслуживает клиента в процессе В или служебный процесс А связан с компонентом в процессе В, процесс А всегда считается не менее важным, чем процесс В.

Так как процесс, выполняющий службу, оценивается выше процесса с фоновыми действиями, действие, запускающее долговременную операцию, может запустить службу для этой операции, а не просто создать рабочий поток, особенно в случае, если операция продлится дольше действия. Например, действие, которое загружает изображение на веб-сайт, должно запустить службу для выполнения загрузки, так что загрузка может продолжаться в фоновом режиме даже после выхода пользователя из действия. Использование службы гарантирует, что операция будет иметь приоритет не ниже «служебного процесса», независимо от того, что происходит с действием. По этой же причине ресиверы должны использовать службы, а не просто ставить в поток операции, требующие много времени для выполнения.

Потоки

При запуске приложения система создает поток выполнения для приложения, который называется «главным». Этот поток очень важен, так как он отвечает за диспетчеризацию событий на виджеты соответствующего интерфейса пользователя, включая события графического представления. Он также является потоком, в котором приложение взаимодействует с компонентами из набора инструментов пользовательского интерфейса Android (компонентами из пакетов android.widget и android.view). По существу, главный поток — это то, что иногда называют потоком пользовательского интерфейса.

Система не создает отдельного потока для каждого экземпляра компонента. Все компоненты, которые выполняются в одном процессе, создают экземпляры в потоке пользовательского интерфейса, и системные вызовы каждого компонента отправляются из этого потока. Поэтому методы, которые отвечают на системные обратные вызовы (такие как метод onKeyDown() для сообщения о действиях пользователя или метод обратного вызова жизненного цикла), всегда выполняются в потоке пользовательского интерфейса процесса.

Например, когда пользователь нажимает кнопку на экране, поток пользовательского интерфейса вашего приложения отправляет событие нажатия в виджет, который, в свою очередь, устанавливает кнопку в нажатое

состояние и отправляет запрос на аннулирование в очередь событий. Поток пользовательского интерфейса исключает запрос из очереди и уведомляет виджет, что он должен отобразиться повторно.

Когда приложение выполняет интенсивную работу в ответ на действия пользователя, эта одиночная модель потока может показывать плохую производительность, если приложение реализовано неправильно. То есть, если все происходит в потоке пользовательского интерфейса, выполнение долговременных операций, таких как сетевой доступ или запросы к базе данных, будет блокировать весь пользовательский интерфейс. Когда поток заблокирован, не могут обрабатываться никакие события, включая события изменения отображения. С точки зрения пользователя приложение выглядит зависшим. Хуже того, если поток пользовательского интерфейса заблокирован более нескольких секунд (в настоящее время около 5 секунд), отображается печально известное диалоговое окно «приложение не отвечает». После этого недовольный пользователь может выйти из вашего приложения и удалить его.

Кроме того, набор инструментов пользовательского интерфейса Android не является потокобезопасным. Поэтому, вы не должны работать с пользовательским интерфейсом из рабочего потока. Манипуляции с пользовательским интерфейсом необходимо выполнять из потока пользовательского интерфейса. Таким образом, существует только два правила однопоточной модели Android:

Не блокируйте поток пользовательского интерфейса

Не обращайтесь к набору инструментов пользовательского интерфейса Android снаружи потока пользовательского интерфейса

Рабочие потоки

Вследствие описанной выше однопоточной модели для динамичности пользовательского интерфейса ваших приложений очень важно не блокировать поток пользовательского интерфейса. Если требуется выполнять операции, занимающие некоторое время, обязательно выполняйте их в отдельных потоках («фоновых» или «рабочих» потоках).

Например, ниже приведен код контроля нажатий, который загружает изображение из отдельного потока и отображает их в виджете ImageView:

```
public void onClick(View v) {  
    new Thread(new Runnable() {  
        public void run() {  
            Bitmap b = loadImageFromNetwork("http://example.com/image.png");  
            mImageView.setImageBitmap(b);  
        }  
    }).start();  
}
```

На первый взгляд, он должен работать хорошо, так как он создает новый поток для обработки сетевой операции. Однако, он нарушает второе правило однопоточной модели: не обращайтесь к набору инструментов пользовательского интерфейса Android снаружи потока пользовательского

интерфейса — этот пример изменяет ImageView из рабочего потока, а не из потока пользовательского интерфейса. Это может привести к неопределенному и непредвиденному поведению, отследить которое будет трудно.

Для устранения этой проблемы Android предлагает несколько путей доступа к потоку пользовательского интерфейса из других потоков. Ниже приведен список полезных методов:

Activity.runOnUiThread(Runnable)

View.post(Runnable)

View.postDelayed(Runnable, long)

Например, можно исправить приведенный выше код с помощью метода View.post(Runnable):

```
public void onClick(View v) {
    new Thread(new Runnable() {
        public void run() {
            final Bitmap bitmap =
loadImageFromNetwork("http://example.com/image.png");
            mImageView.post(new Runnable() {
                public void run() {
                    mImageView.setImageBitmap(bitmap);
                }
            });
        }
    }).start();
}
```

Теперь реализация является потокобезопасной: сетевая операция выполняется из отдельного потока, тогда как ImageView работает из потока пользовательского интерфейса.

Однако по мере роста сложности, код такого типа может становиться запутанным и сложным для поддержания. Чтобы обрабатывать более сложные взаимодействия с рабочим потоком, можно использовать метод Handler в рабочем потоке для обработки сообщений, поступающих из потока пользовательского интерфейса. Вероятно, самым лучшим решением является расширение класса AsyncTask, которое упрощает выполнение заданий рабочего потока, которые должны взаимодействовать с пользовательским интерфейсом.

Использование AsyncTask

Метод AsyncTask позволяет выполнять асинхронную работу в пользовательском интерфейсе. Он выполняет операции блокирования в рабочем потоке и затем публикует результаты в потоке пользовательского интерфейса без необходимости самостоятельно обрабатывать потоки и/или обработчики.

Для использования этого метода необходимо создать подкласс AsyncTask и реализовать метод обратного вызова doInBackground(),

который работает в пуле фоновых потоков. Чтобы обновить пользовательский интерфейс, следует реализовать метод `onPostExecute()`, который доставляет результат из `doInBackground()` и работает в потоке пользовательского интерфейса, так что вы можете безопасно обновлять пользовательский интерфейс. Задача выполняется через вызов метода `execute()` из потока пользовательского интерфейса.

Например, можно реализовать предыдущий пример с помощью метода `AsyncTask` следующим образом:

```
public void onClick(View v) {
    new DownloadImageTask().execute("http://example.com/image.png");
}

private class DownloadImageTask extends AsyncTask<String, Void, Bitmap> {
    /** The system calls this to perform work in a worker thread and
     * delivers it the parameters given to AsyncTask.execute() */
    protected Bitmap doInBackground(String... urls) {
        return loadImageFromNetwork(urls[0]);
    }

    /** The system calls this to perform work in the UI thread and delivers
     * the result from doInBackground() */
    protected void onPostExecute(Bitmap result) {
        mImageView.setImageBitmap(result);
    }
}
```

Теперь пользовательский интерфейс защищен и код стал проще, так как работа разделена на часть, которая должна выполняться в рабочем потоке, и часть, которая должна выполняться в потоке пользовательского интерфейса.

Прочитайте статью [AsyncTask](#), чтобы полностью понять использование этого класса. Здесь приведен краткий обзор его работы:

Можно указывать тип параметров, значения хода выполнения и конечное значение задания с помощью универсальных компонентов

Метод `doInBackground()` выполняется автоматически в рабочем потоке

Методы `onPreExecute()`, `onPostExecute()` и `onProgressUpdate()` запускаются в потоке пользовательского интерфейса

Значение, возвращенное методом `doInBackground()`, отправляется в метод `onPostExecute()`

Можно вызвать `publishProgress()` в любой момент в `doInBackground()` для выполнения `onProgressUpdate()` в потоке пользовательского интерфейса

Задание можно отменить в любой момент из любого потока

Предупреждение! Другая проблема, с которой вы можете столкнуться при использовании рабочего потока, состоит в непредсказуемом перезапуске действия вследствие изменения конфигурации в режиме выполнения,

(например, когда пользователь изменяет ориентацию экрана), что может разрушить рабочий поток. Чтобы увидеть, как можно сохранить задание во время одного из подобных перезапусков и как правильно отменить задание при разрушении действия, изучите исходный код примера приложения Shelves.

Потокобезопасные методы

В некоторых ситуациях реализованные методы могут вызываться из нескольких потоков и, следовательно, должны быть написаны с сохранением потокобезопасности.

В первую очередь это относится к методам, которые можно вызывать удаленно, например, к методам в связанной службе. Когда вызов метода реализуется в классе IBinder, происходящем из того же процесса, в котором выполняется IBinder, метод выполняется в потоке вызывающего метода. Однако, когда вызов происходит из другого процесса, метод выполняется в потоке, выбранном из пула потоков, которые система поддерживает в том же процессе, что и IBinder (он не выполняется в потоке пользовательского интерфейса процесса). Например, поскольку метод onBind() службы будет вызываться из потока пользовательского интерфейса процесса службы, методы, реализованные в объекте, который возвращает onBind() (например, подкласс, который реализует методы RPC), будут вызываться из потоков в пуле. Так как служба может иметь несколько клиентов, несколько потоков из пула могут одновременно использовать один и тот же метод IBinder. Поэтому методы IBinder должны быть реализованы с сохранением потокобезопасности.

Аналогичным образом поставщик контента может получать запросы данных, которые происходят из другого процесса. Хотя классы ContentResolver и ContentProvider скрывают подробности управления взаимодействием процессов, методы ContentProvider, которые отвечают на эти запросы, — методы query(), insert(), delete(), update() и getType() — вызываются из пула потоков в процессе поставщика контента, а не в процессе потока пользовательского интерфейса. Поскольку эти методы могут вызываться из любого числа потоков одновременно, они также должны быть реализованы с сохранением потокобезопасности.

Взаимодействие процессов

Система Android предлагает механизм взаимодействия процессов (IPC) с помощью удаленного вызова процедуры (RPC), при котором метод вызывается действием или другим компонентом приложения, но выполняется удаленно (в другом процессе) с возвратом всех результатов вызывающему компоненту. Это влечет разложение вызова метода и его данных до уровня, понятного операционной системе, передачу его из локального процесса и адресного пространства удаленному процессу и адресному пространству, а затем повторную сборку и восстановление вызова. После этого возвращенные значения передаются в обратном направлении. Система Android содержит все коды для выполнения этих механизмов IPC, так что вы можете

сосредоточиться на определении и реализации программного интерфейса RPC.

Для выполнения IPC приложение должно быть привязано к службе с помощью метода `bindService()`. Дополнительные сведения представлены в разделе Службы руководства для разработчиков.

ЛЕКЦИЯ №46-№47. ИСПОЛЬЗОВАНИЕ ОБЪЕКТОВ `Intent`.

Intent представляет собой объект обмена сообщениями, с помощью которого можно запросить выполнение действия у компонента другого приложения. Несмотря на то, что объекты `Intent` упрощают обмен данными между компонентами по нескольким аспектам, в основном они используются в трех ситуациях:

Для запуска операции:

Компонент Activity представляет собой один экран в приложении. Для запуска нового экземпляра компонента Activity необходимо передать объект `Intent` методу `startActivity()`. Объект `Intent` описывает операцию, которую требуется запустить, а также содержит все остальные необходимые данные.

Если после завершения операции от нее требуется получить результат, вызовите метод `startActivityForResult()`. Ваша операция получит результат в виде отдельного объекта `Intent` в обратном вызове метода `onActivityResult()` операции. Подробные сведения см. в руководстве Операции.

Для запуска службы:

Service является компонентом, который выполняет действия в фоновом режиме без пользовательского интерфейса. Службу можно запустить для выполнения однократного действия (например, чтобы загрузить файл), передав объект `Intent` методу `startService()`. Объект `Intent` описывает службу, которую требуется запустить, а также содержит все остальные необходимые данные.

Если служба сконструирована с интерфейсом клиент-сервер, к ней можно установить привязку из другого компонента, передав объект `Intent` методу `bindService()`. Подробные сведения см. в руководстве Службы.

Для рассылки широковещательных сообщений:

Широковещательное сообщение — это сообщение, которое может принять любое приложение. Система выдает различные широковещательные сообщения о системных событиях, например, когда система загружается или устройство начинает заряжаться. Для выдачи широковещательных сообщений другим приложениям необходимо передать объект `Intent` методу `sendBroadcast()`, `sendOrderedBroadcast()` или `sendStickyBroadcast()`.

Типы объектов `Intent`

Есть два типа объектов `Intent`:

Явные объекты Intent указывают компонент, который требуется запустить, по имени (полное имя класса). Явные объекты Intent обычно используются для запуска компонента из вашего собственного приложения, поскольку вам известно имя класса операции или службы, которую необходимо запустить. Например , можно запустить новую операцию в ответ на действие пользователя или запустить службу, чтобы загрузить файл в фоновом режиме.

Неявные объекты Intent не содержат имени конкретного компонента. Вместо этого они в целом объявляют действие, которое требуется выполнить, что дает возможность компоненту из другого приложения обработать этот запрос. Например, если требуется показать пользователю место на карте, то с помощью неявного объекта Intent можно запросить, чтобы это сделало другое приложение, в котором такая возможность предусмотрена.

Когда создан явный объект Intent для запуска операции или службы, система немедленно запускает компонент приложения, указанный в объекте Intent.

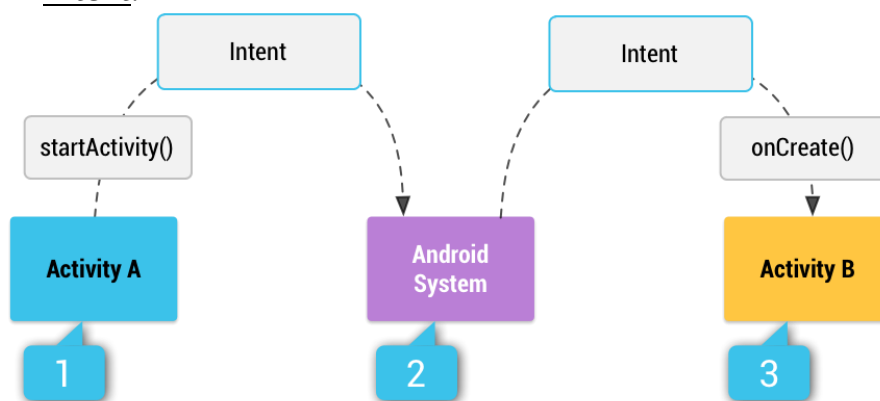


Рисунок 1. Схематическое изображение процесса передачи неявного объекта Intent по системе для запуска другой операции: [1] Операция А создает объект Intent с описанием действия и передает его методу startActivity(). [2] Система Android ищет во всех приложениях фильтры Intent, которые соответствуют данному объекту Intent. Когда приложение с подходящим фильтром найдено, [3] система запускает соответствующую операцию (Операция В), вызвав ее метод onCreate() и передав ему объект Intent.

Когда создан неявный объект Intent, система Android находит подходящий компонент путем сравнения содержимого объекта Intent с фильтрами Intent, объявленными в файлах манифеста других приложений, имеющихся на устройстве. Если объект Intent совпадает с фильтром Intent, система запускает этот компонент и передает ему объект Intent. Если подходящими оказываются несколько фильтров Intent, система выводит диалоговое окно, где пользователь может выбрать приложение для выполнения данного действия.

Фильтр Intent представляет собой выражение в файле манифеста приложения, указывающее типы объектов Intent, которые мог бы принимать компонент. Например, объявив фильтр Intent для операции, вы даете другим

приложениям возможность напрямую запускать вашу операцию с помощью некоторого объекта `Intent`. Точно так же, если вы не объявите какие-либо фильтры `Intent` для операции, то ее можно будет запустить только с помощью явного объекта `Intent`.

Внимание! В целях обеспечения безопасности приложения всегда используйте явный объект `Intent` при запуске `Service` и не объявляйте фильтры `Intent` для своих служб. Запуск служб с помощью неявных объектов `Intent` является рискованным с точки зрения безопасности, поскольку нельзя быть на абсолютно уверенным, какая служба отреагирует на такой объект `Intent`, а пользователь не может видеть, какая служба запускается. Начиная с Android 5.0 (уровень API 21) система вызывает исключение при вызове метода `bindService()` с помощью неявного объекта `Intent`.

Создание объекта `Intent`

Объект `Intent` содержит информацию, на основании которой система Android определяет, какой компонент требуется запустить (например, точное имя компонента или категорию компонентов, которые должны получить этот объект `Intent`), а также сведения, которые необходимы компоненту-получателю, чтобы надлежащим образом выполнить действие (а именно — выполняемое действие и данные, с которыми его требуется выполнить).

Основные сведения, содержащиеся в объекте `Intent`:

Имя компонента

Имя компонента, который требуется запустить.

Эта информация является необязательной, но именно она и делает объект `Intent` явным. Ее наличие означает, что объект `Intent` следует доставить только компоненту приложения, определенному по имени. При отсутствии имени компонента объект `Intent` является неявным, а система определяет, какой компонент получит этот объект `Intent` по другим сведениям, которые в нем содержатся (например, по действию, данным и категории — см. описание далее). Поэтому, если вам требуется запустить определенный компонент из своего приложения, следует указать его имя.

Примечание. При запуске `Service` следует всегда указывать имя компонента. В противном случае вы не сможете быть на абсолютно уверенным в том, какая служба отреагирует на объект `Intent`, а пользователь не может видеть, какая служба запускается.

Это поле объекта `Intent` представляет собой объект `ComponentName`, который можно указать с помощью полного имени класса целевого компонента, включая имя пакета приложения. Например, `com.example.ExampleActivity`. Задать имя компонента можно с помощью метода `setComponent()`, `setClass()`, `setClassName()` или конструктора `Intent`.

Действие

Строка, определяющая стандартное действие, которое требуется выполнить (например, `view` (просмотр) или `pick` (выбор)).

При выдаче объектов `Intent` с широковещательными сообщениями это действие, которое произошло и о котором сообщается. Действие в

значительной степени определяет, каким образом структурирована остальная часть объекта `Intent`,—в частности, что именно содержится в разделе данных и дополнительных данных.

Для использования объектами `Intent` в пределах своего приложения (либо для использования другими приложениями, чтобы вызывать компоненты из вашего приложения) можно указать собственные действия. Обычно же следует использовать константы действий, определенные классом `Intent` или другими классами платформы. Вот несколько стандартных действий для запуска операции:

ACTION_VIEW

Используйте это действие в объекте `Intent` с методом `startActivity()`, когда имеется определенная информация, которую операция может показать пользователю, например, фотография в приложении галереи или адрес для просмотра в картографическом приложении.

ACTION_SEND

Его еще называют объектом `Intent` "share" (намерение предоставить общий доступ). Это действие следует использовать в объекте `Intent` с методом `startActivity()`, при наличии определенных данных, доступ к которым пользователь может предоставить через другое приложение, например приложение для работы с электронной почтой или социальными сетями.

Другие константы, определяющие стандартные действия, см. в справочнике по классу `Intent`. Другие действия определяются в других частях платформы `Android`. Например, в `Settings` определяются действия, открывающие ряд экранов приложения настройки системы.

Действие можно указать для объекта `Intent` с методом `setAction()` или конструктором `Intent`.

Если вы определяете собственные действия, обязательно используйте в качестве их префикса имя пакета вашего приложения. Например:

```
static final String ACTION_TIMETRAVEL =  
"com.example.action.TIMETRAVEL";
```

Данные

`URI` (объект `Uri`), ссылающийся на данные, с которыми будет выполняться действие и/или тип `MIME` этих данных. Тип передаваемых данных обычно определяется действием объекта `Intent`. Например, если действием является `ACTION_EDIT`, в данных должен содержаться `URI` документа, который требуется отредактировать.

При создании объекта `Intent`, помимо `URI`, зачастую бывает важно указать тип данных (их тип `MIME`). Например, операция, которая может выводить на экран изображения, скорее всего, не сможет воспроизвести аудиофайл, даже если и у тех, и у других данных будут одинаковые форматы `URI`. Поэтому указание типа `MIME` данных помогает системе `Android` найти наиболее подходящий компонент для получения вашего объекта `Intent`. Однако тип `MIME` иногда можно унаследовать от `URI` — в частности, когда данные представляют собой `content: URI`, который указывает, что данные

находятся на устройстве и ими управляет ContentProvider, а это дает возможность системе видеть тип MIME данных.

Чтобы задать только URI данных, вызовите setData(). Чтобы задать только тип MIME, вызовите setType(). При необходимости оба этих параметра можно в явном виде задать с помощью setDataAndType().

Внимание! Если требуется задать и URI, и тип MIME, не вызывайте setData() и setType(), поскольку каждый из этих методов аннулирует результат выполнения другого. Чтобы задать URI и тип MIME всегда используйте метод setDataAndType().

Категория

Строка, содержащая прочие сведения о том, каким компонентом должна выполняться обработка этого объекта Intent. В объект Intent можно поместить любое количество описаний категорий, однако большинству объектов Intent категория не требуется. Вот некоторые стандартные категории:

CATEGORY_BROWSABLE

Целевая операция позволяет запускать себя веб-браузером для отображения данных, указанных по ссылке — например, изображения или сообщения электронной почты.

CATEGORY_LAUNCHER

Эта операция является начальной операцией задачи, она указана в средстве запуска приложений системы.

Полный список категорий см. в описании класса Intent.

Указать категорию можно с помощью addCategory().

Приведенные выше свойства (имя компонента, действие, данные и категория) представляют собой характеристики, определяющие объект Intent. На основании этих свойств система Android может решить, какой компонент следует запустить.

Однако в объекте Intent может быть приведена и другая информация, которая не влияет на то, каким образом определяется требуемый компонент приложения. Объект Intent также может содержать:

Дополнительные данные

Пары "ключ-значение", содержащие прочую информацию, которая необходима для выполнения запрошенного действия. Точно так же, как некоторые действия используют определенные виды URI данных, некоторые действия используют определенные дополнительные данные.

Добавлять дополнительные данные можно с помощью различных методов putExtra(), каждый из которых принимает два параметра: имя и значение ключа. Также можно создать объект Bundle со всеми дополнительными данными, затем вставить объект Bundle в объект Intent с помощью метода putExtras().

Например, при создании объекта Intent для отправки сообщения электронной почты с методом ACTION_SEND можно указать получателя с помощью ключа EXTRA_EMAIL, а тему сообщения — с помощью ключа EXTRA_SUBJECT.

Класс Intent указывает много констант EXTRA_* для стандартных типов данных. Если вам требуется объявить собственные дополнительные ключи (для объектов Intent, которые принимает ваше приложение), обязательно указывайте в качестве префикса имя пакета своего приложения. Например:

```
static final String EXTRA_GIGAWATTS =  
"com.example.EXTRA_GIGAWATTS";  
Флаги
```

Флаги, определенные в классе Intent, которые действуют как метаданные для объекта Intent. Флаги должны указывать системе Android, каким образом следует запускать операцию (например, к какой задаче должна принадлежать операция) и как с ней обращаться после запуска (например, будет ли она указана в списке последних операций).

Подробные сведения см. в документе, посвященном методу setFlags().

Пример явного объекта Intent

Явные объекты Intent используются для запуска конкретных компонентов приложения, например определенной операции или службы. Чтобы создать явный объект Intent, задайте имя компонента для объекта Intent — все остальные свойства объекта Intent можно не задавать.

Например, если в своем приложении вы создали службу с именем DownloadService, предназначенную для загрузки файлов из Интернета, то для ее запуска можно использовать следующий код:

```
// Executed in an Activity, so 'this' is the Context  
// The fileUrl is a string URL, such as "http://www.example.com/image.png"  
Intent downloadIntent = new Intent(this, DownloadService.class);  
downloadIntent.setData(Uri.parse(fileUrl));  
startService(downloadIntent);
```

Конструктор Intent(Context, Class) предоставляет Context приложению, а компоненту — объект Class. Фактически этот объект Intent явно запускает класс DownloadService в приложении.

Подробные сведения о создании и запуске службы см. в руководстве Службы.

Пример неявного объекта Intent

Неявный объект Intent указывает действие, которым может быть вызвано любое имеющееся на устройстве приложение, способное выполнить это действие. Неявные объекты Intent используются, когда ваше приложение не может выполнить то или иное действие, а другие приложения, скорее всего, могут и вы хотите, чтобы пользователь имел возможность выбрать, какое приложение для этого использовать.

Например, если у вас есть контент и вы хотите, чтобы пользователь поделился им с другими людьми, создайте объект Intent с действием ACTION_SEND и добавьте дополнительные данные, указывающие на контент, общий доступ к которому следует предоставить. Когда с помощью этого объекта Intent вы вызываете startActivity(), пользователь сможет выбрать

приложение, посредством которого к контенту будет предоставлен общий доступ.

Внимание! Возможна ситуация, когда на устройстве пользователя не будет никакого приложения, которое может откликнуться на неявный объект Intent, отправленный вами методу startActivity(). В этом случае вызов закончится неудачей, а работа приложения аварийно завершится. Чтобы проверить, будет получен ли операцией объект Intent, вызовите метод resolveActivity() для своего объекта Intent. Если результатом будет значение, отличное от null, значит, имеется хотя бы одно приложение, которое способно откликнуться на объект Intent и можно вызывать startActivity(). Если же результатом будет значение null, объект Intent не следует использовать и по возможности следует отключить функцию, которая выдает этот объект Intent.

```
// Create the text message with a string
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
sendIntent.setType("text/plain");

// Verify that the intent will resolve to an activity
if (sendIntent.resolveActivity(getPackageManager()) != null) {
    startActivity(sendIntent);
}
```

Примечание. В этом случае URI не используется, а вместо этого следует объявить тип данных объекта Intent, чтобы указать контент, содержащийся в дополнительных данных.

При вызове метода startActivity() система анализирует все установленные приложения, чтобы определить, какие из них могут откликнуться на объект Intent этого вида (объект Intent с действием ACTION_SEND и данными "text/plain"). Если имеется только одно подходящее приложение, оно будет сразу же открыто и получит данный объект Intent. Если объект Intent принимают несколько операций, система отображает диалоговое окно, в котором пользователь может выбрать приложение для выполнения данного действия.



Рисунок 2. Диалоговое окно выбора.

Принудительное использование блока выбора приложения

При наличии нескольких приложений, откликающихся на ваш неявный объект Intent, пользователь может выбрать требуемое приложение и указать, что оно будет по умолчанию выполнять это действие. Это удобно в случае действия, для выполнения которого пользователь обычно хочет всегда использовать одно и то же приложение, например, при открытии веб-страницы (пользователи обычно используют один и тот же браузер).

Однако если на объект Intent могут откликнуться несколько приложений, возможно, пользователь предпочтет каждый раз использовать другое приложение, поэтому следует явно выводить диалоговое окно выбора. В диалоговом окне выбора приложения пользователю предлагается при каждом запуске выбирать, какое приложение использовать для действия (пользователь не может выбрать приложение, используемое по умолчанию). Например, когда ваше приложение выполняет операцию "share" (поделиться) с помощью действия ACTION_SEND, пользователи могут, в зависимости от ситуации, предпочесть каждый раз делать это с помощью разных приложений

, поэтому следует всегда использовать диалоговое окно выбора, как показано на рисунке 2.

Чтобы вывести на экран блок выбора приложения, создайте Intent с помощью createChooser() и передайте его startActivity(). Например:

```
Intent sendIntent = new Intent(Intent.ACTION_SEND);
```

```
...
```

```
// Always use string resources for UI text.
```

```
// This says something like "Share this photo with"
```

```
String title = getResources().getString(R.string.chooser_title);
```

```
// Create intent to show the chooser dialog
```

```
Intent chooser = Intent.createChooser(sendIntent, title);
```

```
// Verify the original intent will resolve to at least one activity
```

```
if (sendIntent.resolveActivity(getPackageManager()) != null) {  
    startActivity(chooser);
```

```
}
```

В результате на экран будет выведено диалоговое окно со списком приложений, которые могут отреагировать на объект Intent, переданный методу createChooser() и используют указанный текст в качестве заголовка диалога.

Получение неявного объекта Intent

Чтобы указать, какие неявные объекты Intent может принимать ваше приложение, объявите один или несколько фильтров Intent для каждого компонента приложения с помощью элемента <intent-filter> в файле манифеста. Каждый фильтр Intent указывает тип объектов Intent, которые принимает компонент на основании действия, данных и категории, заданных в объекте Intent. Система передаст неявный объект Intent вашему приложению, только если он может пройти через один из ваших фильтров Intent.

Примечание. Явный объект Intent всегда доставляется его целевому компоненту, без учета любых фильтров Intent, объявленных компонентом.

Компонент приложения должен объявлять отдельные фильтры для каждой уникальной работы, которую он может выполнить. Например, у операции из приложения для работы с галереей изображений может быть два фильтра: один фильтр для просмотра изображения, и второй для его редактирования. Когда операция запускается, она анализирует объект Intent и выбирает режим своей работы на основании информации, приведенной в Intent (например, показывать элементы управления редактора или нет).

Каждый фильтр Intent определяется элементом <intent-filter> в файле манифеста приложения, указанном в объявлении соответствующего компонента приложения (например, в элементе <activity>). Внутри элемента <intent-filter>, можно указать тип объектов Intent, которые будут приниматься, с помощью одного или нескольких из следующих трех элементов:

<action>

Объявляет принимаемое действие, заданное в объекте Intent, в атрибуте name. Значение должно быть текстовой строкой действия, а не константой класса.

<data>

Объявляет тип принимаемых данных, для чего используется один или несколько атрибутов, указывающих различные составные части URI данных (scheme, host, port, path и т. д.) и тип MIME.

<category>

Объявляет принимаемую категорию, заданную в объекте Intent, в атрибуте name. Значение должно быть текстовой строкой действия, а не константой класса.

Примечание. Для получения неявных объектов Intent необходимо включить категорию CATEGORY_DEFAULT в фильтр Intent. Методы startActivity() и startActivityForResult() обрабатывают все объекты Intent, как если бы они объявляли категорию CATEGORY_DEFAULT. Если вы не объявляете эту категорию в своем фильтре Intent, никакие неявные объекты Intent не будут переданы в вашу операцию.

В следующем примере объявлена операция с фильтром Intent, определяющим получение объекта Intent ACTION_SEND, когда данные относятся к типу text:

```
<activity android:name="ShareActivity">
  <intent-filter>
    <action android:name="android.intent.action.SEND"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <data android:mimeType="text/plain"/>
  </intent-filter>
</activity>
```

Можно создавать фильтры, в которых будет несколько экземпляров <action>, <data> или <category>. В этом случае просто нужно убедиться в том, что компонент может справиться с любыми сочетаниями этих элементов фильтра.

Когда требуется обрабатывать объекты Intent нескольких видов, но только в определенных сочетаниях действия, типа данных и категории, необходимо создать несколько фильтров Intent.

Ограничение доступа к компонентам

Использование фильтра Intent не является безопасным способом предотвращения запуска ваших компонентов другими приложениями. Несмотря на то, что после применения фильтров Intent компонент будет реагировать только на неявные объекты Intent определенного вида, другое приложение теоретически может запустить компонент вашего приложения с помощью явного объекта Intent, если разработчик определит имена ваших компонентов. Если важно, чтобы только ваше собственное приложение могло запускать один из ваших компонентов, задайте для атрибута exported этого компонента значение "false".

Неявный объект Intent проверяется фильтром путем сравнения объекта Intent с каждым из этих трех элементов. Чтобы объект Intent был доставлен компоненту, он должен пройти все три теста. Если он не будет соответствовать хотя бы одному из них, система Android не доставит этот объект Intent компоненту. Однако, поскольку у компонента может быть несколько фильтров Intent, объект Intent, который не проходит через один из фильтров компонента, может пройти через другой фильтр. Подробные сведения о том, каким образом система принимает решения по поводу объектов Intent, см. в приведенном далее разделе Разрешение объектов Intent.

Внимание! Чтобы случайно не запустить Service другого приложения, всегда используйте явные объекты Intent для запуска собственных служб и не объявляйте для них фильтры Intent.

Примечание. Фильтры Intent необходимо объявлять в файле манифеста для всех операций. Фильтры для приемников широковещательных сообщений можно регистрировать динамически путем вызова registerReceiver(). После этого регистрацию приемника широковещательных сообщений можно отменить с помощью unregisterReceiver(). В результате ваше приложение сможет воспринимать определенные объявления только в течение указанного периода времени в процессе работы приложения.

Примеры фильтров

Чтобы лучше понять различные режимы работы фильтров Intent, рассмотрите следующий фрагмент из файла манифеста приложения для работы с социальными сетями.

```
<activity android:name="MainActivity">
    <!-- This activity is the main entry, should appear in app launcher -->
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>

<activity android:name="ShareActivity">
    <!-- This activity handles "SEND" actions with text data -->
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
    </intent-filter>
    <!-- This activity also handles "SEND" and "SEND_MULTIPLE" with
media data -->
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <action android:name="android.intent.action.SEND_MULTIPLE"/>
        <category android:name="android.intent.category.DEFAULT"/>
```

```

        <data
android:mimeType="application/vnd.google.panorama360+jpg"/>
        <data android:mimeType="image/*"/>
        <data android:mimeType="video/*"/>
    </intent-filter>
</activity>

```

Первая операция (MainActivity) является основной точкой входа приложения — это операция, которая открывается, когда пользователь запускает приложение нажатием на его значок:

Действие ACTION_MAIN указывает на то, что это основная точка входа, и не ожидает никаких данных объекта Intent.

Категория CATEGORY_LAUNCHER указывает, что значок этой операции следует поместить в средство запуска приложений системы. Если элемент <activity> не содержит указаний на конкретный значок с помощью icon, то система воспользуется значком из элемента <application>.

Чтобы операция отображалась в средстве запуска приложений системы, два этих элемента необходимо связать вместе.

Вторая операция (ShareActivity) предназначена для упрощения обмена текстовым и мультимедийным контентом. Несмотря на то, что пользователи могут входить в эту операцию, выбрав ее из MainActivity, они также могут входить в ShareActivity напрямую из другого приложения, которое выдает неявный объект Intent, соответствующий одному из двух фильтров Intent.

Примечание. Тип MIME (application/vnd.google.panorama360+jpg) является особым типом данных, указывающим на панорамные фотографии, с которыми можно работать с помощью API-интерфейсов Google panorama.

Использование ожидающего объекта Intent

Объект PendingIntent является оболочкой, в которую заключается объект Intent. Объект PendingIntent в основном предназначен для того, чтобы предоставлять разрешение внешним приложениям на использование содержащегося в нем объекта Intent, как если бы он исполнялся из процесса вашего собственного приложения.

Основные варианты использования ожидающего объекта Intent:

Объявление объекта Intent, который должен будет исполняться, когда пользователь выполняет действие с вашим уведомлением (NotificationManager системы Android исполняет Intent).

Объявление объекта Intent, который должен будет исполняться, когда пользователь выполняет действие с вашим виджетом приложения (приложение главного экрана исполняет Intent).

Объявление объекта Intent, который должен будет исполняться в указанное время в будущем (AlarmManager системы Android исполняет Intent).

Поскольку каждый объект Intent предназначен для обработки компонентом приложения, который относится к определенному типу (Activity, Service или BroadcastReceiver), объект PendingIntent также следует создавать с учетом этого обстоятельства. При использовании ожидающего

объекта `Intent` ваше приложение не будет исполнять объект `Intent` вызовом, например, `startActivity()`. Вместо этого вам необходимо будет объявить требуемый тип компонента при создании `PendingIntent` путем вызова соответствующего метода-создателя:

Метод `PendingIntent.getActivity()` для `Intent`, который запускает `Activity`.

Метод `PendingIntent.getService()` для `Intent`, который запускает `Service`.

Метод `PendingIntent.getBroadcast()` для `Intent`, который запускает `BroadcastReceiver`.

Если только ваше приложение не принимает ожидающие объекты `Intent` от других приложений, указанные выше методы создания `PendingIntent` являются единственными методами `PendingIntent`, которые вам когда-либо понадобятся.

Каждый метод принимает текущий `Context` приложения, объект `Intent`, который требуется поместить в оболочку, и один или несколько флагов, указывающих, каким образом следует использовать объект `Intent` (например, можно ли использовать объект `Intent` неоднократно).

Подробные сведения об использовании ожидающих объектов `Intent` приведены в документации по каждому из соответствующих вариантов использования, например, в руководствах, посвященных API-интерфейсам: Уведомления и Виджеты приложений.

Разрешение объектов `Intent`

Когда система получает неявный объект `Intent` для запуска операции, она выполняет поиск наиболее подходящей операции путем сравнения объекта `Intent` с фильтрами `Intent` по трем критериям:

действие объекта `Intent`;

данные объекта `Intent` (тип URI и данных);

категория объекта `Intent`.

В следующих разделах описывается, каким образом объекты `Intent` сопоставляются с соответствующими компонентами, а именно, как должен быть фильтр `Intent` объявлен в файле манифеста приложения.

Тестирование действия

Для указания принимаемых действий объекта `Intent` фильтр `Intent` может объявлять любое (в том числе нулевое) число элементов `action`. Например:

```
<intent-filter>
  <action android:name="android.intent.action.EDIT" />
  <action android:name="android.intent.action.VIEW" />
  ...
</intent-filter>
```

Чтобы пройти через этот фильтр, действие, указанное в объекте `Intent`, должно соответствовать одному или нескольким действиям, перечисленным в фильтре.

Если в фильтре не перечислены какие-либо действия, объекту `Intent` будет нечему соответствовать, поэтому все объекты `Intent` не пройдут этот тест. Однако, если в объекте `Intent` не указано действие, он пройдет тест (если в фильтре содержится хотя бы одно действие).

Тестирование категории

Для указания принимаемых категорий объекта Intent фильтр Intent может объявлять любое (в том числе нулевое) число элементов `<category>`. Например:

```
<intent-filter>
  <category android:name="android.intent.category.DEFAULT" />
  <category android:name="android.intent.category.BROWSABLE" />
  ...
</intent-filter>
```

Чтобы объект Intent прошел тестирование категории, все категории, приведенные в объекте `Intent`, должны соответствовать категории из фильтра. Обратное не требуется — фильтр Intent может объявлять и другие категории, которых нет в объекте `Intent`, объект `Intent` при этом все равно пройдет тест. Поэтому объект Intent без категорий всегда пройдет этот тест, независимо от того, какие категории объявлены в фильтре.

Примечание. Система Android автоматически применяет категорию `CATEGORY_DEFAULT` ко всем неявным объектам Intent, которые передаются в `startActivity()` и `startActivityForResult()`. Поэтому, если вы хотите, чтобы ваша операция принимала неявные объекты Intent, в ее фильтрах Intent должна быть указана категория для `"android.intent.category.DEFAULT"` (как показано в предыдущем примере `<intent-filter>`).

Тестирование данных

Для указания принимаемых данных объекта Intent фильтр Intent может объявлять любое (в том числе нулевое) число элементов `<data>`. Например:

```
<intent-filter>
  <data android:mimeType="video/mpeg" android:scheme="http" ... />
  <data android:mimeType="audio/mpeg" android:scheme="http" ... />
  ...
</intent-filter>
```

Каждый элемент `<data>` может конкретизировать структуру URI и тип данных (тип мультимедиа MIME). Имеются отдельные атрибуты — `scheme`, `host`, `port` и `path` — для каждой составной части URI:

```
<scheme>://<host>:<port>/<path>
```

Например:

```
content://com.example.project:200/folder/subfolder/etc
```

В этом URI схема имеет вид `content`, узел — `com.example.project`, порт — `200`, а путь — `folder/subfolder/etc`.

Каждый из этих атрибутов является необязательным в элементе `<data>`, однако имеются линейные зависимости:

Если схема не указана, узел игнорируется.

Если узел не указан, порт игнорируется.

Если не указана ни схема, ни узел, путь игнорируется.

Когда URI, указанный в объекте Intent, сравнивается с URI из фильтра, сравнение выполняется только с теми составными частями URI, которые приведены в фильтре. Например:

Если в фильтре указана только схема, то все URI с этой схемой будет соответствовать фильтру.

Если в фильтре указаны схема и полномочия, но отсутствует путь, все URI с такими же схемой и полномочиями пройдут фильтр, а их пути учитываться не будут.

Если в фильтре указаны схема, полномочия и путь, то только URI с такими же схемой, полномочиями и путем пройдут фильтр.

Примечание. Путь может быть указан с подстановочным символом (*), чтобы потребовалось только частичное соответствие имени пути.

При выполнении тестирования данных сравнивается и URI, и тип MIME, указанные в объекте Intent, с URI и типом MIME из фильтра. Действуют следующие правила:

Объект Intent, который не содержит ни URI, ни тип MIME, пройдет этот тест, только если в фильтре не указано никаких URI или типов MIME.

Объект Intent, в котором имеется URI, но отсутствует тип MIME (ни явный, ни тот, который можно вывести из URI), пройдет этот тест, только если URI соответствует формату URI из фильтра, а в фильтре также не указан тип MIME.

Объект Intent, в котором имеется тип MIME, но отсутствует URI, пройдет этот тест, только если в фильтре указан тот же тип MIME и не указан формат URI.

Объект Intent, в котором имеется и URI, и тип MIME (явный или тот, который можно вывести из URI), пройдет только часть этого теста, проверяющую тип MIME, в том случае, если этот тип совпадает с типом, приведенным в фильтре. Он пройдет часть этого теста, которая проверяет URI, либо если его URI совпадает с URI из фильтра, либо если этот объект содержит URI content: или file:, а в фильтре URI не указан. Другими словами, предполагается, что компонент поддерживает данные content: и file:, если в его фильтре указан только тип MIME.

Это последнее правило (правило (d)) отражает ожидание того, что компоненты будут в состоянии получать локальные данные из файла или от поставщика контента. Поэтому их фильтры могут содержать только тип данных, а явно указывать схемы content: и file: не требуется. Это типичный случай. Например, такой элемент `<data>`, как приведенный далее, сообщает системе Android, что компонент может получать данные изображений от поставщика контента и выводить их на экран:

```
<intent-filter>
  <data android:mimeType="image/*" />
  ...
</intent-filter>
```

Поскольку имеющиеся данные преимущественно распространяются поставщиками контента, фильтры, в которых указан тип данных и нет URI, вероятно, являются самыми распространенными.

Другой стандартной конфигурацией являются фильтры со схемой и типом данных. Например, такой элемент `<data>`, как приведенный далее,

сообщает системе Android, что компонент может получать видеоданные из сети для выполнения действия:

```
<intent-filter>  
    <data android:scheme="http" android:type="video/*" />  
    ...  
</intent-filter>
```

Сопоставление объектов Intent

Объекты Intent сопоставляются с фильтрами Intent не только для определения целевого компонента, который требуется активировать, но также для выявления определенных сведений о наборе компонентов, имеющихся на устройстве. Например, приложение главного экрана заполняет средство запуска приложений путем поиска всех операций с фильтрами Intent, в которых указано действие ACTION_MAIN и категория CATEGORY_LAUNCHER.

Ваше приложение может использовать сопоставление объектов Intent таким же образом. В PackageManager имеется набор методов query...(), которые возвращают все компоненты, способные принять определенный объект, а также схожий набор методов resolve...(), которые определяют наиболее подходящий компонент, способный реагировать на объект Intent. Например, метод queryIntentActivities() возвращает передаваемый как аргумент список всех операций, которые могут выполнить объект Intent, а метод queryIntentServices() возвращает такой же список служб. Ни тот, ни другой метод не активирует компоненты; они просто перечисляют те из них, которые могут откликнуться. Имеется схожий метод для приемников широковещательных сообщений (`@link android.content.pm.PackageManager#queryBroadcastReceivers`).

ЛЕКЦИЯ №48-№49. ИСПОЛЬЗОВАНИЕ РЕСУРСОВ. ССЫЛКИ НА РЕСУРСЫ.

После того, как вы предоставили ресурс в вашем приложении (см. раздел Предоставление ресурсов), этот ресурс можно применить. Для этого необходимо создать ссылку на идентификатор ресурса. Для задания всех таких идентификаторов в вашем проекте используется класс R, который автоматически создается инструментом aapt.

Во время компиляции приложения инструмент aapt создает класс R, в котором находятся идентификаторы для всех ресурсов в каталоге res/. Для каждого типа ресурсов предусмотрен подкласс R (например, класс R.drawable для элементов дизайна), а для каждого ресурса указанного типа существует статическая целочисленная переменная (например, R.drawable.icon). Эта переменная как раз и служит идентификатором ресурса, которую можно использовать для его получения.

Несмотря на то, что в классе R находятся идентификаторы ресурсов, никогда не обращайтесь к нему для поиска идентификатора ресурса. Последний состоит из следующих компонентов:

Тип ресурса: ресурсы объединены по типам, таким как `string`, `drawable` и `layout`. Дополнительные сведения о различных типах представлены в разделе Типы ресурсов.

Имя ресурса, в качестве которого выступает либо имя файла (без расширения), либо значение атрибута XML `android:name` (если ресурс представляет собой простое значение, например строку).

Существует два способа доступа к ресурсу.

Из кода: с помощью статической целочисленной переменной из подкласса вашего класса `R`, например:

`R.string.hello`

`string` — это тип ресурса, а `hello` — это его имя. Существует множество API-интерфейсов Android, которые могут получать доступ к ресурсам, идентификаторы которых указаны в этом формате. См. раздел Доступ к ресурсам из кода.

Из XML: с помощью особого синтаксиса XML, который также соответствует идентификатору ресурса, заданному в классе `R`, например:

`@string/hello`

`string` — это тип ресурса, а `hello` — это его имя. Этот синтаксис можно использовать в любом фрагменте ресурса XML, где ожидается значение, указанное вами в ресурсе. См. раздел Доступ к ресурсам из XML

Доступ к ресурсам из кода

Чтобы использовать ресурс в коде, можно передать идентификатор ресурса в виде параметра метода. Например, с помощью метода `setImageResource()` можно указать использование виджетом `ImageView` ресурса `res/drawable/myimage.png`:

```
ImageView imageView = (ImageView) findViewById(R.id.myimageview);  
imageView.setImageResource(R.drawable.myimage);
```

Отдельные ресурсы также можно получать с помощью методов в классе `Resources`, экземпляр которого можно получить с помощью метода `getResources()`.

Доступ к исходным файлам

В редких случаях может потребоваться получить доступ к исходным файлам и каталогам. В этом случае просто сохранить файлы в каталоге `res/` будет недостаточно, поскольку обратиться к ресурсу из папки `res/` можно только по его идентификатору. Вместо этого ресурсы можно сохранить в каталоге `assets/`.

Файлам, которые сохранены в каталоге `assets/`, не присваиваются идентификаторы ресурсов, поэтому вам не удастся сослаться на них с помощью класса `R` или из ресурсов XML. Вместо этого можно запросить файлы из каталога `assets/`, как в обычной файловой системе, и считать необработанные данные с помощью `AssetManager`.

Однако, если вам требуется всего лишь возможность считать необработанные данные (например, видео- или аудиофайл), сохраните

требуемый файл в каталоге `res/raw/` и считайте поток байтов с помощью метода `openRawResource()`.

Синтаксис

Ниже представлен синтаксис ссылки на ресурс из кода.

`[<package_name>.]R.<resource_type>.<resource_name>`

`<package_name>` — это имя пакета, в котором находится ресурс (не требуется при создании ссылок на ресурсы из вашего собственного пакета).

`<resource_type>` — это подкласс `R` для типа ресурса.

`<resource_name>` — это либо имя файла ресурса (без расширения), либо значение атрибута `android:name` в элементе XML (для простых значений).

Дополнительные сведения о каждом типе ресурсов и порядке создания ссылок на них см. в разделе Типы ресурсов.

Примеры использования

Существует множество методов, которые могут принимать идентификатор ресурса в виде параметра. Для получения ресурсов можно использовать методы, представленные в классе Resources. Можно получить экземпляр Resources с помощью Context.getResources().

Вот некоторые примеры доступа к ресурсам из кода:

```
// Load a background for the current screen from a drawable resource
getWindow().setBackgroundDrawableResource(R.drawable.my_background_image);
```

```
// Set the Activity title by getting a string from the Resources object, because
// this method requires a CharSequence rather than a resource ID
getWindow().setTitle(getResources().getText(R.string.main_title));
```

```
// Load a custom layout for the current screen
setContentView(R.layout.main_screen);
```

```
// Set a slide in animation by getting an Animation from the Resources object
mFlipper.setInAnimation(AnimationUtils.loadAnimation(this,
                                                         R.anim.hyperspace_in));
```

```
// Set the text on a TextView object using a resource ID
TextView msgTextView = (TextView) findViewById(R.id.msg);
msgTextView.setText(R.string.hello_message);
```

Внимание! Никогда не изменяйте файл `R.java` вручную — этот файл создается инструментом `aapt` во время компиляции вашего проекта. Любые внесенные в него изменения перезаписываются после следующей компиляции.

Доступ к ресурсам из XML

Для задания значений для некоторых атрибутов и элементов XML можно использовать ссылку на существующий ресурс. Это зачастую

требуется при создании файлов макета при указании строк и изображений для виджетов.

Например, при добавлении в макет элемента Button необходимо использовать строковый ресурс для надписи на кнопке:

```
<Button
                                android:layout_width="fill_parent"
                                android:layout_height="wrap_content"
                                android:text="@string/submit"           />
```

Синтаксис

Ниже представлен синтаксис ссылки на ресурс из ресурса XML.

@[<package_name>:]<resource_type>/<resource_name>

<package_name> — это имя пакета, в котором находится ресурс (не требуется при создании ссылок на ресурсы из одного и того же пакета).

<resource_type> — это подкласс R для типа ресурса.

<resource_name> — это либо имя файла ресурса (без расширения), либо значение атрибута android:name в элементе XML (для простых значений).

Дополнительные сведения о каждом типе ресурсов и порядке создания ссылок на них см. в разделе Типы ресурсов.

Примеры использования

В некоторых случаях ресурс необходимо использовать в качестве значения в элементе XML (например, чтобы применить графическое изображение к виджету), однако вы также можете использовать ресурс в любом фрагменте XML, где ожидается простое значение. Например, если имеется следующий файл ресурса, включающий цветовой ресурс и строковый ресурс:

```
<?xml                                version="1.0"                                encoding="utf-8"?>
<resources>
                                <color                                name="opaque_red">#f00</color>
                                <string                                name="hello">Hello!</string>
</resources>
```

Эти ресурсы можно использовать в следующем файле макета для задания цвета текста и надписи:

```
<?xml                                version="1.0"                                encoding="utf-8"?>
<EditText                                xmlns:android="http://schemas.android.com/apk/res/android"
                                android:layout_width="fill_parent"
                                android:layout_height="fill_parent"
                                android:textColor="@color/opaque_red"
                                android:text="@string/hello"           />
```

В этом случае в ссылке на ресурс не нужно указывать имя пакета, поскольку ресурсы находятся в вашем собственном пакете. Однако для создания ссылки на системный ресурс вам потребуется указать имя пакета. Например:

```

        <?xml                                version="1.0"                                encoding="utf-8"?>
<EditText                                xmlns:android="http://schemas.android.com/apk/res/android"
                                           android:layout_width="fill_parent"
                                           android:layout_height="fill_parent"
                                           android:textColor="@android:color/secondary_text_dark"
                                           android:text="@string/hello"                                />

```

Примечание. Всегда используйте строковые ресурсы, поскольку ваше приложение может потребоваться перевести на другие языки. Сведения о создании альтернативных ресурсов (например, локализованных строк) представлены в разделе Предоставление альтернативных ресурсов. В разделе Локализация приведено полное руководство по локализации приложения.

Вы даже можете использовать ресурсы в XML для создания псевдонимов. Например, можно создать элемент дизайна, который будет служить псевдонимом для другого элемента дизайна:

```

        <?xml                                version="1.0"                                encoding="utf-8"?>
<bitmap                                xmlns:android="http://schemas.android.com/apk/res/android"
                                           android:src="@drawable/other_drawable"                                />

```

Это может показаться излишним, однако такой подход очень полезен при использовании альтернативных ресурсов. Узнайте подробнее о создании псевдонимов ресурсов.

Ссылка на атрибуты стиля

С помощью ресурса атрибута стиля можно создать ссылку на значение атрибута в текущей примененной теме. Таким образом можно настраивать внешний вид элементов пользовательского интерфейса, подстраивая их под стандартные варианты элементов оформления текущей темы, вместо указания жестко заданных значений. Создание ссылки на атрибут стиля представляет собой своеобразную инструкцию «использовать стиль, заданный этим атрибутом в текущей теме».

Синтаксис для создания ссылки на атрибут стиля практически идентичен обычному формату ресурса, только в этом случае вместо символа@ необходимо указать вопросительный знак (?), а тип ресурса вообще необязательно указывать. Например:

```
?[<package_name>:][<resource_type>/]<resource_name>
```

Ниже представлен пример создания ссылки на атрибут для задания цвета текста в соответствии с «основным» цветом текста системной темы оформления:

```

<EditText                                id="text"
                                           android:layout_width="fill_parent"
                                           android:layout_height="wrap_content"
                                           android:textColor="?android:textColorSecondary"
                                           android:text="@string/hello_world"                                />

```

Здесь атрибут `android:textColor` служит для задания имени атрибута стиля в текущей теме. Теперь в Android используется значение, примененное к атрибуту `android:textColorSecondary` в качестве значения для `android:textColor` в этом виджете. Поскольку инструменту для работы с системными ресурсами известно, что в этом контексте ожидается ресурс атрибута, вам не нужно явно указывать его тип (который должен быть `?android:attr/textColorSecondary`) — тип `attr` можно исключить.

Доступ к ресурсам платформы

В Android предусмотрен ряд стандартных ресурсов, например, стилей, тем и макетов. Для доступа к этим ресурсам укажите в ссылке на ресурс имя пакета `android`. Например, в Android имеется ресурс макета, который можно использовать для элементов списка в виджете ListAdapter:

```
setListAdapter(new ArrayAdapter<String>(this,  
android.R.layout.simple_list_item_1, myarray));
```

В этом примере simple_list_item_1 представляет собой ресурс макета, определенный платформой для элементов в виджете ListView. Вы можете использовать его вместо создания собственных макетов для элементов списка. Дополнительные сведения представлены в руководстве для разработчиков, посвященном представлению в виде списка.

ЛЕКЦИЯ №50-№51. ФАЙЛОВАЯ СИСТЕМА ANDROID. ЧТЕНИЕ И ЗАПИСЬ ФАЙЛОВ.

Работа с настройками уровня `activity` и приложения позволяет сохранить небольшие данные отдельных типов (`string`, `int`), но для работы с большими массивами данных, такими как графически файлы, файлы мультимедиа и т.д., нам придется обращаться к файловой системе.

ОС Android построена на основе Linux. Этот факт находит свое отражение в работе с файлами. Так, в путях к файлам в качестве разграничителя в Linux использует слеш `"/"`, а не обратный слеш `"\"` (как в Windows). А все названия файлов и каталогов являются регистрозависимыми, то есть `"data"` это не то же самое, что и `"Data"`.

Приложение Android сохраняет свои данные в каталоге `/data/data/<название_пакета>/` и, как правило, относительно этого каталога будет идти работа.

Для работы с файлами абстрактный класс `android.content.Context` определяет ряд методов:

`deleteFile(String name)`: удаляет определенный файл

`fileList()`: получает все файлы, которые содержатся в подкаталоге `/files` в каталоге приложения

`getCacheDir()`: получает ссылку на подкаталог `cache` в каталоге приложения

`getDir(String dirName, int mode)`: получает ссылку на подкаталог в каталоге приложения, если такого подкаталога нет, то он создается

`getExternalCacheDir()`: получает ссылку на папку /cache внешней файловой системы устройства

`getExternalFilesDir()`: получает ссылку на каталог /files внешней файловой системы устройства

`getFileStreamPath(String filename)`: возвращает абсолютный путь к файлу в файловой системе

`openFileInput(String filename)`: открывает файл для чтения

`openFileOutput (String name, int mode)`: открывает файл для записи

Все файлы, которые создаются и редактируются в приложении, как правило, хранятся в подкаталоге /files в каталоге приложения.

Для непосредственного чтения и записи файлов применяются также стандартные классы Java из пакета `java.io`.

Итак, применим функционал чтения-записи файлов в приложении. Пусть у нас будет следующая примитивная разметка layout:

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <EditText
        android:id="@+id/save_text"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:gravity="start"
        android:layout_weight="4"/>
    <Button
        android:layout_width="wrap_content"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:layout_margin="16dp"
        android:layout_gravity="center"
        android:onClick="saveText"
        android:text="Сохранить"/>

    <TextView
        android:layout_marginTop="80dp"
        android:id="@+id/open_text"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:gravity="start"
        android:layout_weight="4"/>
    <Button
        android:layout_width="wrap_content"
        android:layout_height="0dp"
        android:layout_margin="16dp"
```

```
        android:layout_weight="1"
        android:layout_gravity="center"
        android:onClick="openText"
        android:text="Открыть"/>
```

```
</LinearLayout>
```

Поле EditText предназначено для ввода текста, а TextView - для вывода ранее сохраненного текста. Для сохранения и восстановления текста добавлены две кнопки.

Теперь в коде Activity пропишем обработчики кнопок с сохранением и чтением файла:

```
package com.example.eugene.filesapp;
```

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;
```

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
```

```
public class MainActivity extends AppCompatActivity {
```

```
    private final static String FILE_NAME = "content.txt";
```

```
    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
```

```
    // сохранение файла
```

```
    public void saveText(View view){
```

```
        FileOutputStream fos = null;
```

```
        try {
```

```
            EditText textBox = (EditText) findViewById(R.id.save_text);
```

```
            String text = textBox.getText().toString();
```

```
            fos = openFileOutput(FILE_NAME, MODE_PRIVATE);
```

```
            fos.write(text.getBytes());
```

```
            Toast.makeText(this, "Файл сохранен",
                Toast.LENGTH_SHORT).show();
```



```

    }
    catch(IOException ex) {

        Toast.makeText(this,                                ex.getMessage(),
Toast.LENGTH_SHORT).show();
    }
    finally{
        try{
            if(fos!=null)
                fos.close();
        }
        catch(IOException ex){

            Toast.makeText(this,                                ex.getMessage(),
Toast.LENGTH_SHORT).show();
        }
    }
}
// открытие файла
public void openText(View view){

    FileInputStream fin = null;
    TextView textView = (TextView) findViewById(R.id.open_text);
    try {
        fin = openFileInput(FILE_NAME);
        byte[] bytes = new byte[fin.available()];
        fin.read(bytes);
        String text = new String (bytes);
        textView.setText(text);
    }
    catch(IOException ex) {

        Toast.makeText(this,                                ex.getMessage(),
Toast.LENGTH_SHORT).show();
    }
    finally{

        try{
            if(fin!=null)
                fin.close();
        }
        catch(IOException ex){

            Toast.makeText(this,                                ex.getMessage(),
Toast.LENGTH_SHORT).show();

```

```

    }
  }
}

```

При нажатии на кнопку сохранения будет создаваться поток вывода `FileOutputStream` `fos` = `openFileOutput(FILE_NAME, MODE_PRIVATE)`

В данном случае введенный текст будет сохраняться в файл "content.txt". При этом будет использоваться режим `MODE_PRIVATE`

Система позволяет создавать файлы с двумя разными режимами:

`MODE_PRIVATE`: файлы могут быть доступны только владельцу приложения (режим по умолчанию)

`MODE_APPEND`: данные могут быть добавлены в конец файла

Поэтому в данном случае если файл "content.txt" уже существует, то он будет перезаписан. Если же нам надо было дописать файл, тогда надо было бы использовать режим `MODE_APPEND`:

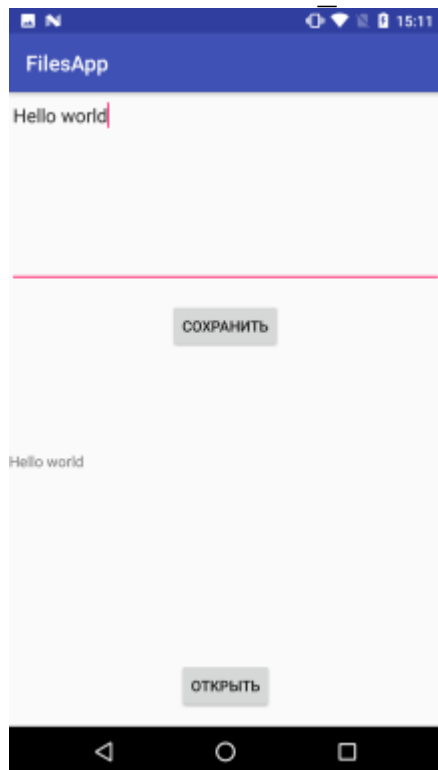
```
FileOutputStream fos = openFileOutput(FILE_NAME, MODE_APPEND);
```

Для чтения файла применяется поток ввода `FileInputStream`:

```
FileInputStream fin = openFileInput(FILE_NAME);
```

Подробнее про использование потоков ввода-вывода можно прочитать в руководстве по Java: <https://metanit.com/java/tutorial/6.3.php>

В итоге после нажатия кнопки сохранения весь текст будет сохранен в файле `/data/data/название_пакета/files/content.txt`



Размещение файлов во внешнем хранилище

В прошлой теме мы рассмотрели сохранение и чтение файлов из каталога приложения. По умолчанию такие файлы доступны только самому приложению. Однако мы можем помещать и работать с файлами из внешнего

хранилища. Это также позволит другим программам открывать данные файлы и при необходимости изменять.

Весь механизм работы с файлами будет таким же, как и при работе с хранилищем приложения. Ключевым отличием здесь будет получение и использование пути к внешнему хранилищу через метод `Environment.getExternalStorageDirectory()`

Итак, пусть в файле `activity_main.xml` будет такая же разметка интерфейса:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <EditText
        android:id="@+id/save_text"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:gravity="start"
        android:layout_weight="4"/>
    <Button
        android:layout_width="wrap_content"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:layout_margin="16dp"
        android:layout_gravity="center"
        android:onClick="saveText"
        android:text="Сохранить"/>

    <TextView
        android:layout_marginTop="80dp"
        android:id="@+id/open_text"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:gravity="start"
        android:layout_weight="4"/>
    <Button
        android:layout_width="wrap_content"
        android:layout_height="0dp"
        android:layout_margin="16dp"
        android:layout_weight="1"
        android:layout_gravity="center"
        android:onClick="openText"
        android:text="Открыть"/>

</LinearLayout>
```

А код класса `MainActivity` будет выглядеть следующим образом:

```

package com.example.eugene.filesapp;

import android.Manifest;
import android.content.pm.PackageManager;
import android.os.Environment;
import android.support.annotation.NonNull;
import android.support.v4.app.ActivityCompat;
import android.support.v4.content.ContextCompat;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class MainActivity extends AppCompatActivity {

    private final static String FILE_NAME = "content.txt";
    private static final int REQUEST_PERMISSION_WRITE = 1001;
    private boolean permissionGranted;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    private File getExternalPath() {
        return(new File(Environment.getExternalStorageDirectory(),
FILE_NAME));
    }
    // сохранение файла
    public void saveText(View view){

        if(!permissionGranted){
            checkPermissions();
            return;
        }
        FileOutputStream fos = null;
        try {
            EditText textBox = (EditText) findViewById(R.id.save_text);
            String text = textBox.getText().toString();

```

```

        fos = new FileOutputStream(getExternalPath());
        fos.write(text.getBytes());
        Toast.makeText(this, "Файл сохранен",
Toast.LENGTH_SHORT).show();
    }
    catch(IOException ex) {

        Toast.makeText(this, ex.getMessage(),
Toast.LENGTH_SHORT).show();
    }
    finally{
        try{
            if(fos!=null)
                fos.close();
        }
        catch(IOException ex){

            Toast.makeText(this, ex.getMessage(),
Toast.LENGTH_SHORT).show();
        }
    }
}
// открытие файла
public void openText(View view){
    if(!permissionGranted){
        checkPermissions();
        return;
    }

    FileInputStream fin = null;
    TextView textView = (TextView) findViewById(R.id.open_text);
    File file = getExternalPath();
    // если файл не существует, выход из метода
    if(!file.exists()) return;
    try {
        fin = new FileInputStream(file);
        byte[] bytes = new byte[fin.available()];
        fin.read(bytes);
        String text = new String (bytes);
        textView.setText(text);
    }
    catch(IOException ex) {

```

```

        Toast.makeText(this,                                ex.getMessage(),
Toast.LENGTH_SHORT).show();
    }
    finally{

        try{
            if(fin!=null)
                fin.close();
        }
        catch(IOException ex){

            Toast.makeText(this,                                ex.getMessage(),
Toast.LENGTH_SHORT).show();
        }
    }
}

// проверяем, доступно ли внешнее хранилище для чтения и записи
public boolean isExternalStorageWriteable(){
    String state = Environment.getExternalStorageState();
    return Environment.MEDIA_MOUNTED.equals(state);
}

// проверяем, доступно ли внешнее хранилище хотя бы только для чтения
public boolean isExternalStorageReadable(){
    String state = Environment.getExternalStorageState();
    return (Environment.MEDIA_MOUNTED.equals(state) ||
            Environment.MEDIA_MOUNTED_READ_ONLY.equals(state));
}

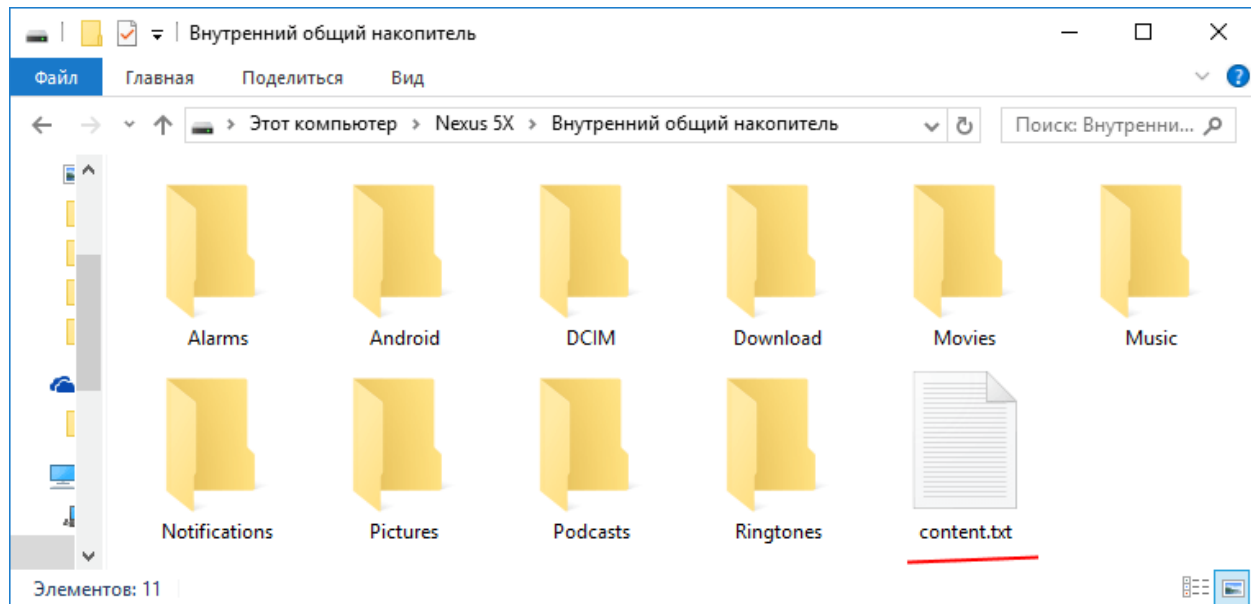
private boolean checkPermissions(){

    if(!isExternalStorageReadable() || !isExternalStorageWriteable()){
        Toast.makeText(this, "Внешнее хранилище не доступно",
Toast.LENGTH_LONG).show();
        return false;
    }

    int permissionCheck = ContextCompat.checkSelfPermission(this,
Manifest.permission.WRITE_EXTERNAL_STORAGE);
    if(permissionCheck!= PackageManager.PERMISSION_GRANTED){
        ActivityCompat.requestPermissions(this,                                new
String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE},
REQUEST_PERMISSION_WRITE);
        return false;
    }
    return true;
}

```


И после операции записи на смартфоне мы сможем увидеть созданный файл:



Работа с json

Для работы с форматом json нет встроенных средств, но есть куча библиотек и пакетов, которые можно использовать для данной цели. Одним из наиболее популярных из них является пакет `com.google.code.gson`.

Для его использования в проекте Android, необходимо добавить соответствующую зависимость в файл `build.gradle`, который относится к модулю `app`:

```
compile 'com.google.code.gson:gson:2.8.0'
```

То есть после добавления секция зависимостей в файле `build.gradle` может выглядеть следующим образом:

```
dependencies {  
    compile fileTree(dir: 'libs', include: ['*.jar'])  
    androidTestCompile('com.android.support.test.espresso:espresso-  
core:2.2.2', {  
        exclude group: 'com.android.support', module: 'support-annotations'  
    })  
    compile 'com.android.support:appcompat-v7:25.3.0'  
    compile 'com.android.support.constraint:constraint-layout:1.0.2'  
    compile 'com.google.code.gson:gson:2.8.0'  
    testCompile 'junit:junit:4.12'  
}
```

После добавления пакета в проект добавим новый класс `Phone`, который будет представлять данные:

```
package com.example.eugene.jsonapp;
```

```
public class Phone {
```

```
    private String name;
```

```
    private int price;
```



```

Phone(String name, int price){
    this.name = name;
    this.price = price;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public int getPrice() {
    return price;
}

public void setPrice(int price) {
    this.price = price;
}

@Override
public String toString(){
    return name + " " + String.valueOf(price);
}
}

```

Объекты этого класса мы будем сериализовать в формат json и наоборот десериализовать из файла.

Для работы с json добавим следующий класс JSONHelper:

```

package com.example.eugene.jsonapp;

import android.content.Context;
import com.google.gson.Gson;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.List;

class JSONHelper {

    private static final String FILE_NAME = "data.json";

    static boolean exportToJSON(Context context, List<Phone> dataList) {

```

```

Gson gson = new Gson();
DataItems dataItems = new DataItems();
dataItems.setPhones(dataList);
String jsonString = gson.toJson(dataItems);

FileOutputStream fileOutputStream = null;

try {
    fileOutputStream = context.openFileOutput(FILE_NAME,
Context.MODE_PRIVATE);
    fileOutputStream.write(jsonString.getBytes());
    return true;
} catch (Exception e) {
    e.printStackTrace();
} finally {
    if (fileOutputStream != null) {
        try {
            fileOutputStream.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

return false;
}

static List<Phone> importFromJSON(Context context) {

    InputStreamReader streamReader = null;
    FileInputStream fileInputStream = null;
    try{
        fileInputStream = context.openFileInput(FILE_NAME);
        streamReader = new InputStreamReader(fileInputStream);
        Gson gson = new Gson();
        DataItems dataItems = gson.fromJson(streamReader,
DataItems.class);
        return dataItems.getPhones();
    }
    catch (IOException ex){
        ex.printStackTrace();
    }
    finally {
        if (streamReader != null) {

```

```

        try {
            streamReader.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    if (fileInputStream != null) {
        try {
            fileInputStream.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

return null;
}

private static class DataItems {
    private List<Phone> phones;

    List<Phone> getPhones() {
        return phones;
    }
    void setPhones(List<Phone> phones) {
        this.phones = phones;
    }
}
}

```

Для работы с json создается объект Gson. Для сериализации данных в формат json у этого объекта вызывается метод toJson(), в который передаются сериализуемые данные.

Для упрощения работы с данными применяется вспомогательный класс DataItems. На выходе метод toJson() возвращает строку, которая затем сохраняется в текстовый файл.

Для десериализации выполняется метод fromJson(), в который передается объект Reader с сериализованными данными и тип, к которому надо десериализовать данные.

Теперь определим основной функционал для взаимодействия с пользователем. Изменим файл activity_main.xml следующим образом:

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"

```

```

android:orientation="vertical">

<EditText
    android:id="@+id/nameText"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Введите название"/>
<EditText
    android:id="@+id/priceText"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Введите цену"/>
<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Добавить"
    android:onClick="addPhone"/>
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal">
    <Button
        android:layout_width="0dp"
        android:layout_weight="1"
        android:layout_height="wrap_content"
        android:text="Сохранить"
        android:onClick="save"/>
    <Button
        android:layout_width="0dp"
        android:layout_weight="1"
        android:layout_height="wrap_content"
        android:text="Открыть"
        android:onClick="open"/>
</LinearLayout>
<ListView
    android:id="@+id/list"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />

</LinearLayout>

```

Здесь определены два текстовых поля для ввода названия модели и цены объекта Phone и одна кнопка для добавления данных в список. Еще одна кнопка выполняет сериализацию данных из списка в файл, а третья кнопка - восстановление данных из файла.

Для вывода самих данных определен элемент ListView.

И изменим класс MainActivity:

```
package com.example.eugene.jsonapp;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.EditText;
import android.widget.ListView;
import android.widget.Toast;

import java.util.ArrayList;
import java.util.List;

public class MainActivity extends AppCompatActivity {

    private ArrayAdapter<Phone> adapter;
    private EditText nameText, priceText;
    private List<Phone> phones;
    ListView listView;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        nameText = (EditText) findViewById(R.id.nameText);
        priceText = (EditText) findViewById(R.id.priceText);

        phones = new ArrayList<>();
        listView = (ListView) findViewById(R.id.list);
        adapter = new ArrayAdapter<>(this,
android.R.layout.simple_list_item_1, phones);
        listView.setAdapter(adapter);
    }

    public void addPhone(View view){
        String name = nameText.getText().toString();
        int price = Integer.parseInt(priceText.getText().toString());
        Phone phone = new Phone(name, price);
        phones.add(phone);
        adapter.notifyDataSetChanged();
    }

    public void save(View view){
```

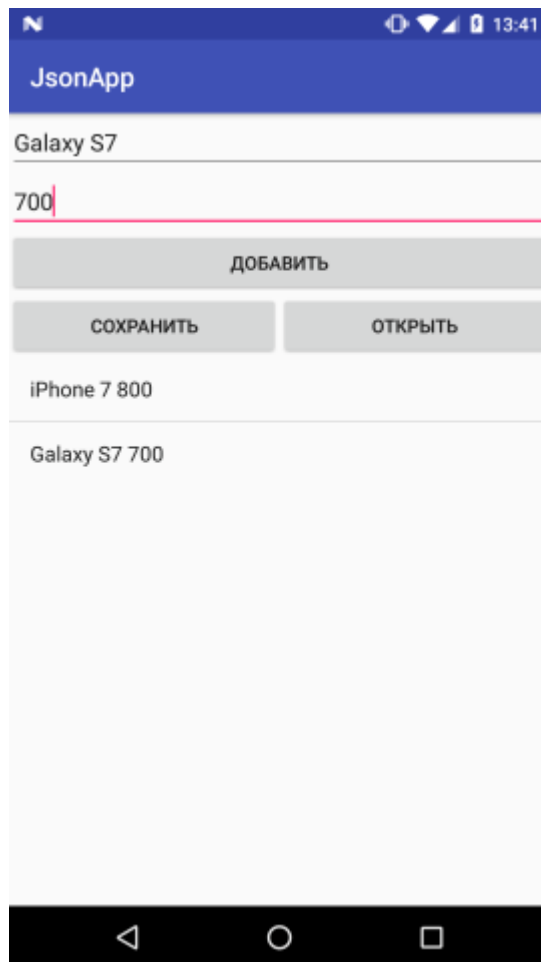
```

        boolean result = JSONHelper.exportToJSON(this, phones);
        if(result){
            Toast.makeText(this, "Данные сохранены",
Toast.LENGTH_LONG).show();
        }
        else{
            Toast.makeText(this, "Не удалось сохранить данные",
Toast.LENGTH_LONG).show();
        }
    }
    public void open(View view){
        phones = JSONHelper.importFromJSON(this);
        if(phones!=null){
            adapter = new ArrayAdapter<>(this,
android.R.layout.simple_list_item_1, phones);
            listView.setAdapter(adapter);
            Toast.makeText(this, "Данные восстановлены",
Toast.LENGTH_LONG).show();
        }
        else{
            Toast.makeText(this, "Не удалось открыть данные",
Toast.LENGTH_LONG).show();
        }
    }
}

```

Все данные находятся в списке phones, который представляет объект List<Phone>. Через адаптер этот список связывается с ListView.

Для сохранения и восстановления данных вызываются ранее определенные методы в классе JSONHelper.



ЛЕКЦИЯ №52-№53. АДАПТЕРЫ ДАННЫХ.

Виды адаптеров настолько повсеместны, что вам нелегко найти популярное приложение для Android, которое их не использует. Название может показаться незнакомым, но если вы думаете, что никогда не видели адаптер, вы, вероятно, ошибаетесь. Каждый раз, когда вы видите элементы интерфейса Android, отображающие элементы пользовательского интерфейса в виде списка, сетки или стека, вы видите представление адаптера в действии.

Представление адаптера, как следует из его названия, является объектом View. Это означает, что вы можете добавить его в свои действия так же, как вы добавляете любой другой виджет пользовательского интерфейса. Однако он не может отображать данные самостоятельно. Его содержимое всегда определяется другим объектом, адаптером. В этом уроке я покажу вам, как создавать адаптеры и использовать их для подачи различных видов адаптеров, таких как ListView и GridView.

1. Что такое адаптер?

Адаптер - это объект класса, который реализует интерфейс Adapter. Он действует как связь между набором данных и видом адаптера, объектом класса, который расширяет абстрактный класс AdapterView. Набор данных может быть любым, что представляет данные структурированным образом. Массивы, объекты List и объекты Cursor являются обычно используемыми наборами данных.

Адаптер отвечает за извлечение данных из набора данных и за создание объектов View на основе этих данных. Сгенерированные объекты View затем используются для заполнения любого вида адаптера, привязанного к адаптеру.

Вы можете создавать собственные классы адаптеров с нуля, но большинство разработчиков предпочитают использовать или расширять классы адаптеров, предоставляемые Android SDK, такие как ArrayAdapter и SimpleCursorAdapter. В этом уроке мы сосредоточимся на классе ArrayAdapter.

2. Как работают представления адаптера?

Представления адаптеров могут очень эффективно отображать большие наборы данных. Например, виджеты ListView и GridView могут отображать миллионы элементов без какого-либо заметного запаздывания, при этом память и загрузка процессора очень низки. Как они это делают? Различные виды адаптеров следуют различным стратегиям. Однако, вот что большинство из них обычно делает.

Они отображают только те объекты View, которые либо уже присутствуют на экране, либо собираются перемещаться по экрану. Таким образом, память, потребляемая представлением адаптера, может быть постоянной и независимой от размера набора данных.

Они также позволяют разработчикам свести к минимуму дорогостоящие операции раздувания макетов и перерабатывать существующие объекты View, которые перемещаются за пределы экрана. Это снижает потребление процессора.

3. Создание ArrayAdapter

Чтобы создать адаптер, вам необходимо следующее:

набор данных

файл ресурсов, содержащий макет сгенерированных объектов View

Кроме того, поскольку класс ArrayAdapter может работать только со строками, вам нужно убедиться, что макет сгенерированных объектов View содержит хотя бы один виджет TextView.

Шаг 1. Создание набора данных

Класс ArrayAdapter может использовать как массивы, так и объекты List в качестве наборов данных. Пока давайте будем использовать массив в качестве набора данных.

```
String[] cheeses = {  
    "Parmesan",  
    "Ricotta",  
    "Fontina",  
    "Mozzarella",  
    "Cheddar"  
};
```

Шаг 2. Создание файла ресурсов

Создайте новый XML-файл макета, корневым элементом которого является LinearLayout и назовите его item.xml. Перетащите в него Large

text виджет и установите значение его атрибута id в cheese_name. XML-файл макета должен выглядеть так:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="@dimen/activity_horizontal_margin">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="Large Text"
        android:id="@+id/cheese_name" />
</LinearLayout>
```

Шаг 3: Создайте адаптер

В своей деятельности создайте новый экземпляр класса ArrayAdapter, используя его конструктор. В качестве аргументов передайте имя файла ресурсов, идентификатор TextView и ссылку на массив. Теперь адаптер готов.

```
ArrayAdapter<String> cheeseAdapter =
    new ArrayAdapter<String>(this,
        R.layout.item,
        R.id.cheese_name,
        cheeses
    );
```

4. Создание списка

Чтобы отобразить вертикально прокручиваемый список элементов, вы можете использовать виджет ListView. Чтобы добавить виджет в свою деятельность, вы можете перетащить его в файл XML макета или создать его с помощью своего конструктора в вашем Java-коде. А пока давайте сделаем последнее.

```
ListView cheeseList = new ListView(this);
```

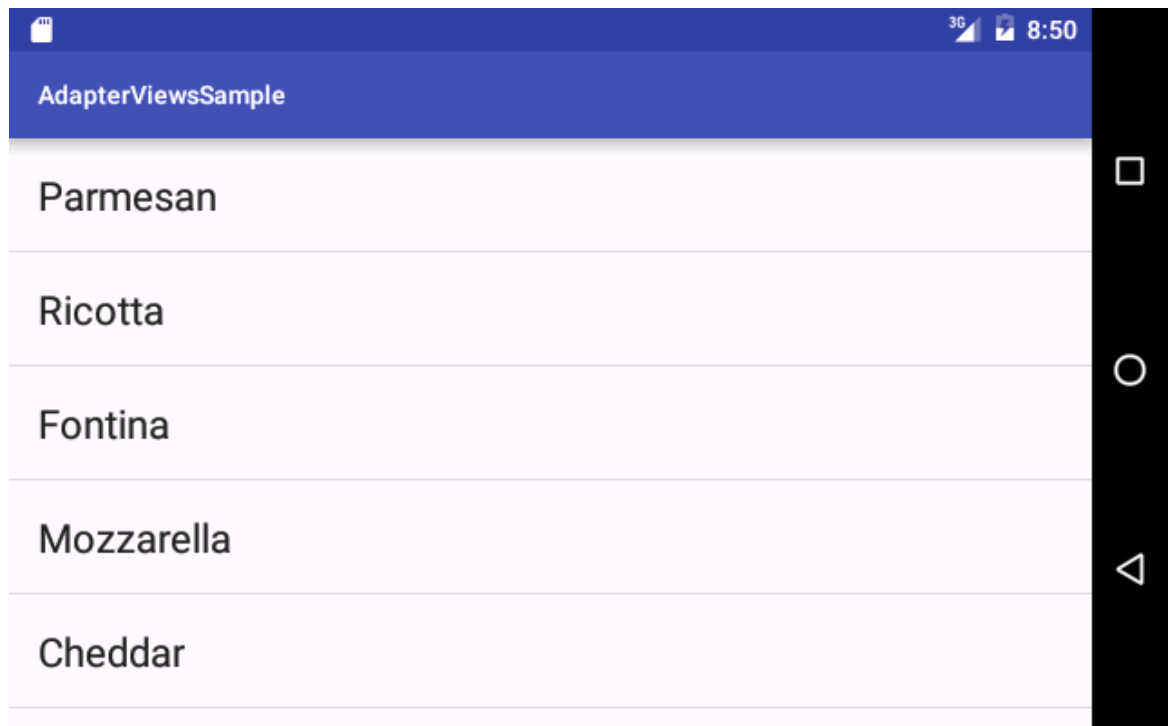
Как правило, никакие другие виджеты пользовательского интерфейса не помещаются внутри макета, содержащего ListView. Поэтому передайте ListView методу setContentView() вашей деятельности, чтобы он занимал весь экран.

```
setContentView(cheeseList);
```

Чтобы связать ListView с адаптером, созданным нами на предыдущем шаге, вызовите метод setAdapter(), как показано ниже.

```
cheeseList.setAdapter(cheeseAdapter);
```

Если вы сейчас запустите свое приложение, вы сможете увидеть содержимое массива в виде списка.



5. Создание сетки

Чтобы отобразить вертикально прокручиваемую двумерную сетку элементов, вы можете использовать виджет `GridView`. И `ListView`, и `GridView` являются подклассами абстрактного класса `AbsListView`, и они имеют много общего. Поэтому, если вы знаете, как использовать его, вы также знаете, как использовать другие.

Используйте конструктор класса `GridView` для создания нового экземпляра и передайте его методу `setContentView()` вашей активности.

```
GridView cheeseGrid = new GridView(this);  
setContentView(cheeseGrid);
```

Чтобы установить количество столбцов в сетке, вызовите его метод `setNumColumns()`. Я собираюсь сделать это двухколоночной сеткой.

```
cheeseGrid.setNumColumns(2);
```

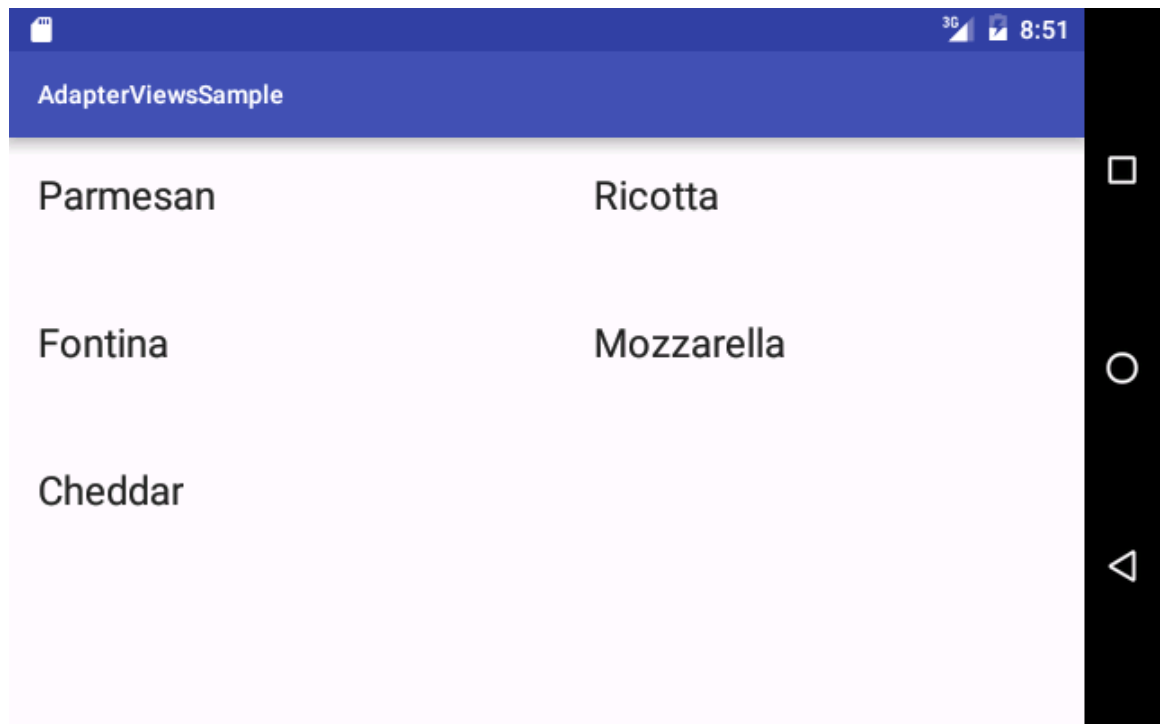
Обычно вы хотите настроить ширину столбцов и расстояние между ними, используя методы `setColumnWidth()`, `setVerticalSpacing()` и `setHorizontalSpacing()`. Обратите внимание, что эти методы используют пиксели в качестве единиц.

```
cheeseGrid.setColumnWidth(60);  
cheeseGrid.setVerticalSpacing(20);  
cheeseGrid.setHorizontalSpacing(20);
```

Теперь вы можете привязать `GridView` к адаптеру, который мы создали ранее, с помощью метода `setAdapter()`.

```
cheeseGrid.setAdapter(cheeseAdapter);
```

Запустите приложение еще раз, чтобы посмотреть, как выглядит `GridView`.



6. Добавление слушателей событий

Возможно прослушивание событий щелчка и длинных кликов на элементах внутри вида адаптера. В качестве примера добавим прослушиватель событий щелчка в GridView.

Создайте новый экземпляр анонимного класса, который реализует интерфейс `AdapterView.OnItemClickListener` и передает его методу `setOnItemClickListener()` объекта `GridView`. Android Studio автоматически создает заглушку для метода `onItemClick()` интерфейса. Вы заметите, что параметры метода включают целое число, определяющее позицию элемента списка. Вы можете использовать это целое число, чтобы узнать, какой элемент в наборе данных пользователь нажал.

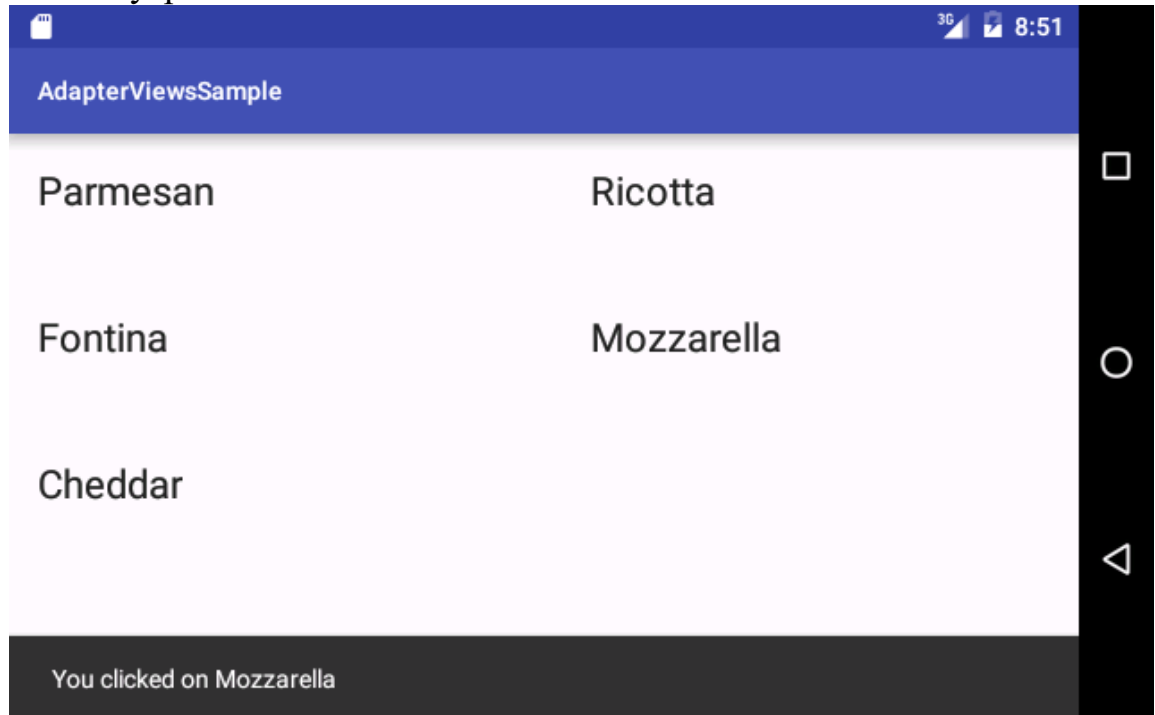
Следующий код иллюстрирует, как отображать простое сообщение в качестве зачатки каждый раз, когда элемент в `GridView` будет нажат.

```
cheeseGrid.setOnItemClickListener(new
AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> adapterView,
        View view, int position, long rowId) {

        // Generate a message based on the position
        String message = "You clicked on " + cheeses[position];

        // Use the message to create a Snackbar
        Snackbar.make(adapterView, message, Snackbar.LENGTH_LONG)
            .show(); // Show the Snackbar
    }
});
```

Если вы запустите приложение и щелкните любой элемент в сетке, в нижней части экрана появится сообщение. Обратите внимание, что вы можете использовать один и тот же код для прослушивания событий щелчка по элементам внутри ListView.



7. Расширение ArrayAdapter

ArrayAdapter может обрабатывать только один виджет TextView внутри макета создаваемых им объектов View. Чтобы расширить свои возможности, вы должны расширить его. Однако прежде чем мы это сделаем, давайте создадим несколько более сложный набор данных.

Вместо строк, скажем, наш набор данных содержит объекты следующего класса:

```
static class Cheese {  
    String name;  
    String description;  
  
    public Cheese(String name, String description) {  
        this.name = name;  
        this.description = description;  
    }  
}
```

Это набор данных, который мы будем использовать:

```
Cheese[] cheeses = {  
    new Cheese("Parmesan", "Hard, granular cheese"),  
    new Cheese("Ricotta", "Italian whey cheese"),  
    new Cheese("Fontina", "Italian cow's milk cheese"),  
    new Cheese("Mozzarella", "Southern Italian buffalo milk cheese"),  
    new Cheese("Cheddar", "Firm, cow's milk cheese"),  
};
```

Как вы можете видеть, класс Cheese содержит два поля, name и description. Чтобы отображать оба поля в списках или сетках, макет элементов должен содержать два виджета TextView.

Создайте новый XML-файл макета и назовите его custom_item.xml. Добавьте в него виджеты Large text и Small text. Установите атрибут id первого виджета на имя cheese_name, а второй - на cheese_description. Содержимое XML-файла макета должно выглядеть следующим образом:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="@dimen/activity_horizontal_margin">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="Large Text"
        android:id="@+id/cheese_name" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceSmall"
        android:text="Small Text"
        android:id="@+id/cheese_description" />
</LinearLayout>
```

ArrayAdapter также должен иметь возможность обрабатывать два виджета TextView. Измените свою деятельность, создайте новый анонимный класс, который расширяет класс ArrayAdapter и переопределяет его метод getView(). Убедитесь, что вы передаете массив как аргумент его конструктору.

```
ArrayAdapter<Cheese> cheeseAdapter =
    new ArrayAdapter<Cheese>(this, 0, cheeses) {
        @Override
        public View getView(int position,
            View convertView,
            ViewGroup parent) {

        }
    };
};
```

Внутри метода getView() вы должны использовать параметр position в качестве индекса массива и выбрать элемент в этом индексе.

```
Cheese currentCheese = cheeses[position];
```

Второй параметр метода getView() - это то, что позволяет нам повторно использовать объекты View. Если вы проигнорируете это,

производительность вашего адаптера будет плохой. Когда метод `getView()` вызывается в первый раз, `convertView` имеет значение `null`. Вы должны инициализировать его, изменяя файл ресурсов, который определяет макет элементов списка. Для этого обратитесь к макету `LayoutInflater` с помощью метода `getLayoutInflater()` и вызовите его метод `inflate()`.

```
// Inflate only once
if(convertView == null) {
    convertView = getLayoutInflater()
        .inflate(R.layout.custom_item, null, false);
}
```

На этом этапе вы можете использовать `findViewById()`, чтобы получить ссылку на виджеты `TextView` внутри макета и вызвать их методы `setText()` для их инициализации с использованием данных из массива.

```
TextView cheeseName =
    (TextView)convertView.findViewById(R.id.cheese_name);
TextView cheeseDescription =
    (TextView)convertView.findViewById(R.id.cheese_description);
```

```
cheeseName.setText(currentCheese.name);
cheeseDescription.setText(currentCheese.description);
```

Наконец, верните `convertView`, чтобы он мог использоваться для заполнения любого вида адаптера, связанного с адаптером.

```
1    return convertView;
```

8. Использование держателя вида

Метод `getView()` многократно вызывается видом адаптера для заполнения. Поэтому вы должны попытаться свести к минимуму количество операций, которые вы в нем выполняете.

На предыдущем шаге вы могли заметить, что, хотя мы убедились, что макет элементов списка заполняется только один раз, метод `findViewById()`, который потребляет много циклов процессора, вызывается каждый раз, когда метод вызывается `getView()`.

Чтобы избежать этого и улучшить производительность представления адаптера, нам нужно сохранить результаты метода `findViewById()` внутри объекта `convertView`. Для этого мы можем использовать объект `view holder`, который представляет собой не что иное, как объект класса, который может хранить виджеты, присутствующие в макете.

Поскольку в макете есть два виджета `TextView`, класс владельца представления должен также иметь два виджета `TextView`. Я назвал класс `ViewHolder`.

```
static class ViewHolder{
    TextView cheeseName;
    TextView cheeseDescription;
}
```

В методе `getView()` после раздувания макета вы можете теперь инициализировать объект держателя вида с помощью метода `findViewById()`.

```
ViewHolder viewHolder = new ViewHolder();
viewHolder.cheeseName =
    (TextView)convertView.findViewById(R.id.cheese_name);
viewHolder.cheeseDescription =
    (TextView)convertView.findViewById(R.id.cheese_description);
```

Чтобы сохранить объект-держатель вида в `convertView`, используйте его метод `setTag()`.

```
// Store results of findViewById
convertView.setTag(viewHolder);
```

И теперь, каждый раз, когда вызывается `getView()`, вы можете получить объект держателя вида из `convertView` с помощью метода `getTag()` и обновить виджеты `TextView` внутри него, используя их методы `setText()`.

```
TextView cheeseName =
    ((ViewHolder)convertView.getTag()).cheeseName;
TextView cheeseDescription =
    ((ViewHolder)convertView.getTag()).cheeseDescription;
```

```
cheeseName.setText(currentCheese.name);
cheeseDescription.setText(currentCheese.description);
```

Если вы запустите приложение сейчас, вы увидите, что `GridView` отображает две строки текста в каждой ячейке.

Заключение

В этом уроке вы узнали, как создать адаптер и использовать его для заполнения различных видов адаптеров. Вы также узнали, как создать свой собственный адаптер. Хотя мы ориентируемся только на классы `ArrayAdapter`, `ListView` и `GridView`, вы можете использовать те же методы для других адаптеров и адаптеров, которые предлагает `Android SDK`.

`Android Support Library` включает в себя класс `RecyclerView`. Он очень похож на вид адаптера, но не является подклассом класса `AdapterView`. Вы должны использовать его, если хотите создать более сложные списки, особенно те, которые используют несколько файлов макета для своих элементов.

ЛЕКЦИЯ №54-№55. ПОЛЬЗОВАТЕЛЬСКИЕ НАСТРОЙКИ. ИСПОЛЬЗОВАНИЕ SHAREDREFERENCES.

В приложениях часто содержатся настройки, которые позволяют пользователю изменять возможности и поведение приложения. Например, некоторые приложения позволяют пользователям включать и выключать уведомления или указывать частоту синхронизации данных приложения с облаком.

Если вы хотите предоставить настройки для вашего приложения, вы должны использовать API-интерфейсы Preference системы Android для построения интерфейса, согласованного с привычным для пользователей других приложений Android (включая системные настройки). В этом документе показано, как построить настройки вашего приложения посредством API-интерфейсов Preference.

Дизайн настроек

Подробную информацию о дизайне настроек см. в руководстве по дизайну настроек.

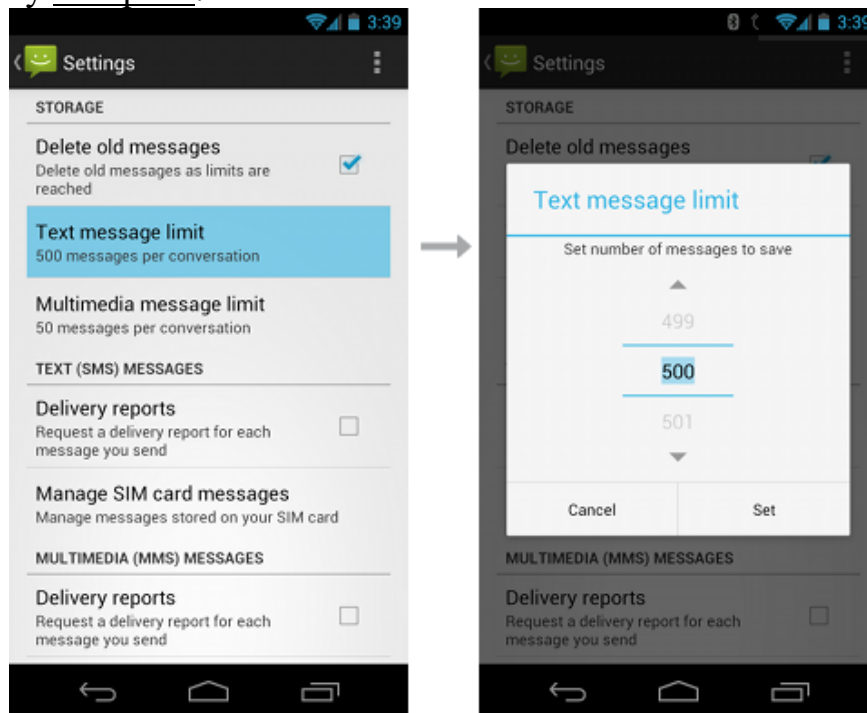


Рисунок 1. Снимки экранов настроек приложения Android для обмена сообщениями. Выбор элемента, заданного посредством Preference, открывает интерфейс для изменения значения.

Обзор

Вместо использования отображаемых объектов View для построения пользовательского интерфейса, настройки создаются с помощью различных подклассов класса Preference, который вы объявляете в XML-файле.

Объект Preference является строительным блоком для отдельной настройки. Каждый объект Preference отображается в виде элемента в списке и предоставляет соответствующий пользовательский интерфейс для изменения настройки пользователями. Например, CheckBoxPreference создает элемент списка, который показывает флажок, а ListPreference создает элемент, который открывает диалоговое окно со списком вариантов для выбора.

Каждый добавляемый вами объект Preference имеет соответствующую пару «ключ-значение», которую система использует для сохранения настройки в файле SharedPreferences значений настроек вашего приложения по умолчанию. Когда пользователь изменяет настройку, система обновляет соответствующее значение в файле SharedPreferences. Вам потребуется напрямую взаимодействовать с файлом, связанным с SharedPreferences,

только в случае, когда нужно прочитать значение для определения поведения вашего приложения на основе пользовательских настроек.

Значение, сохраненное в SharedPreferences для каждой настройки, может относиться к одному из следующих типов данных:

Логическое значение

Число с плавающей точкой

Целое число

Длинное целое число

Строка

Строка Set

Поскольку пользовательский интерфейс настроек вашего приложения создается посредством объектов Preference, а не объектов View, вам потребуется использовать специализированные подклассы Activity или Fragment для отображения настроек из списка:

Если ваше приложение поддерживает версии Android старше 3.0 (API уровня 10 и ниже), для построения операции необходимо наследовать класс PreferenceActivity.

В операционных системах Android 3.0 и более поздних версиях вы должны вместо этого использовать традиционный класс Activity, который содержит объект PreferenceFragment для отображения настроек вашего приложения. Однако, когда у вас есть несколько групп настроек, вы можете также использовать PreferenceActivity для создания макета с двумя панелями для больших экранов.

Настройка объекта PreferenceActivity и экземпляров PreferenceFragment описана в разделах Создание операции предпочтения и Использование фрагментов предпочтений.

Предпочтения

Каждая настройка для вашего приложения представлена конкретным подклассом класса Preference. Каждый подкласс содержит набор основных свойств, которые позволяют вам указывать, например, заголовок для настройки и ее значение по умолчанию. Каждый подкласс также содержит собственные специализированные свойства и пользовательский интерфейс. В качестве примера на рисунке 1 показан снимок экрана настроек приложения Android для обмена сообщениями. Каждый элемент списка на экране настроек возвращается отдельным объектом Preference.

Ниже приведены самые распространенные предпочтения:

CheckBoxPreference

Отображает элемент с флажком для настройки, которая может быть включена или выключена. Сохраненное значение является логическим (true, если флажок установлен).

ListPreference

Открывает диалоговое окно со списком переключателей. Сохраненное значение может относиться к одному из поддерживаемых типов значений (перечисленных выше).

EditTextPreference

Открывает диалоговое окно с виджетом EditText. Сохраненное значение — String.

См. класс Preference, который содержит список всех остальных подклассов и их соответствующих свойств.

Конечно, встроенные классы не обеспечивают всех потребностей, и вашему приложению может понадобиться что-либо более специализированное. Например, в настоящее время система не предоставляет класс Preference для выбора числа или даты. Поэтому вам может потребоваться определить свой собственный подкласс Preference. См. раздел Построение пользовательского предпочтения.

Определение предпочтений в XML

Хотя вы можете создавать новые экземпляры объектов Preference в режиме выполнения, вы должны определить список настроек в файле XML с иерархией объектов Preference. Использование файла XML для определения вашей коллекции настроек предпочтительней, поскольку файл обладает удобочитаемой структурой, которую легко обновлять. Кроме того, настройки вашего приложения обычно определены заранее, хотя у вас сохраняется возможность изменять коллекцию в режиме выполнения.

Каждый подкласс класса Preference может быть объявлен посредством элемента XML, который соответствует имени класса, например, `<CheckBoxPreference>`.

Вы должны сохранить файл XML в каталоге `res/xml/`. Хотя вы можете назвать файл любым именем, традиционно его называют `preferences.xml`. Обычно вам требуется лишь один файл, поскольку ветви иерархии (которые открывают собственный список настроек) объявлены с помощью вложенных экземпляров PreferenceScreen.

Примечание. Если вы хотите создать макет с несколькими панелями для ваших настроек, вам потребуются отдельные файлы XML для каждого фрагмента.

Корневой узел XML-файла должен быть элементом `<PreferenceScreen>`. Внутри этого элемента вы добавляете каждый элемент Preference. Каждый дочерний элемент, который вы добавляете внутри элемента `<PreferenceScreen>`, отображается в виде одного пункта в списке настроек.

Например:

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
  <CheckBoxPreference
    android:key="pref_sync"
    android:title="@string/pref_sync"
    android:summary="@string/pref_sync_summ"
    android:defaultValue="true" />
  <ListPreference
    android:dependency="pref_sync"
    android:key="pref_syncConnectionType"
```

```

android:title="@string/pref_syncConnectionType"
android:dialogTitle="@string/pref_syncConnectionType"
android:entries="@array/pref_syncConnectionTypes_entries"
android:entryValues="@array/pref_syncConnectionTypes_values"
android:defaultValue="@string/pref_syncConnectionTypes_default" />
</PreferenceScreen>

```

В этом примере есть CheckBoxPreference и ListPreference. Оба содержат следующие три атрибута:

`android:key`

Этот атрибут необходим для предпочтений, которые сохраняют значение данных. Он задает уникальный ключ (строку), который использует система при сохранении значения этой настройки в SharedPreferences.

Этот атрибут не является обязательным только когда предпочтение представляет собой PreferenceCategory или PreferenceScreen, либо предпочтение указывает намерение Intent для вызова (посредством элемента `<intent>`) или фрагмент Fragment для отображения (с помощью атрибута `android:fragment`).

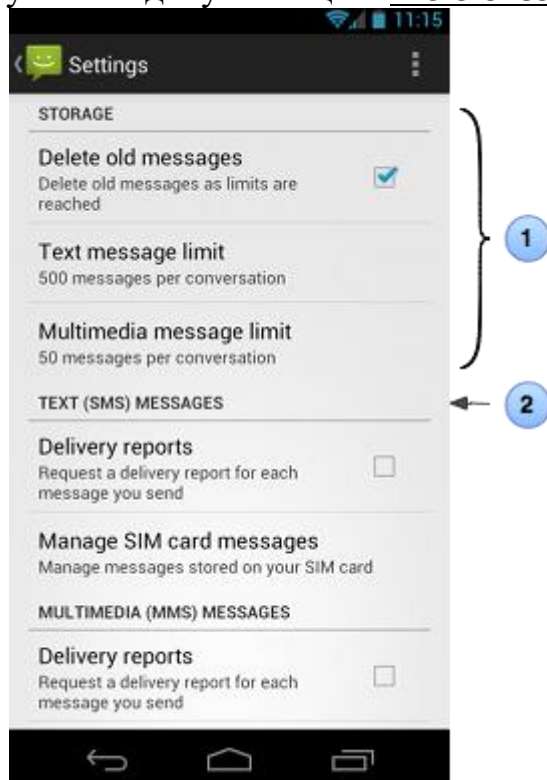
`android:title`

Этот атрибут предоставляет имя настройки, отображаемое для пользователя.

`android:defaultValue`

Этот атрибут указывает исходное значение, которое система должна установить в файле SharedPreferences. Вы должны указать значения по умолчанию для всех настроек.

Для получения информации обо всех других поддерживаемых атрибутах см. документацию Preference (и соответствующий подкласс).



- Рисунок 2. Создание категорий с заголовками.
1. Категория задана элементом `<PreferenceCategory>`.
 2. Заголовок задан посредством атрибута `android:title`.

Когда список ваших настроек содержит более 10 элементов, вы, вероятно, захотите добавить заголовки для определения групп настроек или отобразить эти группы на отдельном экране. Эти возможности описаны в следующих разделах.

Создание групп настроек

Если вы представляете список из 10 или более настроек, пользователям может быть трудно их просматривать, воспринимать и обрабатывать. Это можно исправить, разделив некоторые или все настройки на группы, что эффективно преобразует один длинный список в несколько более коротких списков. Группа связанных настроек может быть представлена одним из двух способов:

Использование заголовков

Использование подэкранов

Вы можете пользоваться одним или обоими из этих методов группировки для организации настроек в вашем приложении. Принимая решение об используемом варианте и о разделении настроек на группы, вы должны следовать инструкциям в разделе Настройки руководства «Дизайн для Android».

Использование заголовков

Если вы хотите создать разделители с заголовками между группами настроек (как показано на рисунке 2), поместите каждую группу объектов `Preference` внутри `PreferenceCategory`.

Например:

```
<PreferenceScreen
xmlns:android="http://schemas.android.com/apk/res/android">
  <PreferenceCategory
    android:title="@string/pref_sms_storage_title"
    android:key="pref_key_storage_settings">
    <CheckBoxPreference
      android:key="pref_key_auto_delete"
      android:summary="@string/pref_summary_auto_delete"
      android:title="@string/pref_title_auto_delete"
      android:defaultValue="false"... />
    <Preference
      android:key="pref_key_sms_delete_limit"
      android:dependency="pref_key_auto_delete"
      android:summary="@string/pref_summary_delete_limit"
      android:title="@string/pref_title_sms_delete"... />
    <Preference
      android:key="pref_key_mms_delete_limit"
      android:dependency="pref_key_auto_delete"
      android:summary="@string/pref_summary_delete_limit"
```

```

        android:title="@string/pref_title_mms_delete" ... />
    </PreferenceCategory>
    ...
</PreferenceScreen>

```

Использование подэкранов

Если вы хотите поместить группу настроек на подэкран (как показано на рисунке 3), поместите каждую группу объектов Preference внутри PreferenceScreen.

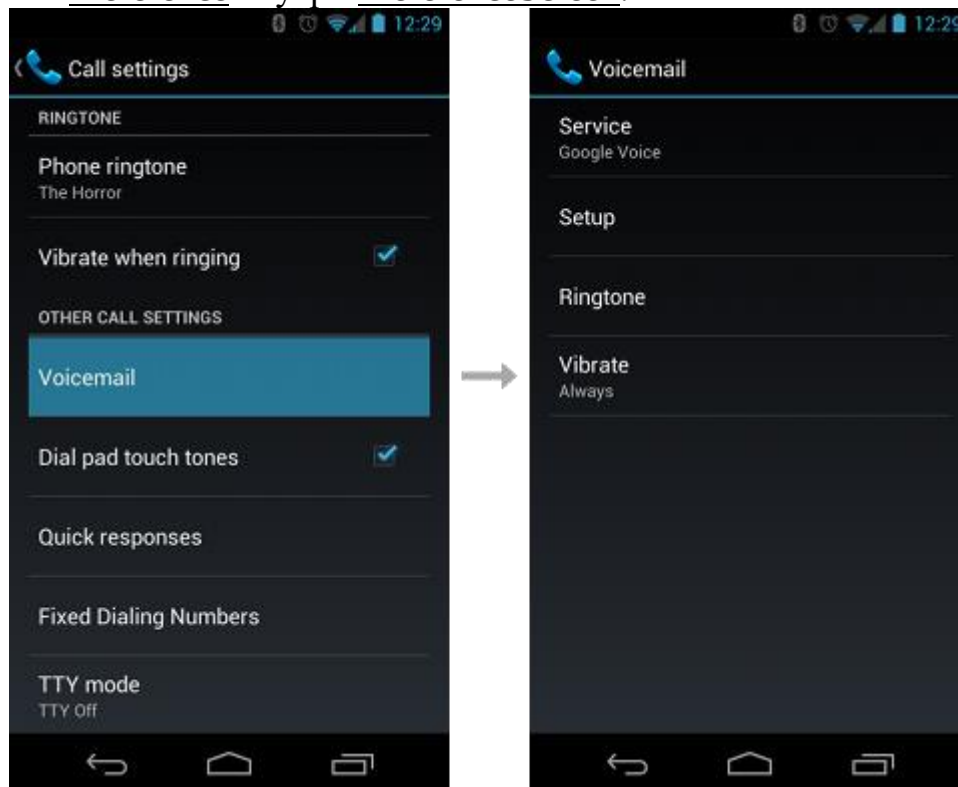


Рисунок 3. Создание подэкранов. Элемент `<PreferenceScreen>` создает пункт, при выборе которого открывается отдельный список вложенных настроек.

Например:

```

<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/and
roid">
    <!-- opens a subscreen of settings -->
    <PreferenceScreen
        android:key="button_voicemail_category_key"
        android:title="@string/voicemail"
        android:persistent="false">
        <ListPreference
            android:key="button_voicemail_provider_key"
            android:title="@string/voicemail_provider" ... />
        <!-- opens another nested subscreen -->
        <PreferenceScreen
            android:key="button_voicemail_setting_key"
            android:title="@string/voicemail_settings"

```

```

        android:persistent="false">
        ...
    </PreferenceScreen>
    <RingtonePreference
        android:key="button_voicemail_ringtone_key"
        android:title="@string/voicemail_ringtone_title"
        android:ringtoneType="notification" ... />
    ...
</PreferenceScreen>
...
</PreferenceScreen>

```

Использование намерений

В некоторых случаях может потребоваться, чтобы элемент предпочтений открывал другую операцию, а не экран настроек, например, веб-браузер для просмотра веб-страницы. Чтобы вызвать Intent, когда пользователь выбирает элемент предпочтений, добавьте элемент `<intent>` в качестве дочернего элемента соответствующего элемента `<Preference>`.

Например, здесь показано использование элемента предпочтений для открытия веб-страницы:

```

<Preference
    android:title="@string/prefs_web_page"
    <intent
        android:action="android.intent.action.VIEW"
        android:data="http://www.example.com"
    />
</Preference>

```

Вы можете создавать неявные и явные намерения с помощью следующих атрибутов:

`android:action`

Назначаемое действие, как в методе `setAction()`.

`android:data`

Назначаемые данные, как в методе `setData()`.

`android:mimeType`

Назначаемый тип MIME, как в методе `setType()`.

`android:targetClass`

Часть имени компонента, означающая класс, как в методе `setComponent()`.

`android:targetPackage`

Пакетная часть имени компонента, как в методе `setComponent()`.

Создание операции предпочтений

Для отображения ваших настроек в операции наследуйте класс `PreferenceActivity`. Это наследование традиционного класса `Activity`, который отображает список настроек на основе иерархии объектов `Preference`. `PreferenceActivity` автоматически сохраняет настройки, связанные с каждым объектом `Preference`, когда пользователь вносит изменения.

Примечание. При разработке приложения для версии Android 3.0 или выше вместо этого следует использовать PreferenceFragment. Прочитайте следующий раздел Использование фрагментов предпочтений.

Запомните самое важное: не загружайте макет отображаемых объектов во время обратного вызова onCreate(). Вместо этого вызовите addPreferencesFromResource() для добавления предпочтений, объявленных в XML-файле для операции. Например, здесь приведен минимальный код, необходимый для работы PreferenceActivity:

```
public class SettingsActivity extends PreferenceActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        addPreferencesFromResource(R.xml.preferences);
    }
}
```

Этого кода действительно достаточно для некоторых приложений, поскольку как только пользователь изменяет предпочтение, система сохраняет изменения в файле SharedPreferences по умолчанию, который другие компоненты вашего приложения могут читать, когда требуется проверить пользовательские настройки. Однако многим приложениям требуется немного больше кода, чтобы отслеживать изменения, происходящие с предпочтениями. Информацию об отслеживании изменений в файле SharedPreferences см. в разделе Чтение предпочтений.

Использование фрагментов предпочтений

При разработке приложений для Android 3.0 (API уровня 11) и более поздних версий необходимо использовать PreferenceFragment для отображения списка объектов Preference. Вы можете добавить PreferenceFragment в любую операцию, при этом необязательно использовать PreferenceActivity.

Фрагменты обеспечивают более универсальную архитектуру для вашего приложения по сравнению с использованием отдельных операций, вне зависимости от типа создаваемой операции. Фактически, для управления отображением ваших настроек мы предлагаем вам использовать PreferenceFragment вместо PreferenceActivity при каждой возможности.

Ваша реализация PreferenceFragment может содержать просто определение метода onCreate() для загрузки файла предпочтений посредством addPreferencesFromResource(). Например:

```
public static class SettingsFragment extends PreferenceFragment {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Load the preferences from an XML resource
        addPreferencesFromResource(R.xml.preferences);
    }
}
```

```

    }
    ...
}

```

Затем вы можете добавить этот фрагмент в операцию Activity, как вы сделали бы это для любого другого фрагмента Fragment. Например:

```

public class SettingsActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Display the fragment as the main content.
        getFragmentManager().beginTransaction()
            .replace(android.R.id.content, new SettingsFragment())
            .commit();
    }
}

```

Примечание. Фрагмент PreferenceFragment не содержит собственного объекта Context. Если вам требуется объект Context, вы можете вызвать getActivity(). Однако разработчик должен быть внимательным и вызывать метод getActivity() только в том случае, когда фрагмент прикреплен к операции. Если фрагмент еще не прикреплен или был откреплён в конце его жизненного цикла, метод getActivity() вернет null.

Установка значений по умолчанию

Вероятно, создаваемые вами предпочтения определяют важное поведение вашего приложения, поэтому необходимо инициализировать соответствующий файл SharedPreferences, записав в него значения по умолчанию для каждого предпочтения Preference при первом запуске вашего приложения пользователем.

В первую очередь необходимо указать значение по умолчанию для каждого объекта Preference в вашем XML-файле посредством атрибута android:defaultValue. Значение может относиться к любому типу данных, подходящему для соответствующего объекта Preference. Например:

```

<!-- default value is a boolean -->
<CheckBoxPreference
    android:defaultValue="true"
    ... />

<!-- default value is a string -->
<ListPreference
    android:defaultValue="@string/pref_syncConnectionTypes_default"
    ... />

```

Затем из метода onCreate() основной операции вашего приложения (и из любой другой операции, через которую пользователь может войти в ваше приложение в первый раз) вызовите setDefaultValues():

`PreferenceManager.setDefaultValues(this, R.xml.advanced_preferences, false);`

Вызов этого метода при выполнении `onCreate()` гарантирует, что ваше приложение правильно инициализируется и получит настройки по умолчанию, которые могут потребоваться вашему приложению для определенного поведения (например, следует ли загружать данные при работе в сотовой сети).

Этот метод имеет три аргумента:

Context вашего приложения.

Идентификатор ресурса для XML-файла предпочтений, для которого вы хотите установить значения по умолчанию.

Логическое значение, которое указывает, требуется ли значения по умолчанию устанавливать более одного раза.

При значении `false` система устанавливает значения по умолчанию только в том случае, если этот метод никогда не вызывался ранее (или атрибут `KEY_HAS_SET_DEFAULT_VALUES` в файле общих предпочтений по умолчанию имеет значение `false`).

Когда для третьего аргумента установлено значение `false`, вы можете вызывать этот метод при каждом запуске операции, не опасаясь перезаписи сохраненных пользовательских предпочтений из-за их сброса в состояние по умолчанию. Однако, если установить для этого аргумента значение `true`, вы будете перезаписывать все предыдущие значения значениями по умолчанию.

Использование заголовков предпочтений

В редких случаях может потребоваться такая структура настроек, при которой на первом экране отображается только список подэкранов (например, как в приложении системных настроек, показанных на рисунках 4 и 5). При разработке такого дизайна для Android 3.0 и более поздних версий вы должны использовать новую возможность Android 3.0 — «заголовки», вместо создания подэкранов посредством вложенных элементов PreferenceScreen.

Чтобы создать настройки с заголовками, выполните следующие действия:

Выделите каждую группу настроек в отдельный экземпляр PreferenceFragment. Таким образом, каждая группа настроек должна иметь отдельный XML-файл.

Создайте XML-файл заголовков, в котором перечислены все группы настроек и объявления, какой фрагмент содержит соответствующий список настроек.

Наследуйте класс PreferenceActivity, который будет содержать ваши настройки.

Реализуйте обратный вызов onBuildHeaders(), чтобы указать файл заголовков.

Огромное преимущество использования этого дизайна состоит в том, что при запуске на больших экранах PreferenceActivity автоматически создает макет с двумя панелями, показанный на рисунке 4.

Даже если ваше приложение поддерживает версии Android старше 3.0, вы можете создать приложение, использующее PreferenceFragment для двухпанельного представления на новых устройствах и поддерживающее традиционную многоэкранную иерархию на более старых устройствах (см. раздел Поддержка старых версий посредством заголовков предпочтений).

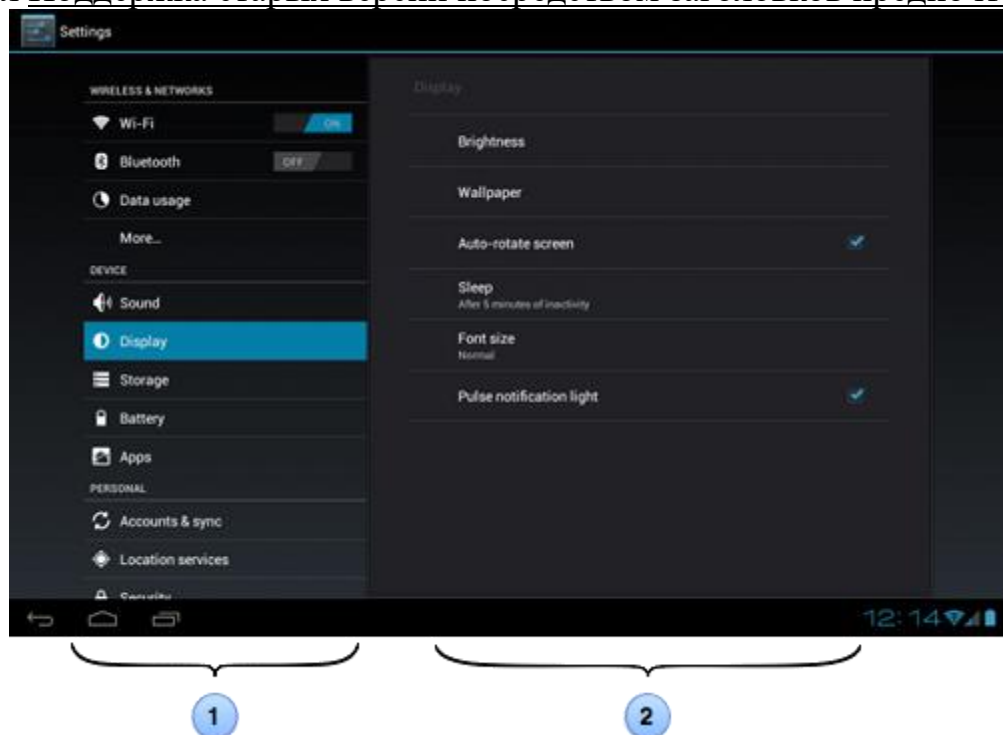


Рисунок 4. Двухпанельный макет с заголовками.

1. Заголовки определяются посредством XML-файла заголовков.
2. Каждая группа настроек определяется с помощью фрагмента PreferenceFragment, который указывается элементом `<header>` в файле заголовков.

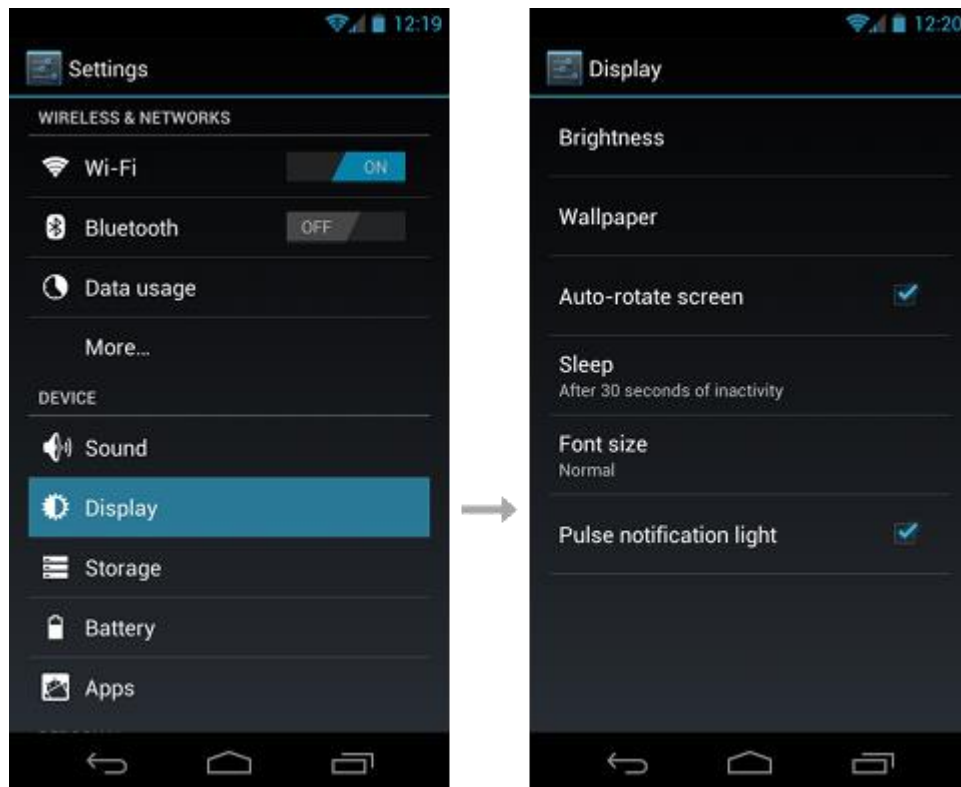


Рисунок 5. Смартфон с заголовками настроек. При выборе пункта вместо заголовков отображается соответствующий фрагмент PreferenceFragment.

Создание файла заголовков

Каждая группа настроек в вашем списке заголовков указывается отдельным элементом `<header>` внутри корневого элемента `<preference-headers>`. Например:

```
<?xml version="1.0" encoding="utf-8"?>
<preference-headers
xmlns:android="http://schemas.android.com/apk/res/android">
  <header
    android:fragment="com.example.prefs.SettingsActivity$SettingsFragmentOne"
    android:title="@string/prefs_category_one"
    android:summary="@string/prefs_summ_category_one" />
  <header
    android:fragment="com.example.prefs.SettingsActivity$SettingsFragmentTwo"
    android:title="@string/prefs_category_two"
    android:summary="@string/prefs_summ_category_two" >
    <!-- key/value pairs can be included as arguments for the fragment. -->
    <extra android:name="someKey" android:value="someHeaderValue"
  />
</header>
</preference-headers>
```

Посредством атрибута `android:fragment` каждый заголовок объявляет экземпляр фрагмента `PreferenceFragment`, который должен открываться при выборе этого заголовка пользователем.

Элемент `<extras>` позволяет передавать пары «ключ-значение» фрагменту в объекте `Bundle`. Фрагмент может извлекать аргументы путем вызова метода `getArguments()`. Вы можете передавать аргументы фрагменту по различным причинам, но хорошим поводом является повторное использование одного и того же подкласса `PreferenceFragment` для каждой группы и использование аргументов для указания XML-файла предпочтений, который должен быть загружен фрагментом.

Например, здесь приведен фрагмент, который можно использовать повторно для нескольких групп настроек, когда каждый заголовок определяет аргумент `<extra>` с ключом "settings":

```
public static class SettingsFragment extends PreferenceFragment {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        String settings = getArguments().getString("settings");
        if ("notifications".equals(settings)) {
            addPreferencesFromResource(R.xml.settings_wifi);
        } else if ("sync".equals(settings)) {
            addPreferencesFromResource(R.xml.settings_sync);
        }
    }
}
```

Отображение заголовков

Чтобы отобразить заголовки предпочтений, вы должны реализовать метод обратного вызова `onBuildHeaders()` и вызвать `loadHeadersFromResource()`. Например:

```
public class SettingsActivity extends PreferenceActivity {
    @Override
    public void onBuildHeaders(List<Header> target) {
        loadHeadersFromResource(R.xml.preference_headers, target);
    }
}
```

Когда пользователь выбирает пункт в списке заголовков, система открывает связанный `PreferenceFragment`.

Примечание. При использовании заголовков предпочтений ваш подкласс `PreferenceActivity` не должен реализовывать метод `onCreate()`, поскольку единственной обязательной задачей операции является загрузка заголовков.

Поддержка старых версий с заголовками предпочтений

Если ваше приложение поддерживает версии Android старше 3.0, вы можете использовать заголовки для предоставления двухпанельного макета

при работе на Android 3.0 или более поздней версии. Достаточно создать дополнительный XML-файл настроек, использующий базовые элементы `<Preference>`, которые ведут себя аналогично пунктам заголовка (для использования в более старых версиях Android).

Вместо открытия новых экранов `PreferenceScreen` каждый из элементов `<Preference>` отправляет намерение `Intent` в `PreferenceActivity` с указанием XML-файла предпочтений для загрузки.

В качестве примера приведен XML-файл для заголовков предпочтений, который используется в Android версии 3.0 и более поздних версий (`res/xml/preference_headers.xml`):

```
<preference-headers
xmlns:android="http://schemas.android.com/apk/res/android">
  <header
    android:fragment="com.example.prefs.SettingsFragmentOne"
    android:title="@string/prefs_category_one"
    android:summary="@string/prefs_summ_category_one" />
  <header
    android:fragment="com.example.prefs.SettingsFragmentTwo"
    android:title="@string/prefs_category_two"
    android:summary="@string/prefs_summ_category_two" />
</preference-headers>
```

А здесь представлен файл предпочтений, который содержит те же самые заголовки для версий старше Android 3.0 (`res/xml/preference_headers_legacy.xml`):

```
<PreferenceScreen
xmlns:android="http://schemas.android.com/apk/res/android">
  <Preference
    android:title="@string/prefs_category_one"
    android:summary="@string/prefs_summ_category_one" >
    <intent
      android:targetPackage="com.example.prefs"
      android:targetClass="com.example.prefs.SettingsActivity"
      android:action="com.example.prefs.PREFS_ONE" />
  </Preference>
  <Preference
    android:title="@string/prefs_category_two"
    android:summary="@string/prefs_summ_category_two" >
    <intent
      android:targetPackage="com.example.prefs"
      android:targetClass="com.example.prefs.SettingsActivity"
      android:action="com.example.prefs.PREFS_TWO" />
  </Preference>
</PreferenceScreen>
```

Так как поддержка `<preference-headers>` была добавлена в версии Android 3.0, система вызывает `onBuildHeaders()` в

методе PreferenceActivity только при работе в Android версии 3.0 или более поздней версии. Чтобы загрузить «старый» файл заголовков (preference_headers_legacy.xml), вы должны проверить версию Android и, если версия старше Android 3.0 (HONEYCOMB), вызвать addPreferencesFromResource() для загрузки старого файла заголовков. Например:

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    ...

    if (Build.VERSION.SDK_INT <
Build.VERSION_CODES.HONEYCOMB) {
        // Load the legacy preferences headers
        addPreferencesFromResource(R.xml.preference_headers_legacy);
    }
}

// Called only on Honeycomb and later
@Override
public void onBuildHeaders(List<Header> target) {
    loadHeadersFromResource(R.xml.preference_headers, target);
}

Остается обработать намерение Intent, переданное в операцию, чтобы
идентифицировать файл предпочтений для загрузки. Поэтому извлеките
операцию намерения и сравните ее с известными строками действия, которые
вы использовали в тегах <intent> XML-файла предпочтений:
    final static String ACTION_PREFS_ONE =
"com.example.prefs.PREFS_ONE";
    ...

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    String action = getIntent().getAction();
    if (action != null && action.equals(ACTION_PREFS_ONE)) {
        addPreferencesFromResource(R.xml.preferences);
    }
    ...

    else if (Build.VERSION.SDK_INT <
```

```

    }
}

```

При этом помните, что последующие вызовы `addPreferencesFromResource()` будут помещать все предпочтения в один список, поэтому обязательно используйте операторы `else-if`, чтобы обеспечить только однократный вызов метода при изменении условий.

Чтение предпочтений

По умолчанию все предпочтения вашего приложения сохраняются в файле, который доступен из любого места вашего приложения посредством вызова статического метода `PreferenceManager.getDefaultSharedPreferences()`. Он возвращает объект `SharedPreferences`, содержащий все пары «ключ-значение», связанные с объектами `Preference`, использованными в вашей операции `PreferenceActivity`.

В качестве примера показано чтение одного из значений предпочтений из любой другой операции в вашем приложении:

```

SharedPreferences                sharedPref                =
PreferenceManager.getDefaultSharedPreferences(this);
String                            syncConnPref              =
sharedPref.getString(SettingsActivity.KEY_PREF_SYNC_CONN, "");

```

Отслеживание изменений предпочтений

Существует несколько причин, по которым вы можете захотеть получать уведомления, как только пользователь изменяет одно из предпочтений. Чтобы получать обратный вызов при изменении любого из предпочтений, реализуйте интерфейс `SharedPreferences.OnSharedPreferenceChangeListener` и зарегистрируйте приемник для объекта `SharedPreferences` посредством вызова `registerOnSharedPreferenceChangeListener()`.

Этот интерфейс содержит только один метод обратного вызова, `onSharedPreferenceChanged()`, и вы, вероятно, сочтете его самым простым способом реализации интерфейса в составе своей операции. Например:

```

public class SettingsActivity extends PreferenceActivity
    implements OnSharedPreferenceChangeListener {
    public static final String KEY_PREF_SYNC_CONN =
"pref_syncConnectionType";
    ...

    public void onSharedPreferenceChanged(SharedPreferences
sharedPreferences,
String key) {
    if (key.equals(KEY_PREF_SYNC_CONN)) {
        Preference connectionPref = findPreference(key);
        // Set summary to be the user-description for the selected value
        connectionPref.setSummary(sharedPreferences.getString(key, ""));
    }
}

```

```
    }  
}
```

В этом примере метод проверяет, выполнено ли изменение настройки для известного ключа предпочтений. Он вызывает `findPreference()` для получения объекта `Preference`, который был изменен, поэтому он может изменить сводку пункта, описывающего выбор пользователя. То есть, когда настройка представляет собой `ListPreference` или другую настройку с несколькими вариантами выбора, при изменении этой настройки вы должны вызвать `setSummary()` для отображения текущего состояния (например, настройка спящего режима, показанная на рисунке 5).

Примечание. В соответствии с рекомендациями раздела Настройки руководства «Дизайн для Android», мы рекомендуем вам обновлять сводку для `ListPreference` при каждом изменении предпочтения пользователем, чтобы описать текущую настройку.

Для правильного управления жизненным циклом в операции мы рекомендуем вам регистрировать или отменять регистрацию вашего приемника `SharedPreferences.OnSharedPreferenceChangeListener` во время выполнения обратных вызовов `onResume()` и `onPause()` соответственно:

```
@Override  
protected void onResume() {  
    super.onResume();  
    getPreferenceScreen().getSharedPreferences()  
        .registerOnSharedPreferenceChangeListener(this);  
}  
  
@Override  
protected void onPause() {  
    super.onPause();  
    getPreferenceScreen().getSharedPreferences()  
        .unregisterOnSharedPreferenceChangeListener(this);  
}
```

Внимание! Когда вы вызываете приемник `registerOnSharedPreferenceChangeListener()`, диспетчер предпочтений не сохраняет строгую ссылку на приемник. Вы должны сохранить строгую ссылку на приемник, в противном случае она будет чувствительной к очистке памяти. Мы рекомендуем хранить ссылку на приемник в данных экземпляра объекта, который будет существовать, пока вам нужен приемник.

Например, в следующем коде вызывающий объект не сохраняет ссылку на приемник. В результате этого приемник будет удален при очистке памяти и через некоторое время приведет к сбою:

```
prefs.registerOnSharedPreferenceChangeListener(  
    // Bad! The listener is subject to garbage collection!  
    new SharedPreferences.OnSharedPreferenceChangeListener() {
```



```

    public void onSharedPreferenceChanged(SharedPreferences prefs, String
key) {
        // listener implementation
    }
});

```

Вместо этого сохраните ссылку на приемник в поле данных экземпляра объекта, который будет существовать, пока нужен приемник:

```

SharedPreferences.OnSharedPreferenceChangeListener listener =
    new SharedPreferences.OnSharedPreferenceChangeListener() {
        public void onSharedPreferenceChanged(SharedPreferences prefs, String
key) {
            // listener implementation
        }
    };
prefs.registerOnSharedPreferenceChangeListener(listener);

```

Контроль использования сети

Начиная с версии Android 4.0, системное приложение «Настройки» позволяет пользователям просматривать использование сетевых данных приложениями, работающими на переднем плане и в фоновом режиме. После этого пользователи могут отключить использование данных в фоновом режиме для отдельных приложений. Для того, чтобы пользователи не отключали доступ вашего приложения к данным в фоновом режиме, вы должны эффективно использовать подключение в режиме передачи данных и предоставить пользователям возможность настройки использования данных вашим приложением посредством настроек приложения.

Например, вы можете позволить пользователям управлять частотой синхронизации данных приложения, выполнением загрузки только в режиме подключения по Wi-Fi, использованием данных в роуминге и т. д. Когда эти возможности управления доступны, пользователи с меньшей вероятностью отключат доступ вашего приложения к данным, когда оно достигает установленных в системных настройках лимитов, поскольку вместо отключения они могут точно контролировать объем данных, который использует ваше приложение.

После добавления необходимых предпочтений в вашу операцию PreferenceActivity для управления поведением вашего приложения в отношении данных вы должны добавить фильтр намерений для ACTION_MANAGE_NETWORK_USAGE в вашем файле манифеста. Например:

```

<activity android:name="SettingsActivity" ... >
    <intent-filter>
        <action
android:name="android.intent.action.MANAGE_NETWORK_USAGE" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>

```

Этот фильтр манифеста указывает системе, что эта операция управляет использованием данных вашим приложением. Так, когда пользователь проверяет объем использованных приложением данных в системном приложении «Настройки», отображается кнопка Просмотреть настройки приложения, которая запускает вашу операцию PreferenceActivity, чтобы пользователь мог уточнить, сколько данных использует ваше приложение.

Построение пользовательского предпочтения

Система Android содержит множество подклассов Preference, которые позволяют вам строить пользовательский интерфейс для нескольких различных типов настроек. Тем не менее, вы можете обнаружить, что для нужной вам настройки нет встроенного решения, например, для выбора числа или даты. В таком случае вам потребуется создать нестандартное предпочтение путем наследования класса Preference или одного из других подклассов.

При наследовании класса Preference нужно выполнить несколько важных пунктов:

Укажите пользовательский интерфейс, который должен отображаться при выборе этой настройки пользователем.

При необходимости сохраните значение настройки.

Инициализируйте Preference текущим значением (или значением по умолчанию), когда предпочтение отображается.

Укажите значение по умолчанию в ответ на запрос системы.

Если Preference содержит свой собственный пользовательский интерфейс (например, диалоговое окно), сохраните и восстановите состояние для обработки изменений жизненного цикла (например, когда пользователь поворачивает экран).

В следующих разделах описано выполнение каждой из этих задач.

Указание пользовательского интерфейса

Если вы наследуете класс Preference непосредственно, вы должны реализовать метод onClick(), чтобы задать действие, происходящее при выборе пункта пользователем. Однако большая часть нестандартных настроек наследует DialogPreference, чтобы отобразить диалоговое окно, что упрощает процедуру. Когда вы наследуете DialogPreference, вы должны вызвать setDialogLayoutResources(), находясь в конструкторе класса, чтобы указать макет диалогового окна.

В качестве примера показан конструктор нестандартного диалогового окна DialogPreference, в котором объявляется макет и указывается текст для положительной и отрицательной кнопок диалога по умолчанию:

```
public class NumberPickerPreference extends DialogPreference {  
    public NumberPickerPreference(Context context, AttributeSet attrs) {  
        super(context, attrs);  
  
        setDialogLayoutResource(R.layout.numberpicker_dialog);  
        setPositiveButton(android.R.string.ok);  
        setNegativeButton(android.R.string.cancel);  
    }  
}
```

```

        setDialogIcon(null);
    }
    ...
}

```

Сохранение значения настройки

Вы можете сохранить значение настройки в любой момент, вызвав один из методов `persist*()` класса `Preference`, например, `persistInt()`, если настройка имеет целое значение, или `persistBoolean()` для сохранения логического значения.

Примечание. Каждое предпочтение `Preference` может сохранять только один тип данных, поэтому вы должны использовать метод `persist*()`, соответствующий типу данных, используемых вашим пользовательским предпочтением `Preference`.

Выбор метода сохранения настройки может зависеть от наследованного класса `Preference`. Если вы наследуете `DialogPreference`, вы должны сохранять значение только при закрытии диалога с положительным результатом (пользователь нажал кнопку «ОК»).

Когда `DialogPreference` закрывается, система вызывает метод `onDialogClosed()`. Этот метод содержит логический аргумент, который указывает, является ли результат пользователя «положительным» — если аргумент имеет значение `true`, значит пользователь выбрал положительную кнопку и вы должны сохранить новое значение. Например:

```

@Override
protected void onDialogClosed(boolean positiveResult) {
    // When the user selects "OK", persist the new value
    if (positiveResult) {
        persistInt(mNewValue);
    }
}

```

В этом примере `mNewValue` — это член класса, который содержит текущее значение настройки. При вызове `persistInt()` значение сохраняется в файл `SharedPreferences` (с автоматическим использованием ключа, указанного в XML-файле для этого предпочтения `Preference`).

Инициализация текущего значения

Когда система добавляет ваше предпочтение `Preference` на экран, она вызывает метод `onSetInitialValue()`, чтобы уведомить вас, имеет ли настройка сохраненное значение. Если сохраненного значения нет, этот вызов предоставляет вам значение по умолчанию.

Метод `onSetInitialValue()` передает логическое значение, `restorePersistedValue`, чтобы показать, было ли уже сохранено значение для настройки. Если значение равно `true`, вы должны извлечь сохраненное значение, вызвав один из методов `getPersisted*()` класса `Preference`, например, `getPersistedInt()` для целого значения. Обычно требуется извлечь сохраненное значение, чтобы

можно было правильно обновить пользовательский интерфейс для отражения ранее сохраненного значения.

Если `restorePersistedValue` имеет значение `false`, вы должны использовать значение по умолчанию, которое передается во втором аргументе.

```
@Override
protected void onSetInitialValue(boolean restorePersistedValue, Object
defaultValue) {
    if (restorePersistedValue) {
        // Restore existing state
        mCurrentValue = this.getPersistedInt(DEFAULT_VALUE);
    } else {
        // Set default state from the XML attribute
        mCurrentValue = (Integer) defaultValue;
        persistInt(mCurrentValue);
    }
}
```

Каждый метод `getPersisted*()` содержит аргумент, который указывает значение по умолчанию на случай, когда действительно нет сохраненного значения, или не существует ключ. В приведенном выше примере локальная константа служит для указания значения по умолчанию на случай, если `getPersistedInt()` не может вернуть сохраненное значение.

Внимание! Вы не можете использовать `defaultValue` в качестве значения по умолчанию в методе `getPersisted*()`, так как его значение всегда равно `null`, когда `restorePersistedValue` имеет значение `true`.

Предоставление значения по умолчанию

Если экземпляр вашего класса Preference указывает значение по умолчанию (с помощью атрибута `android:defaultValue`), система вызывает `onGetDefaultValue()`, когда она создает экземпляр объекта для извлечения значения. Вы должны реализовать этот метод, чтобы сохранить значение по умолчанию для системы в SharedPreferences. Например:

```
@Override
protected Object onGetDefaultValue(TypedArray a, int index) {
    return a.getInteger(index, DEFAULT_VALUE);
}
```

Аргументы метода предоставляют все необходимое: массив атрибутов и указатель положения `android:defaultValue`, который вы должны извлечь. Причина, по которой вы должны реализовать этот метод, чтобы извлечь значение по умолчанию из атрибута, состоит в том, что вы должны указать локальное значение по умолчанию для атрибута в случае, когда значение не определено.

Сохранение и восстановление состояния предпочтений

Как и View в макете, ваш подкласс Preference отвечает за сохранение и восстановление своего состояния в случае перезапуска операции или фрагмента (например, когда пользователь поворачивает экран). Чтобы

правильно сохранять и восстанавливать состояние вашего класса Preference, вы должны реализовать методы обратного вызова жизненного цикла onSaveInstanceState() и onRestoreInstanceState().

Состояние вашего Preference определяется объектом, который реализует интерфейс Parcelable. Система Android предоставляет вам такой объект в качестве начальной точки для определения вашего объекта состояния: класс Preference.BaseSavedState.

Чтобы определить, как ваш класс Preference сохраняет свое состояние, вы должны наследовать класс Preference.BaseSavedState. Вы должны переопределить лишь несколько методов и определить объект CREATOR.

Для большинства приложений вы можете скопировать следующую реализацию и просто изменить строки, которые обрабатывают value, если ваш подкласс Preference сохраняет типы данных, отличные от целых.

```
private static class SavedState extends BaseSavedState {
    // Member that holds the setting's value
    // Change this data type to match the type saved by your Preference
    int value;

    public SavedState(Parcelable superState) {
        super(superState);
    }

    public SavedState(Parcel source) {
        super(source);
        // Get the current preference's value
        value = source.readInt(); // Change this to read the appropriate data type
    }

    @Override
    public void writeToParcel(Parcel dest, int flags) {
        super.writeToParcel(dest, flags);
        // Write the preference's value
        dest.writeInt(value); // Change this to write the appropriate data type
    }

    // Standard creator object using an instance of this class
    public static final Parcelable.Creator<SavedState> CREATOR =
        new Parcelable.Creator<SavedState>() {

        public SavedState createFromParcel(Parcel in) {
            return new SavedState(in);
        }

        public SavedState[] newArray(int size) {
            return new SavedState[size];
        }
    }
}
```

```
    }
};
}
```

После добавления показанной выше реализации Preference.BaseSavedState в ваше приложение (обычно в качестве подкласса вашего подкласса Preference), вам потребуется реализовать методы onSaveInstanceState() и onRestoreInstanceState() для вашего подкласса Preference.

Например:

```
@Override
protected Parcelable onSaveInstanceState() {
    final Parcelable superState = super.onSaveInstanceState();
    // Check whether this Preference is persistent (continually saved)
    if (isPersistent()) {
        // No need to save instance state since it's persistent,
        // use superclass state
        return superState;
    }

    // Create instance of custom BaseSavedState
    final SavedState myState = new SavedState(superState);
    // Set the state's value with the class member that holds current
    // setting value
    myState.value = mNewValue;
    return myState;
}
```

```
@Override
protected void onRestoreInstanceState(Parcelable state) {
    // Check whether we saved the state in onSaveInstanceState
    if (state == null || !state.getClass().equals(SavedState.class)) {
        // Didn't save the state, so call superclass
        super.onRestoreInstanceState(state);
        return;
    }

    // Cast state to custom BaseSavedState and pass to superclass
    SavedState myState = (SavedState) state;
    super.onRestoreInstanceState(myState.getSuperState());

    // Set this Preference's widget to reflect the restored state
    mNumberPicker.setValue(myState.value);
}
```

ЛЕКЦИЯ №56-№57. РАБОТА С ГРАФИКОЙ. DRAWABLE И CANVAS.

Класс Drawable

Android предлагает двумерную графическую библиотеку рисования `android.graphics.drawable`. Класс `Drawable` является базовым классом для всех классов работы с графикой. Это общая абстракция для рисуемого объекта на холсте `Canvas`. Класс `Drawable` определяет разнообразные виды графики, включая:

- `AnimationDrawable` - покадровая анимация. XML-аналог `<animation-list>`
- `AnimatedRotateDrawable` - анимация вращения. XML-аналог `<animated-rotate>`
- `AnimatedVectorDrawable` - анимация векторных объектов (Android 21). XML-аналог `<animated-vector>`
- `AnimatedStateListDrawable`
- `BadgeDrawable`
- `BitmapDrawable` - `Bitmap` - изображение PNG и JPG, создание, растягивание, выравнивание раstra. XML-аналог `<bitmap>`
- `ChipDrawable`
- `ClipDrawable` - XML-аналог `<clip>`
- `ColorDrawable` - заполнение цветом. XML-аналог `<color>`
- `GradientDrawable` - создание градиентов
- `InsetDrawable` - XML-аналог `<inset>`
- `LayerDrawable` - многослойные картинки, контейнеры для дочерних элементов визуализации. XML-аналог `<layer-list>`
- `LevelListDrawable` - контейнер для одного объекта-потомка. Основана на целом числе, например, индикатор заряда или уровень силы сигнала. XML-аналог `<level-list>`
- `NinePathDrawable` - масштабируемое PNG-изображения `NinePath`. Используется в качестве фонового изображения для кнопок с изменяемыми размерами. XML-аналог `<nine-patch>`
- `PictureDrawable`
- `RippleDrawable` - анимационный эффект ряби (Android 21). XML-аналог `<ripple>`
- `RotateDrawable` - XML-аналог `<rotate>`
- `RoundedBitmapDrawable` - создание круглых изображений (из библиотеки совместимости)
- `ScaleDrawable` - контейнер для одного объекта-потомка. Изменяет размер на основе текущего заданного значения. Используется для создания выювера картинок с возможностью увеличения. XML-аналог `<scale>`
- `ShapeDrawable` - `Shape` - команды рисования векторных объектов, основанные на `Path`. XML-аналог `<shape>`
- `StateListDrawable` - специальный контейнер. Используют для создания различных вариантов выделения кнопок и установки фокуса на них. XML-аналог `<selector>`

- `TransitionDrawable` - плавные переходы и др. XML-аналог `<transition>`
- `VectorDrawable` - векторные изображения (Android 21). Аналог `<vector>`.

Если вы придумаете свой XML-тег типа `<kitty>` или сделаете опечатку, то получите исключение `Resources.NotFoundException`.

Есть два способа определить и инициализировать объект `Drawable`:

- использовать готовые изображения, сохраненные в каталоге `res/drawable/`
- использовать XML-файл, в котором будут определены свойства объекта `Drawable`

Самый простой способ добавить графику в приложение — это использование готовых изображений из ресурсов. Наиболее предпочтительный тип PNG, также можно использовать JPG (приемлемо) и GIF (нежелательно). Обычно таким способом загружают значки и картинки.

Ресурсы изображений, помещенные в каталог `res/drawable/` во время компиляции проекта, могут быть автоматически оптимизированы со сжатием изображения утилитой `aapt`. Если требуется загружать растровые изображения BMP, которые утилита `aapt` обязательно оптимизирует, без сжатия, поместите их в каталог `res/raw/`, где они не будут оптимизироваться. Однако потребуются загрузка с использованием методов потокового ввода-вывода и последующая конвертация графики в растровый рисунок.

В отличие от `View` объект `Drawable` не нуждается в разметке и размерах контейнера. Его задача - просто рисовать.

Размеры самого объекта `Drawable` можно узнать с помощью методов `getIntrinsicWidth()` и `getIntrinsicHeight()`. Также есть метод `getBounds()`, возвращающий ограничивающий прямоугольник.

С помощью метода `setState()` можно управлять состоянием рисунка, например, менять цвет при касании экрана. Метод `setLevel()` может задать уровень вывода картинки.

Объект `Drawable` можно создать с помощью XML в виде ресурсов и сохранить в папках со словом `drawable`. А затем присвоить, например объекту `ImageView` с помощью атрибута `android:src="@drawable/ic_launcher"`.

Программно такого же эффекта можно достичь через код.

```
imageView.setImageResource(R.drawable.ic_launcher);
```

Создание объектов `Drawable` из ресурсов

В других случаях вы можете захотеть обработать ваш ресурс изображения как объект `Drawable`. Чтобы это сделать, создайте `Drawable`, загрузив изображение из ресурсов примерно так:

```
Resources res = mContext.getResources();
```

```
Drawable drawable = res.getDrawable(R.drawable.ic_launcher);
```

Каждый уникальный ресурс в вашем проекте может поддерживать только одно состояние независимо от того, сколько различных объектов вы можете инициализировать в программном коде для этого ресурса. Например,

если вы инициализируете два объекта Drawable от одного ресурса изображения, а затем измените свойство, например alpha (прозрачность), для одного из объектов Drawable, другой объект также изменит это свойство.

Напишем простой пример и попробуем получить объект от одного и того же значка.

```
Resources res = getResources();
Drawable drawable1 = res.getDrawable(R.drawable.ic_launcher);
Drawable drawable2 = res.getDrawable(R.drawable.ic_launcher);
Log.i("Drawable", "drawable1: " + drawable1);
Log.i("Drawable", "drawable2: " + drawable2);
```

Как ни странно, но результаты различаются. У меня вышло следующее:
drawable1: android.graphics.drawable.BitmapDrawable@b1080a68
drawable2: android.graphics.drawable.BitmapDrawable@b1080ad8

Метод getDrawable() всегда возвращает новый Drawable, даже если источник один и тот же. Тем не менее метод getConstantState() вернёт одинаковый результат.

```
Log.i("Drawable", "drawable1: " + drawable1.getConstantState());
Log.i("Drawable", "drawable2: " + drawable2.getConstantState());
Смотрим логи:
drawable1:
```

```
android.graphics.drawable.BitmapDrawable$BitmapState@b107e3b8
drawable2:
android.graphics.drawable.BitmapDrawable$BitmapState@b107e3b8
```

С помощью Drawable можно замостить экран активности, используя маленький Bitmap.

Конвертируем Drawable в Bitmap и выводим в ImageView

Предположим, у нас в ресурсах есть картинка (Drawable). Нам нужно конвертировать её в объект Bitmap и вывести в ImageView:

```
private ImageView mTargetImageView;

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    mTargetImageView =
    (ImageView)findViewById(R.id.imageViewTarget);

    // Загружаем растр из ресурсов
    Bitmap localBitmap = BitmapFactory.decodeResource(
        getApplicationContext().getResources(),
        R.drawable.ic_launcher);
    mTargetImageView.setImageBitmap(localBitmap);
    =
```

```
}
```

Программное создание Drawable

Drawable является абстрактным классом. Если вы собираетесь создавать класс на его основе, то необходимо реализовать следующие методы:

- draw(Canvas canvas) – рисовать на холсте
- getOpacity() - вернуть значение прозрачности Drawable.

Используются константы:

1. UNKNOWN – прозрачность неизвестна
 2. TRANSPARENT – Drawable будет полностью прозрачным
 3. TRANSLUCENT – Drawable будет состоять из прозрачных и непрозрачных участков
 4. OPAQUE – Drawable будет полностью непрозрачным
- setAlpha(int alpha) – установить прозрачность
 - setColorFilter(ColorFilter colorFilter) – установить цвет при помощи фильтра ColorFilter

Создадим свой вариант Drawable, рисующий шестиугольник (Источник для примера).

```
package ru.alexanderklimov.emptyactivity;
```

```
import android.graphics.Canvas;  
import android.graphics.ColorFilter;  
import android.graphics.Paint;  
import android.graphics.Path;  
import android.graphics.PixelFormat;  
import android.graphics.Rect;  
import android.graphics.drawable.Drawable;
```

```
public class HexagonDrawable extends Drawable {
```

```
    private Paint mPaint = new Paint(Paint.ANTI_ALIAS_FLAG);  
    private Path mPath = new Path();
```

```
    @Override  
    public void draw(Canvas canvas) {  
        canvas.drawPath(mPath, mPaint);  
    }
```

```
    @Override  
    public void setAlpha(int alpha) {  
        mPaint.setAlpha(alpha);  
    }
```

```
    @Override  
    public void setColorFilter(ColorFilter colorFilter) {
```

```

        mPaint.setColorFilter(colorFilter);
    }

    @Override
    public int getOpacity() {
        return PixelFormat.TRANSLUCENT;
    }

    @Override
    protected void onBoundsChange(Rect bounds) {
        super.onBoundsChange(bounds);
        int width = bounds.width();
        int height = bounds.height();
        mPath.reset();
        mPath.moveTo(0, height / 2);
        mPath.lineTo(width / 4, 0);
        mPath.lineTo(width * 3 / 4, 0);
        mPath.lineTo(width, height / 2);
        mPath.lineTo(width * 3 / 4, height);
        mPath.lineTo(width / 4, height);
        mPath.close();
    }
}

```

В методе draw() выводим контур mPath, который будет сформирован в другом методе, на холст.

В методе getOpacity() возвращаем TRANSLUCENT, т.к. у нас будет непрозрачный шестиугольник, а оставшееся пространство будет прозрачным.

В методах setAlpha() и setColorFilter() настроим кисть. Эти методы можно не реализовывать и оставить пустыми, объект будет работать и без них. Но если вы захотите сделать его полупрозрачным или применить фильтр, то методы следует реализовать.

Кроме четырёх обязательных методов пришлось реализовать метод onBoundsChange(), который вызывается при изменении размеров. Так как нам нужно нарисовать шестиугольник на всю область, мы должны знать его размер. Здесь мы получаем ширину и высоту Drawable и используем их для создания фигуры шестиугольника.

Применим созданный объект к ImageView:

```

ImageView imageView = (ImageView) findViewById(R.id.imageView);
imageView.setBackground(new HexagonDrawable());

```

Класс BitmapDrawable

Можно сконвертировать Bitmap в Drawable с помощью BitmapDrawable.

```

Drawable drawable = new BitmapDrawable(getResources(), tempBitmap);

```

Допустим, мы загружаем картинку (Bitmap) из Галереи и выводим его в ImageView

Возьмём готовый пример. Подготовим разметку.

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <Button
        android:id="@+id/loadimage"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Load Image 1" />

    <TextView
        android:id="@+id/sourceuri"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

    <ImageView
        android:id="@+id/result"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:adjustViewBounds="true"
        android:background="@android:color/background_dark"
        android:scaleType="centerInside" />
```

```
</LinearLayout>
```

И код.

```
package ru.alexanderklimov.test;

import java.io.FileNotFoundException;

import android.net.Uri;
import android.os.Bundle;
import android.app.Activity;
```

```

import android.content.Intent;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Canvas;
import android.graphics.drawable.BitmapDrawable;
import android.graphics.drawable.Drawable;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.TextView;

public class MainActivity extends Activity {

    Button btnLoadImage;
    TextView textSource;
    ImageView imageResult;

    final int RQS_IMAGE1 = 1;

    Uri source;
    Bitmap bitmapMaster;
    Canvas canvasMaster;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        btnLoadImage = (Button) findViewById(R.id.loadimage);
        textSource = (TextView) findViewById(R.id.sourceuri);
        imageResult = (ImageView) findViewById(R.id.result);

        btnLoadImage.setOnClickListener(new OnClickListener() {

            @Override
            public void onClick(View arg0) {
                Intent intent = new Intent(
                    Intent.ACTION_PICK,

                    android.provider.MediaStore.Images.Media.EXTERNAL_CONTENT_UR
I);

                startActivityForResult(intent, RQS_IMAGE1);
            }
        });
    }

```

```

    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent
data) {
        super.onActivityResult(requestCode, resultCode, data);

        Bitmap tempBitmap;

        if (resultCode == RESULT_OK) {
            switch (requestCode) {
                case RQS_IMAGE1:
                    source = data.getData();

                    try {
                        tempBitmap = BitmapFactory

.decodeStream(getContentResolver().openInputStream(
                                                                    source));

                        // Convert bitmap to drawable
                        Drawable drawable = new
BitmapDrawable(getResources(),
                                                                    tempBitmap);

                        imageResult.setImageDrawable(drawable);

                        textSource.setText(drawable.getClass().toString());

                    } catch (FileNotFoundException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                    }

                    break;
                }
            }
        }
    }

```

Класс ClipDrawable

Примеры с классом - ProgressBar, <clip> (класс ClipDrawable)

Класс ColorDrawable

Класс `ColorDrawable` позволяют указывать свойство изображения, основанное на единственном сплошном цвете. В качестве альтернативы можно использовать тег `<color>` в каталоге `res/drawable`:

```
<color xmlns:android="http://schemas.android.com/apk/res/android"
    android:color="#FF0000" />
```

Класс `GradientDrawable`

Класс `GradientDrawable` позволяет закрашивать градиентом кнопки, фон и так далее. Каждый градиент описывает плавный переход между двумя или тремя цветами с помощью линейного или радиального алгоритма, а также с помощью метода развёртки. В ресурсах `GradientDrawable` описывается в виде тега `<gradient>`, находясь внутри определения ресурса `ShapeDrawable` (`<shape>`).

Пример

Класс `TransitionDrawable`

Одна из возможностей использования объектов `Drawable` — загрузка картинок с плавными переходами. Создание переходов между изображениями с помощью класса `TransitionDrawable` лучше рассмотреть на примере.

Класс `ShapeDrawable`

Класс `ShapeDrawable`, входящий в пакет `android.graphics.drawable`, позволяет определять различные свойства рисунка с помощью различных методов, например `setAlpha()` — для установки прозрачности, `setColorFilter()` и т.д.

С объектами `ShapeDrawable` вы можете программно создавать любые примитивные формы и их стили. Для работы через XML используется тег `<shape>`. В библиотеке Android есть несколько классов, производных от базового класса `Shape`:

- `PathShape`;
- `RectShape`;
- `ArcShape`;
- `OvalShape`;
- `RoundRectShape`

Класс `RectShape`

Для рисования прямоугольника используется класс `RectShape`. Также необходимо установить цвет и границы фигуры. Если вы не установите границы, то прямоугольник не будет рисоваться. Если вы не установите цвет, фигура будет черной (значение по умолчанию).

Если задать высоту или ширину в 1-2 пиксела, то можно получить не прямоугольник, а вертикальную или горизонтальную линию:

```
ShapeDrawable line = new ShapeDrawable(new RectShape());
line.setIntrinsicHeight(2);
line.setIntrinsicWidth(150);
line.getPaint().setColor(Color.MAGENTA);
```

Класс `OvalShape`

Для рисования эллипсов используется класс `OvalShape`:

```
ShapeDrawable oval = new ShapeDrawable(new OvalShape());
oval.setIntrinsicHeight(100);
oval.setIntrinsicWidth(150);
oval.getPaint().setColor(Color.RED);
```

Класс `RoundRect`

Прорисовка прямоугольника с закругленными сторонами (`RoundRect`) несколько сложнее. Конструктор класса `RoundRect` для рисования прямоугольника с закругленными сторонами принимает несколько параметров:

`RoundRectShape(float[] outerRadii, RectF inset, float[] innerRadii)`

- `outerRadii` — массив из восьми значений радиуса закругленных сторон внешнего прямоугольника. Первые два значения для верхнего левого угла, остальные пары отсчитываются по часовой стрелке от первой пары. Если на внешнем прямоугольнике закруглений не будет, передается `null`;
- `inset` — объект класса `RectF`, который определяет расстояние от внутреннего прямоугольника до каждой стороны внешнего прямоугольника. Конструктор `RectF` принимает четыре параметра: пары `X` и `Y` координат левого верхнего и правого нижнего углов внутреннего прямоугольника. Если внутреннего прямоугольника нет, передается `null`;
- `innerRadii` — массив из восьми значений радиуса закруглений углов для внутреннего прямоугольника. Первые два значения — координаты верхнего левого угла, остальные пары отсчитываются по часовой стрелке от первой пары. Если на внутреннем прямоугольнике закруглений не будет, передается `null`.

```
float[] outR = new float[] {6, 6, 6, 6, 6, 6, 6, 6 };
RectF rectF = new RectF(8, 8, 8, 8);
float[] inR new float[] { 6, 6, 6, 6, 6, 6, 6, 6 };
```

```
ShapeDrawable shape = new ShapeDrawable(
    new RoundRectShape(outR, rectF, inR));
shape.setIntrinsicHeight(120);
shape.setIntrinsicWidth(150);
shape.getPaint().setColor(Color.BLUE);
```

Класс `Path`

Класс `Path` формирует множественный контур геометрических путей, состоящих из прямых линейных сегментов, квадратичных и кубических кривых. Для установки точек и перемещения линий (или кривых) используются методы `moveTo()` и `lineTo()`. Например, с помощью класса `Path` можно нарисовать пятиконечную звезду:

```
Path p = new Path();
p.moveTo(50, 0);
```



```

p.lineTo(25, 100);
p.lineTo(100,50);
p.lineTo(0,50);
p.lineTo(75,100);
p.lineTo(50,0);

```

```

ShapeDrawable d = new ShapeDrawable(new PathShape(p, 100, 100));
d.setIntrinsicHeight(100);
d.setIntrinsicWidth(100);
d.getPaint().setColor(Color.YELLOW);
d.getPaint().setStyle(Paint.Style.STROKE);

```

Класс ArcShape

Класс ArcShape создает фигуру в форме дуги. Принимает два параметра: угол начала прорисовки дуги в градусах и угловой размер дуги в градусах

```
ArcShape(float startAngle, float sweepAngle)
```

ScaleDrawable

Класс ScaleDrawable используется для масштабирования.

RotateDrawable

Класс RotateDrawable используется для поворотов.

LayerDrawable

LayerDrawable позволяет накладывать объекты Drawable друг на друга. Создав массив из полупрозрачных картинок, вы можете наложить их один поверх другого. Подготовьте три полупрозрачных изображения круга (red.png, blue.png, green.png) и поместите их в ресурсы res/drawable. Создадим фон для макета при помощи наложения этих элементов.

```
// В onCreate()
```

```
LinearLayout linear = (LinearLayout)findViewById(R.id.linear);
```

```
linear.setBackground(createLayerDrawable());
```

```

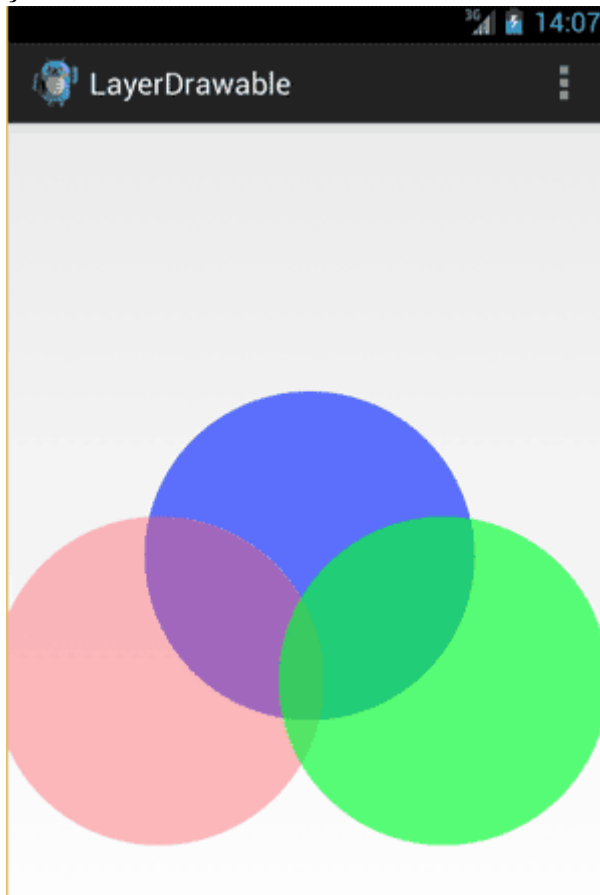
private LayerDrawable createLayerDrawable() {
    BitmapDrawable drawable0 = (BitmapDrawable)
getResources().getDrawable(
    R.drawable.blue);
    drawable0.setGravity(Gravity.CENTER_VERTICAL);
    BitmapDrawable drawable1 = (BitmapDrawable)
getResources().getDrawable(
    R.drawable.red);
    drawable1.setGravity(Gravity.LEFT);
    BitmapDrawable drawable2 = (BitmapDrawable)
getResources().getDrawable(
    R.drawable.green);
    drawable2.setGravity(Gravity.RIGHT);
}

```

```

    Drawable drawableArray[] = new Drawable[] { drawable0, drawable1,
drawable2 };
    LayerDrawable layerDraw = new LayerDrawable(drawableArray);
    layerDraw.setLayerInset(1, -20, 300, 0, 0); // смещаем второй слой
    layerDraw.setLayerInset(2, 100, 300, 0, 0); // смещаем третий слой
    return layerDraw;
}

```



Также можно создать объекты LayerDrawable через XML с помощью тега layer-list.

StateListDrawable

У многих компонентов есть различные состояния: нажато, выбрано и т.д.. В Android есть возможность задавать изображения для любого из таких состояний при помощи класса StateListDrawable. Но удобнее это делать через XML-ресурсы.

LevelListDrawable

Используя LevelListDrawable, вы можете эффективно размещать ресурсы Drawable один поверх другого, указывая целочисленный индекс для каждого слоя.

Drawable. Фигуры и градиенты

-
- Shape и ShapeDrawable
- Градиенты: gradient и GradientDrawable

Shape и ShapeDrawable

Фигуры являются подмножеством Drawable-ресурсов.

Данный вид ресурсов на основе класса ShapeDrawable позволяет описывать простые геометрические фигуры, указывая их размеры, фон и контур с помощью тега <shape>.

Можно создавать ресурсы фигур на основе стандартных фигур вроде прямоугольника, эллипса, линии. Для использования ресурсов фигур нужно создать в подкаталоге res/drawable XML-файл, в котором будет присутствовать тег <shape>, который в свою очередь может содержать дочерние элементы <corners>, <gradient>, <padding>, <size>, <solid>, <stroke>.

Имя файла без расширения будет служить идентификатором (ID): R.drawable.filename в Java-коде и @[package:]drawable/filename в XML-файлах.

Элементы фигуры

<shape>

Фигура. Является корневым элементом в XML.

Атрибуты:

- xmlns:android - обязательный атрибут со строкой "http://schemas.android.com/apk/res/android"
- android:shape - задаёт тип фигуры: rectangle (прямоугольник, заполняющий элемент, является фигурой по умолчанию), oval (овал), line (линия, требуется также наличие элемента <stroke> для задания ширины линии), ring (окружность, для данной фигуры можно использовать атрибуты android:innerRadius, android:innerRadiusRatio, android:thickness, android:thicknessRatio, android:useLevel)
- <corners> - создаёт закругленные углы для фигуры. Только для прямоугольника. Возможные атрибуты: android:radius, android:topLeftRadius, android:topRightRadius, android:bottomLeftRadius, android:bottomRightRadius
- <gradient> - задаёт градиентную заливку для фигуры. Возможные атрибуты: android:angle, android:centerX, android:centerY, android:centerColor, android:endColor, android:gradientRadius, android:startColor, android:type, android:useLevel
- <padding> - отступы. Возможные атрибуты: android:left, android:top, android:right, android:bottom
- <size> - размеры фигуры. Возможные атрибуты: android:height, android:width
- <solid> - сплошной цвет для фигуры. Возможные атрибуты: android:color
- <stroke> - контур фигуры. Возможные атрибуты: android:width, android:color, android:dashGap (расстояние между черточками), android:dashWidth (длина пунктирной черточки)

rectangle (Прямоугольник)



shape_rect.xml - Атрибут android:shape здесь необязателен: rectangle — это значение по умолчанию.

```

<selector
  xmlns:android="http://schemas.android.com/apk/res/android">
  <item>
    <shape android:shape="rectangle">
      <solid android:color="#0377BE"/>
    </shape>
  </item>
</selector>

```

Пример с градиентным прямоугольником в качестве разделителя
Создадим файл separator.xml:

```

<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
  android:shape="rectangle" >

  <gradient
    android:angle="0"
    android:centerColor="#47CF4F"
    android:endColor="#000000"
    android:startColor="#000000" />

</shape>

```

В разметке приложения добавим код:

```

<TextView
  android:id="@+id/tvSource"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="TextView" />

<View
  android:layout_width="wrap_content"
  android:background="@drawable/separator"
  android:layout_height="1dp" />

<TextView
  android:id="@+id/tvResult"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="TextView" />

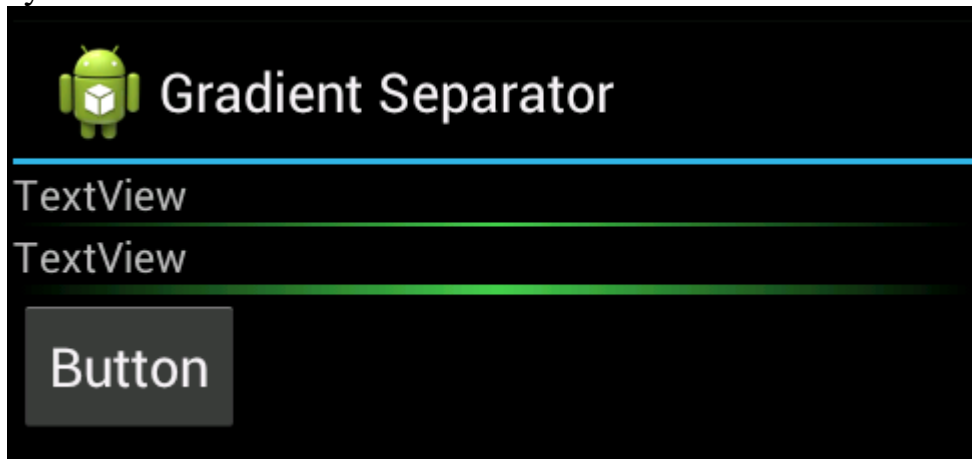
<View
  android:layout_width="wrap_content"
  android:background="@drawable/separator"

```

```
android:layout_height="3dp" />
```

```
<Button  
    android:id="@+id/button1"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Button" />
```

У первого разделителя ширина 1dp, у второго - 3dp. Получили красивую полосу.



У прямоугольников можно скруглить углы при помощи тега corners



rectangle_rounded_all.xml

```
<?xml version="1.0" encoding="utf-8"?>  
<selector  
    xmlns:android="http://schemas.android.com/apk/res/android">  
    <item>  
        <shape>  
            <solid android:color="#B902B0"/>  
            <corners android:radius="10.0dip" />  
        </shape>  
    </item>  
</selector>
```

Можно закруглить углы по отдельности:



rectangle_rounded_some.xml

```
<?xml version="1.0" encoding="utf-8"?>  
<selector  
    xmlns:android="http://schemas.android.com/apk/res/android">  
    <item>
```

```

    <shape>
        <solid android:color="#EC6118"/>
        <corners android:bottomRightRadius="0.1dp"
            android:bottomLeftRadius="7dip"
            android:topLeftRadius="7dip"
            android:topRightRadius="0.1dp"/>
    </shape>
</item>
</selector>
oval (Эллипс)
shape_oval.xml

```



```

<?xml version="1.0" encoding="utf-8"?>
<selector
    xmlns:android="http://schemas.android.com/apk/res/android">
    <item>
        <shape android:shape="oval">
            <solid android:color="#FCD366"/>
        </shape>
    </item>
</selector>

```

Другой вариант с пунктиром:

```

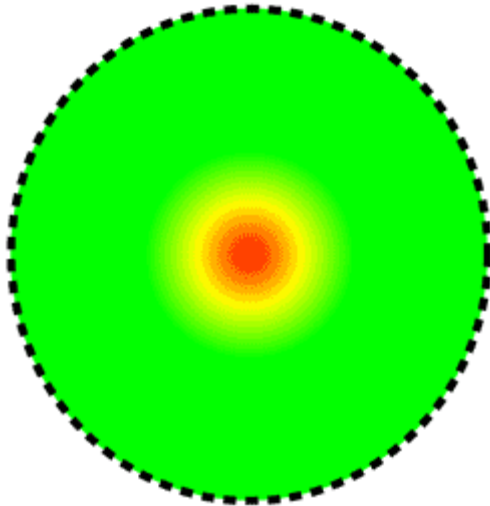
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="oval" >

    <gradient
        android:centerColor="#FFFF00"
        android:centerX="0.5"
        android:centerY="0.5"
        android:endColor="#00FF00"
        android:gradientRadius="100"
        android:startColor="#FF0000"
        android:type="radial" />

    <stroke
        android:dashGap="6dip"
        android:dashWidth="8dip"
        android:width="5dip"
        android:color="#000000" />

```

</shape>



ring (Кольцо)



shape_ring.xml - Для кольца имеются дополнительные атрибуты:

innerRadius

Внутренний радиус

innerRadiusRatio

Отношение между внешним и внутренним радиусами. По умолчанию равно 3

thickness

Толщина кольца (т.е. разница между внешним и внутренним радиусами)

thicknessRatio

Отношение ширины кольца к его толщине. По умолчанию равно 9

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<selector
```

```
  xmlns:android="http://schemas.android.com/apk/res/android">
```

```
    <item>
```

```
      <shape android:shape="ring" android:innerRadiusRatio="3"
```

```
        android:thicknessRatio="5.333">
```

```
        <solid android:color="#7DBE15"/>
```

```
      </shape>
```

```
    </item>
```

```
</selector>
```

line (Горизонтальная линия)



shape_line.xml - Линия может быть только горизонтальной

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
  <item>
    <shape android:shape="line">
      <stroke android:width="1dip" android:color="#F20107" />
    </shape>
  </item>
</selector>
```

Градиенты: gradient и GradientDrawable

Тег gradient (класс GradientDrawable) позволяет создавать сложные градиентные заливки. Каждый градиент описывает плавный переход между двумя или тремя цветами с помощью линейного/радиального алгоритма или же используя метод развертки.

Тег gradient внутри тега shape. Основные атрибуты: type, startColor (обязателен), endColor (обязателен) и middleColor (необязателен). Также иногда оказывается полезным атрибут centerColor.

Используя атрибут type, вы можете описать свой градиент:

linear



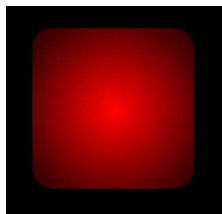
gradient_linear.xml

- android:type="linear" можно опустить, он так и есть по умолчанию. Отображает прямой переход от цвета startColor к цвету endColor под углом, заданным в атрибуте angle.
- Атрибут android:angle используется только линейным градиентом и должен быть кратным значению 45.

```
<?xml version="1.0" encoding="utf-8"?>
<selector
  xmlns:android="http://schemas.android.com/apk/res/android">
  <item>
    <shape>
      <gradient android:type="linear" android:endColor="#3C0000"
        android:startColor="#FF0202" android:angle="45.0" />
      <corners android:radius="10.0dip" />
    </shape>
  </item>
</selector>
```

Дополнительный материал: [Android Dev Tip #3](#) - помните о прозрачности, который может привести к другому результату.

radial



- `android:gradientRadius` является обязательным для радиального градиента, а у остальных игнорируется. Рисует круговой градиент, начиная с цвета `startColor` и заканчивая `endColor`, от внешнего края фигуры до ее центра. Требуется атрибут `gradientRadius`, который указывает радиус градиентного перехода в пикселях. Поддерживаются также необязательные атрибуты `centerX` и `centerY`, описывающие сдвиг центральной точки градиента. Поскольку радиус градиента указывается в пикселях, он не будет автоматически масштабироваться при разной плотности точек на экране. Чтобы минимизировать эффект ступенчатости, необходимо указывать разные значения радиуса для дисплеев с разным разрешением.

- Имеются также атрибуты `android:centerX` и `android:centerY`, в которых можно задавать относительное (0.0 – 1.0) расположение центра градиента.

`gradient_radial.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<selector
  xmlns:android="http://schemas.android.com/apk/res/android">
  <item>
    <shape>
      <gradient android:type="radial" android:endColor="#3C0000"
        android:startColor="#FF0202" android:gradientRadius="50"/>
      <corners android:radius="10.0dip" />
    </shape>
  </item>
</selector>
sweep
```

Рисует развёрточный градиент с помощью перехода между цветами `startColor` и `endColor` вдоль внешнего края фигуры (как правило, кольца).



Можно использовать атрибуты `android:centerX` и `android:centerY`.
`gradient_sweep.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<selector
```

```

xmlns:android="http://schemas.android.com/apk/res/android">
    <item>
        <shape>
            <gradient    android:type="sweep"    android:endColor="#3C0000"
android:startColor="#FF0202" />
            <corners android:radius="10.0dip" />
        </shape>
    </item>
</selector>
gradient_circle.xml

```

```

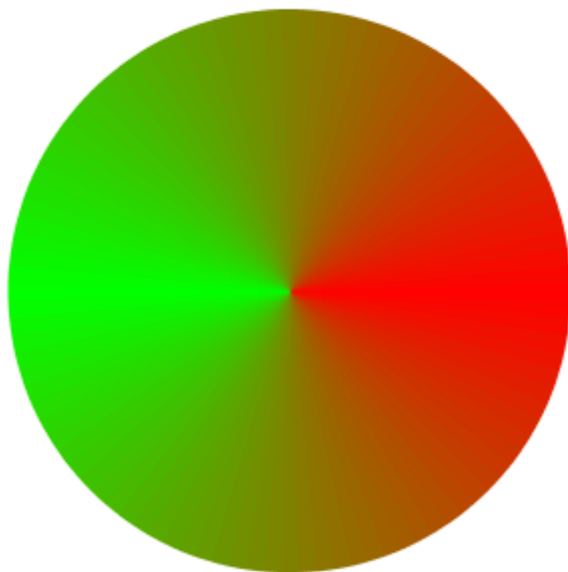
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="oval" >

    <gradient
        android:centerColor="#0f0"
        android:endColor="#f00"
        android:startColor="#f00"
        android:type="sweep" />

    <size
        android:height="400dp"
        android:width="400dp" />

</shape>

```



Примеры с shape

Закругляем уголки у компонентов

Создадим отдельный файл res/drawable/roundrect.xml и с его помощью скруглим уголки у LinearLayout, ImageView, TextView, EditText:

```

<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle" >

    <corners android:radius="15dp" />

    <gradient
        android:angle="90"
        android:endColor="#FFFFFF"
        android:startColor="#FF0000"
        android:type="linear" />

    <stroke
        android:width="1dp"
        android:color="#FF0000" />

    <padding
        android:bottom="15dp"
        android:left="15dp"
        android:right="15dp"
        android:top="15dp" />

</shape>

```

В разметке активности пишем следующее:

```

<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_margin="10dp"
    android:background="@drawable/roundrect"
    android:orientation="vertical"
    tools:context=".MainActivity" >

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="@drawable/roundrect"
        android:src="@drawable/ic_launcher"
        android:contentDescription="Round Image View"
        tools:ignore="HardcodedText" />

    <TextView

```

```
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_margin="2dp"
android:background="@drawable/roundrect"
android:text="Я - TextView"
tools:ignore="HardcodedText" />
```

```
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="2dp"
    android:background="@drawable/roundrect"
    android:hint="EditText with roundrect"
    android:text="Я - EditText"
    tools:ignore="HardcodedText" />
```

</LinearLayout>

Результат:



Овальный кабинет

В Белом доме есть Овальный кабинет. Если вам придётся писать приложение для администрации президента США, то все элементы нужно сделать овальными. Создадим файл `res/drawable/oval.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
```

```
android:shape="oval" >
```

```
<gradient  
    android:angle="90"  
    android:endColor="#FFFFFFF"  
    android:startColor="#FF000000"  
    android:type="linear" />
```

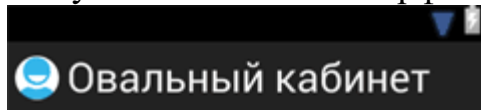
```
<stroke  
    android:width="1dp"  
    android:color="#FF000000" />
```

```
<padding  
    android:bottom="15dp"  
    android:left="15dp"  
    android:right="15dp"  
    android:top="15dp" />
```

```
</shape>
```

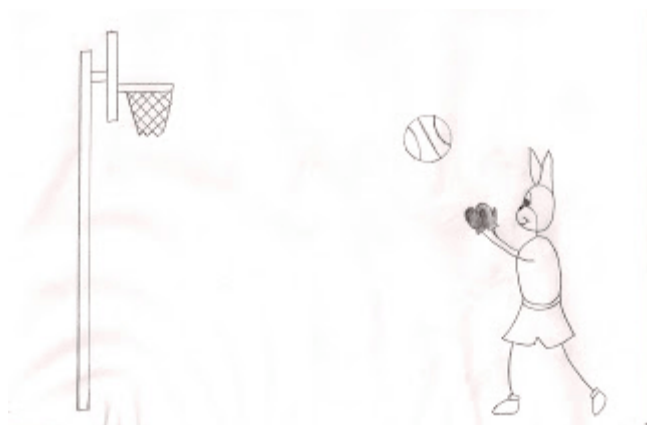
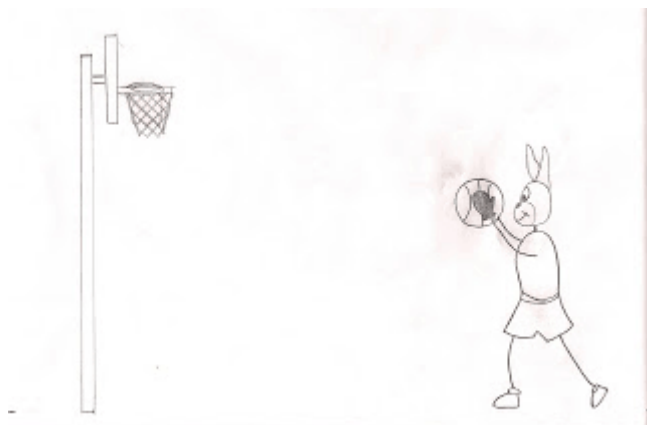
Заменим в предыдущем примере `android:background="@drawable/roundrect"` на `android:background="@drawable/oval"`.

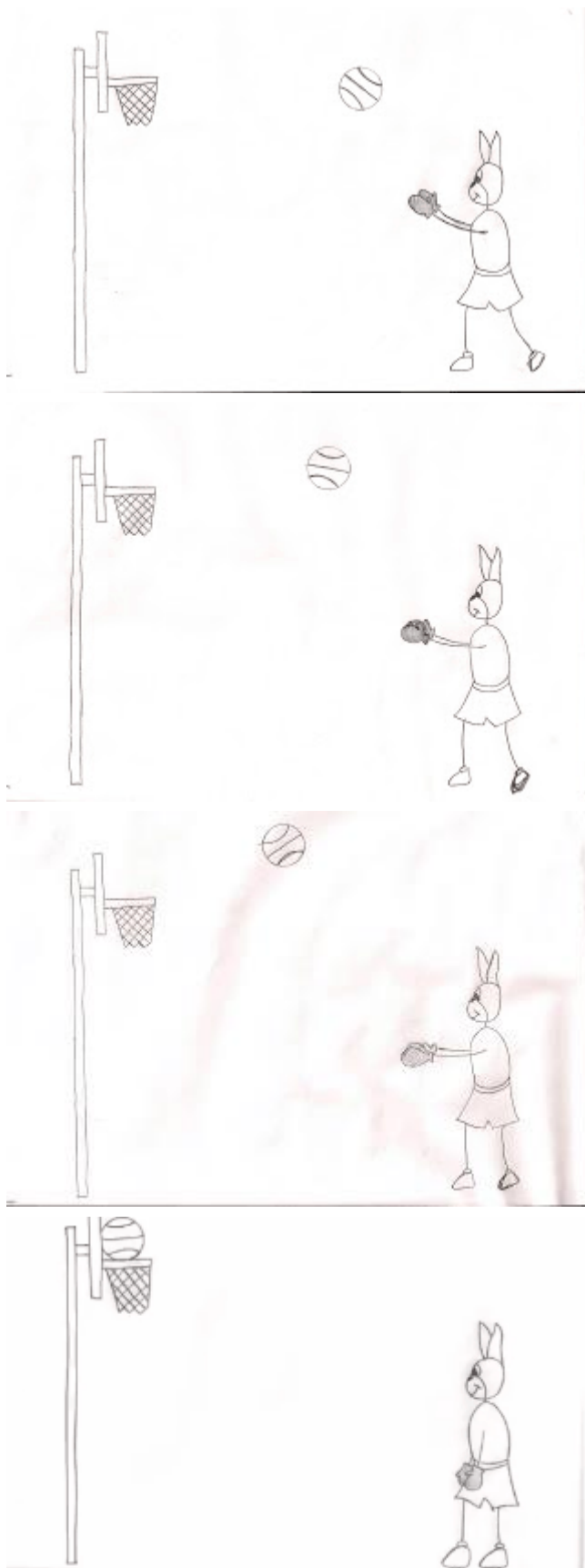
Получим овальный интерфейс:



ЛЕКЦИЯ №58-№59. РАБОТА С АНИМАЦИЕЙ. TWEEN ANIMATION И FRAME ANIMATION.

Cell animation представляет собой технику анимации, при которой ряд изображений или кадров последовательно сменяют друг друга за короткий промежуток времени. Подобная техника довольно распространена при создании мультфильмов. Например, имеется следующий набор изображений:





При достаточно быстрой смене кадров получится динамический эффект зайца, забрасывающего мяч в баскетбольную корзину. Теперь рассмотрим, как сделать подобную анимацию в приложении Android.

Во-первых, нам надо добавить все эти изображения в проект в папку res/drawable. И в эту же папку добавим новый xml-файл. Назовем его rabbit_animation.xml и поместим в него следующее содержимое:

```
<?xml version="1.0" encoding="utf-8"?>

<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
    android:oneshot="false" >

    <item android:drawable="@drawable/a1" android:duration="250" />
    <item android:drawable="@drawable/a2" android:duration="250" />
    <item android:drawable="@drawable/a3" android:duration="250" />
    <item android:drawable="@drawable/a4" android:duration="250" />
    <item android:drawable="@drawable/a5" android:duration="250" />
    <item android:drawable="@drawable/a6" android:duration="250" />
    <item android:drawable="@drawable/a7" android:duration="250" />

</animation-list>
```

Анимация определяется с помощью корневого элемента animation-list, который содержит набор ключевых кадров в виде элементов item.

Свойство android:oneshot="false" в определении корневого элемента указывает, что анимация будет продолжаться циклически. А при значении true анимация срабатывала только один раз.

Каждый элемент анимации устанавливает ссылку на ресурс изображения с помощью свойства android:drawable, а также с помощью свойства android:duration устанавливает время в миллисекундах, которое будет отображаться изображение.

В разметке интерфейса для отображения анимации используется элемент ImageView:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <ImageView android:id="@+id/animationView"
```



```
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:src="@drawable/rabit_animation"
android:adjustViewBounds="true" android:padding="40dip"/>
```

```
</RelativeLayout>
```

Для установки ресурса анимации применяется свойство `android:src`, и также для растягивания по ширине контейнера с сохранением аспектного отношения между шириной и высотой используется свойство `android:adjustViewBounds="true"`

Больше для запуска анимации ничего не надо. И при запуске приложения анимация будет автоматически запускаться.

Также можно установить анимацию в коде java:

```
ImageView img = (ImageView)findViewById(R.id.animationView);
img.setBackgroundResource(R.drawable.rabit_animation);

// получаем объект анимации
AnimationDrawable frameAnimation = (AnimationDrawable)
img.getBackground();

frameAnimation.setOneShot(false);

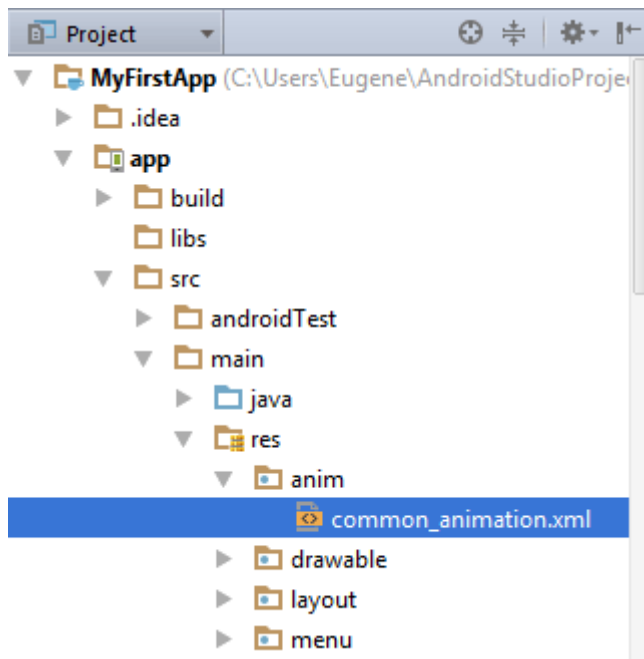
// запуск анимации
frameAnimation.start();
```

Tween-анимация

Tween-анимация представляет анимацию различных свойств объекта, при которой система сама рассчитывает некоторые промежуточные значения с помощью определенного алгоритма, который называется интерполяцией. В Android алгоритм интерполяции определяется встроенным классом `Animation`.

От данного класса наследуются классы, которые описывают конкретные типы анимаций, такие как `AlphaAnimation` (изменение прозрачности), `RotateAnimation` (анимация вращения), `ScaleAnimation` (анимация масштабирования), `TranslateAnimation` (анимация перемещения).

Мы можем определить анимацию как в коде java, так и в файле xml. Для хранения ресурсов анимации в папке res предназначена папка anim. По умолчанию данная папка отсутствует в проекте, поэтому создадим ее. Затем добавим в нее новый xml-файл, который назовем common_animation.xml:



Определим в этом файле следующее содержание:

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator="@android:anim/linear_interpolator">
    <scale android:fromXScale="1.0" android:toXScale="0.5"
        android:fromYScale="1.0" android:toYScale="0.5"
        android:pivotX="50%" android:pivotY="50%" android:duration="4500"
        android:repeatCount="infinite" android:repeatMode="reverse" />
    <rotate
        android:fromDegrees="0.0"
        android:toDegrees="60.0"
        android:pivotX="50%"
        android:pivotY="50%" />
    <alpha                android:fromAlpha="1.0"                android:toAlpha="0.1"
    android:duration="2250"
```

```
        android:repeatCount="infinite" android:repeatMode="reverse" />
<translate android:fromXDelta="0.0"
        android:toXDelta="50.0"
        android:fromYDelta="20.0"
        android:toYDelta="80.0"
        android:duration="2250"
        android:repeatMode="reverse"
        android:repeatCount="infinite" />
</set>
```

Здесь задействуются четыре типа анимаций: элемент scale представляет масштабирование, элемент rotate - вращение, элемент alpha - изменение прозрачности, элемент translate - перемещение. Если бы мы использовали одну анимацию, то могли бы определить один корневой элемент по типу анимации. Но так как мы используем набор, то все анимации помещаются в элемент set, который представляет класс AnimationSet

Все виды анимаций принимают ряд общих свойств. В частности, свойство android:repeatMode, которое указывает на режим выполнения. Если имеет значение reverse, то анимация выполняется также и в обратную сторону

Свойство android:repeatCount указывает на количество повторов анимации. Значение infinite устанавливает бесконечное число повторов.

Время анимации задается с помощью свойства android:duration

Для всех анимаций также характерно указание начальной и конечной точки трансформации.

Анимация масштабирования

Для анимация масштабирования задается начальное масштабирование по оси x (android:fromXScale) и по оси y (android:fromYScale) и конечные значения масштабирования android:toXScale и android:toYScale. Например, так как android:fromXScale=1.0, а android:toXScale=0.5, то по ширине будет происходить сжатие на 50%.

Также при масштабировании устанавливаются значения android:pivotX и android:pivotY, которые указывают на центр масштабирования или опорную точку.

Анимация вращения

Для анимации вращения задается начальное (android:fromDegrees) и конечное значения поворота (android:toDegrees).

С помощью свойств android:pivotX и android:pivotY также, как и при масштабировании, задается опорная точка вращения.

Анимация прозрачности

Для анимации прозрачности задается начальное значение прозрачности (android:fromAlpha) и финальное значение, устанавливаемое при завершении анимации (android:toAlpha). Все значения варьируются в диапазоне от 1.0 (непрозрачный) до 0.0 (полностью прозрачный)

Анимация перемещения

Для перемещения также устанавливаются начальные (android:fromXDelta и android:fromYDelta) и конечные значения (android:toXDelta и android:toYDelta)

Для всех анимаций начальные и конечные значения указывают некий диапазон, в котором будут ранжироваться значения. Само вычисление значений на этом промежутке зависит от конкретного алгоритма. В данном случае в качестве алгоритма устанавливается линейная интерполяция. Для этого у корневого элемента set определено свойство android:interpolator="@android:anim/linear_interpolator".

Кроме данного значения свойство android:interpolator может принимать еще ряд других: bounce_interpolator, cycle_interpolator и т.д.

Данная анимация будет применяться к элементу ImageView, который определим в файле разметки интерфейса:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <ImageView android:id="@+id/animationView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:adjustViewBounds="true" android:padding="40dip"/>
```

```
</RelativeLayout>
```

Теперь определим и запустим анимацию в классе activity:

```
package com.example.eugene.myfirstapp;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.ImageView;
import android.view.animation.AnimationUtils;
import android.view.animation.Animation;

public class MainActivity extends Activity{

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        ImageView img = (ImageView)findViewById(R.id.animationView);
        // определим для ImageView какое-нибудь изображение
        img.setImageResource(R.drawable.image2);
        // создаем анимацию
        Animation animation = AnimationUtils.loadAnimation(this,
R.anim.common_animation);
        // запуск анимации
        img.startAnimation(animation);
    }
}
```

Сначала определяем анимацию по тому файлу common_animation.xml, который содержит набор анимаций:

```
Animation animation = AnimationUtils.loadAnimation(this, R.anim.common_animation);
```

А потом запускаем ее:

```
img.startAnimation(animation);
```



ЛЕКЦИЯ №60-№61. СЛУЖБЫ В ANDROID. КОМПОНЕНТ SERVICE.

Service является компонентом приложения, который может выполнять длительные операции в фоновом режиме и не содержит пользовательского интерфейса. Другой компонент приложения может запустить службу, которая продолжит работу в фоновом режиме даже в том случае, когда пользователь перейдет в другое приложение. Кроме того, компонент может привязаться к службе для взаимодействия с ней и даже выполнять межпроцессное взаимодействие (IPC). Например, служба может обрабатывать сетевые транзакции, воспроизводить музыку, выполнять ввод-вывод файла или взаимодействовать с поставщиком контента, и все это в фоновом режиме.

Фактически служба может принимать две формы:

Запущенная

Служба является «запущенной», когда компонент приложения (например, операция) запускает ее вызовом `startService()`. После запуска служба может работать в фоновом режиме в течение неограниченного времени, даже если уничтожен компонент, который ее запустил. Обычно запущенная служба выполняет одну операцию и не возвращает результатов вызывающему компоненту. Например, она может загружать или выгружать файл по сети. Когда операция выполнена, служба должна остановиться самостоятельно.

Привязанная

Служба является «привязанной», когда компонент приложения привязывается к ней вызовом `bindService()`. Привязанная служба предлагает интерфейс клиент-сервер, который позволяет компонентам взаимодействовать со службой, отправлять запросы, получать результаты и даже делать это между разными процессами посредством межпроцессного взаимодействия (IPC). Привязанная служба работает только пока к ней привязан другой компонент приложения. К службе могут быть привязаны несколько компонентов одновременно, но когда все они отменяют привязку, служба уничтожается.

Хотя в этой документации эти два типа служб обсуждаются отдельно, служба может работать обоими способами — она может быть запущенной (и работать в течение неограниченного времени) и допускать привязку. Это зависит от реализации пары методов обратного вызова: `onStartCommand()` позволяет компонентам запускать службу, а `onBind()` позволяет выполнять привязку.

Независимо от состояния приложения (запущенное, привязанное или и оба сразу) любой компонент приложения может использовать службу (даже из отдельного приложения) подобно тому, как любой компонент может использовать операцию — запустив ее с помощью `Intent`. Однако вы можете объявить закрытую службу в файле манифеста и заблокировать доступ к ней из других приложений.

Внимание! Служба работает в основном потоке ведущего процесса — служба не создает своего потока и не выполняется в отдельном процессе (если вы не указали иное). Это означает, что если ваша служба собирается выполнять любую работу с высокой нагрузкой ЦП или блокирующие операции (например, воспроизведение MP3 или сетевые операции), вы должны создать в службе новый поток для выполнения этой работы. Используя отдельный поток, вы снижаете риск возникновения ошибок «Приложение не отвечает», и основной поток приложения может обрабатывать взаимодействие пользователя с вашими операциями.

Основы

Что лучше использовать — службу или поток?

Служба — это просто компонент, который может выполняться в фоновом режиме, даже когда пользователь не взаимодействует с приложением. Следовательно, вы должны создавать службу только в том случае, если вам нужно именно это.

Если вам требуется выполнить работу за пределами основного потока, но только в то время, когда пользователь взаимодействует с приложением, то вам, вероятно, следует создать новый поток, а не службу. Например, если вы хотите воспроизводить определенную музыку, но только во время работы операции, вы можете создать поток в onCreate(), запустить его выполнение в методе onStart(), а затем остановить его в методе onStop(). Также рассмотрите возможность использования класса AsyncTask или HandlerThread вместо обычного класса Thread. В документе Процессы и потоки содержится дополнительная информация об этих потоках.

Помните, что если вы действительно используете службу, она выполняется в основном потоке вашего приложения по умолчанию, поэтому вы должны создать новый поток в службе, если она выполняет интенсивные или блокирующие операции.

Чтобы создать службу, необходимо создать подкласс класса Service (или одного из существующих его подклассов). В вашей реализации необходимо переопределить некоторые методы обратного вызова, которые обрабатывают ключевые моменты жизненного цикла службы и при необходимости предоставляют механизм привязывания компонентов. Наиболее важные методы обратного вызова, которые необходимо переопределить:

onStartCommand()

Система вызывает этот метод, когда другой компонент, например, операция, запрашивает запуск этой службы, вызывая startService(). После выполнения этого метода служба запускается и может в течение неограниченного времени работать в фоновом режиме. Если вы реализуете такой метод, вы обязаны остановить службу посредством вызова stopSelf() или stopService(). (Если требуется только обеспечить привязку, реализовывать этот метод не обязательно).

onBind()

Система вызывает этот метод, когда другой компонент хочет выполнить привязку к службе (например, для выполнения удаленного вызова процедуры) путем вызова bindService(). В вашей реализации этого метода вы должны обеспечить интерфейс, который клиенты используют для взаимодействия со службой, возвращая IBinder. Всегда необходимо реализовывать этот метод, но если вы не хотите разрешать привязку, необходимо возвращать значение `null`.

onCreate()

Система вызывает этот метод при первом создании службы для выполнения однократных процедур настройки (перед вызовом onStartCommand() или onBind()). Если служба уже запущена, этот метод не вызывается.

onDestroy()

Система вызывает этот метод, когда служба более не используется и выполняется ее уничтожение. Ваша служба должна реализовать это для очистки ресурсов, таких как потоки, зарегистрированные приемники, ресиверы и т. д. Это последний вызов, который получает служба.

Если компонент запускает службу посредством вызова `startService()` (что приводит к вызову `onStartCommand()`), то служба продолжает работу, пока она не остановится самостоятельно с помощью `stopSelf()` или другой компонент не остановит ее посредством вызова `stopService()`.

Если компонент вызывает `bindService()` для создания службы (и `onStartCommand()` не вызывается), то служба работает, пока к ней привязан компонент. Как только выполняется отмена привязки службы ко всем клиентам, система уничтожает службу.

Система Android будет принудительно останавливать службу только в том случае, когда не хватает памяти, и необходимо восстановить системные для операции, которая отображается на переднем плане. Если служба привязана к операции, которая отображается на переднем плане, менее вероятно, что она будет уничтожена, и если служба объявлена для выполнения на переднем плане (как обсуждалось выше), она почти никогда не будет уничтожаться. В противном случае, если служба была запущена и является длительной, система со временем будет опускать ее положение в списке фоновых задач, и служба станет очень чувствительной к уничтожению — если ваша служба запущена, вы должны предусмотреть изящную обработку ее перезапуска системой. Если система уничтожает вашу службу, она перезапускает ее, как только снова появляется доступ к ресурсам (хотя это также зависит от значения, возвращаемого методом `onStartCommand()`, как обсуждается ниже). Дополнительная информация о ситуациях, в которых система может уничтожить службу приведена в документе Процессы и потоки.

В следующих разделах описаны способы создания служб каждого типа и использования их из других компонентов приложения.

Объявление службы в манифесте

Все службы, как и операции (и другие компоненты), должны быть объявлены в файле манифеста вашего приложения.

Чтобы объявить службу, добавьте элемент `<service>`, в качестве дочернего элемента `<application>`. Например:

```
<manifest ... >
...
<application ... >
  <service android:name=".ExampleService" />
  ...
</application>
</manifest>
```

Дополнительные сведения об объявлении службы в манифесте см. в справке по элементу `<service>`.

Имеются другие атрибуты, которые можно включить в элемент `<service>` для задания свойств, например, необходимых для запуска разрешений, и процесса, в котором должна выполняться служба.

Атрибут `android:name` является единственным обязательным атрибутом — он указывает имя класса для службы. После публикации вашего приложения вам не следует менять это имя, поскольку это может разрушить код из-за зависимости от явных намерений, используемых, чтобы запустить или привязать службу (ознакомьтесь с публикацией Вещи, которые нельзя менять в блоге разработчиков).

Для обеспечения безопасности приложения всегда используйте явное намерение при запуске или привязке `Service` и не объявляйте фильтров намерений для службы. Если вам важно допустить некоторую неопределенность в отношении того, какая служба запускается, вы можете предоставить фильтры намерений для ваших служб и исключить имя компонента из `Intent`, но затем вы должны установить пакет для намерения с помощью `setPackage()`, который обеспечивает достаточное устранение неоднозначности для целевой службы.

Дополнительно можно обеспечить доступность вашей службы только для вашего приложения, включив атрибут `android:exported` и установив для него значение "false". Это не позволяет другим приложениям запускать вашу службу даже при использовании явного намерения.

Создание запущенной службы

Запущенная служба — это служба, которую запускает другой компонент вызовом `startService()`, что приводит к вызову метода `onStartCommand()` службы.

При запуске служба обладает сроком жизни, не зависящим от запустившего ее компонента, и может работать в фоновом режиме в течение неограниченного времени, даже если уничтожен компонент, который ее запустил. Поэтому после выполнения своей работы служба должна остановиться самостоятельно посредством вызова метода `stopSelf()`, либо ее может остановить другой компонент посредством вызова метода `stopService()`.

Компонент приложения, например, операция, может запустить службу, вызвав метод `startService()` и передав объект `Intent`, который указывает службу и любые данные, которые служба должна использовать. Служба получает этот объект `Intent` в методе `onStartCommand()`.

Предположим, что операции требуется сохранить некоторые данные в сетевой базе данных. Операция может запустить службу и предоставить ей данные для сохранения, передав намерение в метод `startService()`. Служба получает намерение в методе `onStartCommand()`, подключается к Интернету и выполняет транзакцию с базой данных. Когда транзакция выполнена, служба останавливается самостоятельно и уничтожается.

Внимание! По умолчанию службы работают в том же процессе, что и приложение, в котором они объявлены, а также в основном потоке этого приложения. Поэтому, если ваша служба выполняет интенсивные или блокирующие операции, в то время как пользователь взаимодействует с операцией из того же приложения, служба будет замедлять выполнение операции. Чтобы избежать негативного воздействия на скорость работы приложения, вы должны запустить новый поток внутри службы.

Традиционно имеется два класса, которые вы можете наследовать для создания запущенной службы:

Service

Это базовый класс для всех служб. Когда вы наследуете этот класс, важно создать новый поток, в котором будет выполняться вся работа службы, поскольку по умолчанию служба использует основной поток вашего приложения, что может замедлить любую операцию, которую выполняет ваше приложение.

IntentService

Это подкласс класса Service, который использует рабочий поток для обработки всех запросов запуска поочередно. Это оптимальный вариант, если вам не требуется, чтобы ваша служба обрабатывала несколько запросов одновременно. Достаточно реализовать метод onHandleIntent(), который получает намерение для каждого запроса запуска, позволяя выполнять фоновую работу.

В следующих разделах описано, как реализовать службу с помощью любого из этих классов.

Наследование класса IntentService

Так как большинству запущенных приложений не требуется обрабатывать несколько запросов одновременно, (что может быть действительно опасным сценарием), вероятно будет лучше, если вы реализуете свою службу с помощью класса IntentService.

Класс IntentService делает следующее:

Создает рабочий поток по умолчанию, который выполняет все намерения, доставленные в метод onStartCommand(), отдельно от основного потока вашего приложения.

Создает рабочую очередь, которая передает намерения по одному в вашу реализацию метода onHandleIntent(), поэтому вы не должны беспокоиться относительно многопоточности.

Останавливает службу после обработки всех запросов запуска, поэтому вам никогда не требуется вызывать stopSelf().

Предоставляет реализацию метода onBind() по умолчанию, которая возвращает null.

Предоставляет реализацию метода onStartCommand() по умолчанию, которая отправляет намерение в рабочую очередь и затем в вашу реализацию onHandleIntent().

Все это означает, что вам достаточно реализовать метод onHandleIntent() для выполнения работы, предоставленной клиентом. (Хотя, кроме того, вы должны предоставить маленький конструктор для службы).

Здесь приведен пример реализации класса IntentService:

```
public class HelloIntentService extends IntentService {
```

```
/**
```

```
* A constructor is required, and must call the super IntentService(String)
```

```

* constructor with a name for the worker thread.
*/
public HelloIntentService() {
    super("HelloIntentService");
}

/**
 * The IntentService calls this method from the default worker thread with
 * the intent that started the service. When this method returns, IntentService
 * stops the service, as appropriate.
 */
@Override
protected void onHandleIntent(Intent intent) {
    // Normally we would do some work here, like download a file.
    // For our sample, we just sleep for 5 seconds.
    long endTime = System.currentTimeMillis() + 5*1000;
    while (System.currentTimeMillis() < endTime) {
        synchronized (this) {
            try {
                wait(endTime - System.currentTimeMillis());
            } catch (Exception e) {
            }
        }
    }
}
}
}

```

Это все, что нужно: конструктор и реализация класса onHandleIntent().

Если вы решили переопределить также и другие методы обратного вызова, такие как onCreate(), onStartCommand() или onDestroy(), обязательно вызовите реализацию суперкласса, чтобы класс IntentService мог правильно обрабатывать жизненный цикл рабочего потока.

Например, метод onStartCommand() должен возвращать реализацию по умолчанию (которая доставляет намерение в onHandleIntent()):

```

@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    Toast.makeText(this, "service starting", Toast.LENGTH_SHORT).show();
    return super.onStartCommand(intent, flags, startId);
}

```

Помимо onHandleIntent(), единственный метод, из которого вам не требуется вызывать суперкласс, это метод onBind() (но его нужно реализовывать только в случае, если ваша служба допускает привязку).

В следующем разделе вы увидите, как реализовывается служба такого же типа при наследовании базового класса Service, которая содержит намного

больше кода, но которая может подойти, если вам требуется обрабатывать одновременные запросы запуска.

Наследование класса Service

Как вы видели в предыдущем разделе, использование класса IntentService значительно упрощает реализацию запущенной службы. Однако, если необходимо, чтобы ваша служба поддерживала многопоточность (вместо обработки запросов запуска через рабочую очередь), можно наследовать класс Service для обработки каждого намерения.

В качестве примера приведена следующая реализация класса Service, которая выполняет ту же работу, как и пример выше, использующий класс IntentService. То есть для каждого запроса запуска он использует рабочий поток для выполнения задания и обрабатывает запросы по одному.

```
public class HelloService extends Service {
    private Looper mServiceLooper;
    private ServiceHandler mServiceHandler;

    // Handler that receives messages from the thread
    private final class ServiceHandler extends Handler {
        public ServiceHandler(Looper looper) {
            super(looper);
        }
        @Override
        public void handleMessage(Message msg) {
            // Normally we would do some work here, like download a file.
            // For our sample, we just sleep for 5 seconds.
            long endTime = System.currentTimeMillis() + 5*1000;
            while (System.currentTimeMillis() < endTime) {
                synchronized (this) {
                    try {
                        wait(endTime - System.currentTimeMillis());
                    } catch (Exception e) {
                    }
                }
            }
            // Stop the service using the startId, so that we don't stop
            // the service in the middle of handling another job
            stopSelf(msg.arg1);
        }
    }

    @Override
    public void onCreate() {
        // Start up the thread running the service. Note that we create a
        // separate thread because the service normally runs in the process's
        // main thread, which we don't want to block. We also make it
```

```

// background priority so CPU-intensive work will not disrupt our UI.
HandlerThread thread = new HandlerThread("ServiceStartArguments",
    Process.THREAD_PRIORITY_BACKGROUND);
thread.start();

// Get the HandlerThread's Looper and use it for our Handler
mServiceLooper = thread.getLooper();
mServiceHandler = new ServiceHandler(mServiceLooper);
}

@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    Toast.makeText(this, "service starting", Toast.LENGTH_SHORT).show();

    // For each start request, send a message to start a job and deliver the
    // start ID so we know which request we're stopping when we finish the job
    Message msg = mServiceHandler.obtainMessage();
    msg.arg1 = startId;
    mServiceHandler.sendMessage(msg);

    // If we get killed, after returning from here, restart
    return START_STICKY;
}

@Override
public IBinder onBind(Intent intent) {
    // We don't provide binding, so return null
    return null;
}

@Override
public void onDestroy() {
    Toast.makeText(this, "service done", Toast.LENGTH_SHORT).show();
}
}

```

Как можно видеть, этот код значительно длиннее, чем код с использованием класса IntentService.

Однако, так как вы обрабатываете каждый вызов onStartCommand() самостоятельно, вы можете выполнять несколько запросов одновременно. Данный код выполняет не совсем эту работу, но при необходимости вы можете создавать новые потоки для каждого запроса и сразу запускать их (а не ожидать завершения предыдущего запроса).

Обратите внимание, что метод onStartCommand() должен возвращать целое число. Это целое число описывает, как система должна продолжать

выполнение службы в случае, когда система уничтожила ее (как описано выше, реализация по умолчанию для класса IntentService обрабатывает эту ситуацию, хотя вы изменить ход реализации). Значение, возвращаемое методом onStartCommand(), должно быть одной из следующих констант:

START_NOT_STICKY

Если система уничтожает службу после возвращения из onStartCommand(), не нужно повторно создавать службу, если нет ожидающих доставки намерений. Это самый безопасный вариант, позволяющий избежать запуска вашей службы, когда это не нужно и когда ваше приложение может просто перезапустить любые незавершенные задания.

START_STICKY

Если система уничтожает службу после возвращения из onStartCommand(), повторно создайте службу и вызовите onStartCommand(), но не передавайте последнее намерение повторно. Вместо этого система вызывает метод onStartCommand() с намерением, которое имеет значение null, если нет ожидающих намерений для запуска службы. Если ожидающие намерения есть, они доставляются. Это подходит для мультимедийных проигрывателей (или подобных служб), которые не выполняют команды, а работают независимо и ожидают задание.

START_REDELIVER_INTENT

Если система уничтожает службу после возвращения из onStartCommand(), повторно создайте службу и вызовите onStartCommand() с последним намерением, которое было доставлено в службу. Все ожидающие намерения доставляются по очереди. Это подходит для служб, активно выполняющих задание, которое должно быть возобновлено немедленно, например, для загрузок файла.

Для получения дополнительных сведений об этих возвращаемых значениях см. справочную документацию по ссылке для каждой константы.

Запуск службы

Можно запустить службу из операции или другого компонента приложения, передав объект Intent (указывающий службу, которую требуется запустить) в startService(). Система Android вызывает метод onStartCommand() службы и передает ей Intent. (Ни в коем случае не следует вызывать метод onStartCommand() напрямую).

Например, операция может запустить службу из примера в предыдущем разделе (HelloService), используя явное намерение с помощью startService():

```
Intent intent = new Intent(this, HelloService.class);
startService(intent);
```

Метод startService() возвращается немедленно, и система Android вызывает метод службы onStartCommand(). Если служба еще не выполняется, система сначала вызывает onCreate(), а затем onStartCommand().

Если служба также не представляет привязку, намерение, доставляемое с помощью startService(), является единственным режимом связи между

компонентом приложения и службой. Однако, если вы хотите, чтобы служба управляла результатом обратно, клиент, который запускает службу, может создать объект PendingIntent для сообщения (с помощью getBroadcast()) и доставить его в службу в объекте Intent, который запускает службу. Затем служба может использовать сообщение для доставки результата.

Несколько запросов запуска службы приводят к нескольким соответствующим вызовам метода onStartCommand() службы. Однако для ее остановки достаточно только одного запроса на остановку службы (с помощью stopSelf() или stopService()).

Остановка службы

Запущенная служба должна управлять своим жизненным циклом. То есть, система не останавливает и не уничтожает службу, если не требуется восстановить память системы, и служба продолжает работу после возвращения из метода onStartCommand(). Поэтому служба должна останавливаться самостоятельно посредством вызова метода stopSelf(), либо другой компонент может остановить ее посредством вызова метода stopService().

Получив запрос на остановку посредством stopSelf() или stopService(), система как можно скорее уничтожает службу.

Однако, если служба обрабатывает несколько запросов onStartCommand() одновременно, вы не должны останавливать службу после завершения обработки запроса запуска, поскольку вы, вероятно, уже получили новый запрос запуска (остановка в конце первого запроса привела бы к прерыванию второго). Чтобы избежать этой проблемы, вы можете использовать метод stopSelf(int), гарантирующий, что ваш запрос на остановку службы всегда основан на самом последнем запросе запуска. То есть, когда вы вызываете stopSelf(int), вы передаете идентификатор запроса запуска (идентификатор startId, доставленный в onStartCommand()), которому соответствует ваш запрос остановки. Тогда, если служба получит новый запрос запуска до того, как вы сможете вызвать stopSelf(int), идентификатор не будет совпадать и служба не будет остановлена.

Внимание! Ваше приложение обязательно должно останавливать свои службы по окончании работы, чтобы избежать расходования ресурсов системы и потребления энергии аккумулятора. При необходимости другие компоненты могут остановить службу посредством вызова метода stopService(). Даже если вы можете выполнять привязку службы, следует всегда останавливать службу самостоятельно, если она когда-либо получила вызов onStartCommand().

Дополнительные сведения о жизненном цикле службы представлены в разделе Управление жизненным циклом службы ниже.

Создание привязанной службы

Привязанная служба — это служба, которая допускает привязку к ней компонентов приложения посредством вызова bindService() для создания долговременного соединения (и обычно не позволяет компонентам запускать ее посредством вызова startService()).

Вы должны создать привязанную службу, когда вы хотите взаимодействовать со службой из операций и других компонентов вашего приложения или показывать некоторые функции вашего приложения другим приложениям посредством межпроцессного взаимодействия (IPC).

Чтобы создать привязанную службу, необходимо реализовать метод обратного вызова onBind() для возвращения объекта IBinder, который определяет интерфейс взаимодействия со службой. После этого другие компоненты приложения могут вызвать метод bindService() для извлечения интерфейса и начать вызывать методы службы. Служба существует только для обслуживания привязанного к ней компонента приложения, поэтому, когда нет компонентов, привязанных к службе, система уничтожает ее (вам не требуется останавливать привязанную службу, как это требуется для службы, запущенной посредством onStartCommand()).

Чтобы создать привязанную службу, необходимо в первую очередь определить интерфейс, взаимодействия клиента со службой. Этот интерфейс между службой и клиентом должен быть реализацией объекта IBinder, которую ваша служба должна возвращать из метода обратного вызова onBind(). После того, как клиент получает объект IBinder, он может начать взаимодействие со службой посредством этого интерфейса.

Одновременно к службе могут быть привязаны несколько клиентов. Когда клиент заканчивает взаимодействие со службой, он вызывает unbindService() для отмены привязки. Как только не остается ни одного клиента, привязанного к службе, система уничтожает службу.

Существует несколько способов реализации привязанной службы, и эти реализации сложнее, чем реализации запущенной службы, поэтому обсуждение привязанной службы приведено в отдельном документе Привязанные службы.

Отправка уведомлений пользователю

После запуска служба может уведомлять пользователя о событиях, используя Всплывающие уведомления или Уведомления в строке состояния.

Всплывающее уведомление — это сообщение, кратковременно появляющееся на поверхности текущего окна, тогда как уведомление в строке состояния — это значок в строке состояния с сообщением, который пользователь может выбрать, чтобы выполнить действие (такое как запуск операции).

Обычно уведомление в строке состояния является самым удобным решением, когда завершается какая-то фоновая работа (например, завершена загрузка файла), и пользователь может действовать. Когда пользователь выбирает уведомление в расширенном виде, уведомление может запустить операцию (например, для просмотра загруженного файла).

Дополнительную информацию см. в руководствах для разработчиков Всплывающие уведомления и Уведомления в строке состояния.

Запуск службы на переднем плане

Служба переднего плана — это служба, о которой пользователь активно осведомлен, и поэтому она не является кандидатом для удаления системой в случае нехватки памяти. Служба переднего плана должна выводить уведомление в строку состояния, которая находится под заголовком «Постоянные». Это означает, что уведомление не может быть удалено, пока служба не будет остановлена или удалена с переднего плана.

Например, музыкальный проигрыватель, который воспроизводит музыку из службы, должен быть настроен на работу на переднем плане, так как пользователь точно знает о его работе. Уведомление в строке состояния может показывать текущее произведение и позволять пользователю запускать операцию для взаимодействия с музыкальным проигрывателем.

Для запроса на выполнение вашей службы на переднем плане вызовите метод `startForeground()`. Этот метод имеет два параметра: целое число, которое однозначно идентифицирует уведомление и объект `Notification` для строки состояния. Например:

```
Notification notification = new Notification(R.drawable.icon,
getText(R.string.ticker_text),
System.currentTimeMillis());
Intent notificationIntent = new Intent(this, ExampleActivity.class);
PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, notificationIntent,
0);
notification.setLatestEventInfo(this, getText(R.string.notification_title),
getText(R.string.notification_message), pendingIntent);
startForeground(ONGOING_NOTIFICATION_ID, notification);
```

Внимание! Целочисленный идентификатор ID, который вы передаете в метод `startForeground()`, не должен быть равен 0.

Чтобы удалить службу с переднего плана, вызовите `stopForeground()`. Этот метод содержит логическое значение, указывающее, следует ли также удалять уведомление в строке состояния. Этот метод не останавливает службу. Однако, если вы останавливаете службу, работающую на переднем плане, уведомление также удаляется.

Дополнительную информацию об уведомлениях см. в разделе Создание уведомлений в строке состояния.

Управление жизненным циклом службы

Жизненный цикл службы намного проще, чем жизненный цикл операции. Однако, намного важнее уделить пристальное внимание тому, как ваша служба создается и уничтожается, так как служба может работать в фоновом режиме без ведома пользователя.

Жизненный цикл службы от создания до уничтожения может следовать двум разным путям:

Запущенная служба

Служба создается, когда другой компонент вызывает метод `startService()`. Затем служба работает в течение неограниченного времени и должна остановиться самостоятельно посредством вызова

метода stopSelf(). Другой компонент также может остановить службу посредством вызова метода stopService(). Когда служба останавливается, система уничтожает ее.

Привязанная служба

Служба создается, когда другой компонент (клиент) вызывает метод bindService(). Затем клиент взаимодействует со службой через интерфейс IBinder. Клиент может закрыть соединение посредством вызова метода unbindService(). К одной службе могут быть привязано несколько клиентов, и когда все они отменяют привязку, система уничтожает службу. (Служба не должна останавливаться самостоятельно.)

Эти два способа необязательно работают независимо друг от друга. То есть вы можете привязать службу, которая уже была запущена посредством метода startService(). Например, фоновая музыкальная служба может быть запущена посредством вызова метода startService() с объектом Intent, который идентифицирует музыку для воспроизведения. Позже, например, когда пользователь хочет получить доступ к управлению проигрывателем или информацию о текущем произведении, операция может установить привязку к службе посредством вызова метода bindService(). В подобных случаях методы stopService() и stopSelf() фактически не останавливают службу, пока не будет отменена привязка всех клиентов.

Реализация обратных вызовов жизненного цикла

Подобно операции, служба содержит методы обратного вызова жизненного цикла, которые можно реализовать для контроля изменений состояния службы и выполнения работы в соответствующие моменты времени. Указанная ниже базовая служба показывает каждый из методов жизненного цикла.

```
public class ExampleService extends Service {
    int mStartMode;    // indicates how to behave if the service is killed
    IBinder mBinder;   // interface for clients that bind
    boolean mAllowRebind; // indicates whether onRebind should be used

    @Override
    public void onCreate() {
        // The service is being created
    }
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        // The service is starting, due to a call to startService()
        return mStartMode;
    }
    @Override
    public IBinder onBind(Intent intent) {
        // A client is binding to the service with bindService()
        return mBinder;
    }
}
```

```

@Override
public boolean onUnbind(Intent intent) {
    // All clients have unbound with unbindService()
    return mAllowRebind;
}
@Override
public void onRebind(Intent intent) {
    // A client is binding to the service with bindService(),
    // after onUnbind() has already been called
}
@Override
public void onDestroy() {
    // The service is no longer used and is being destroyed
}
}

```

Примечание. В отличие от методов обратного вызова жизненного цикла операции, вам не требуется вызывать реализацию суперкласса этих методов обратного вызова.

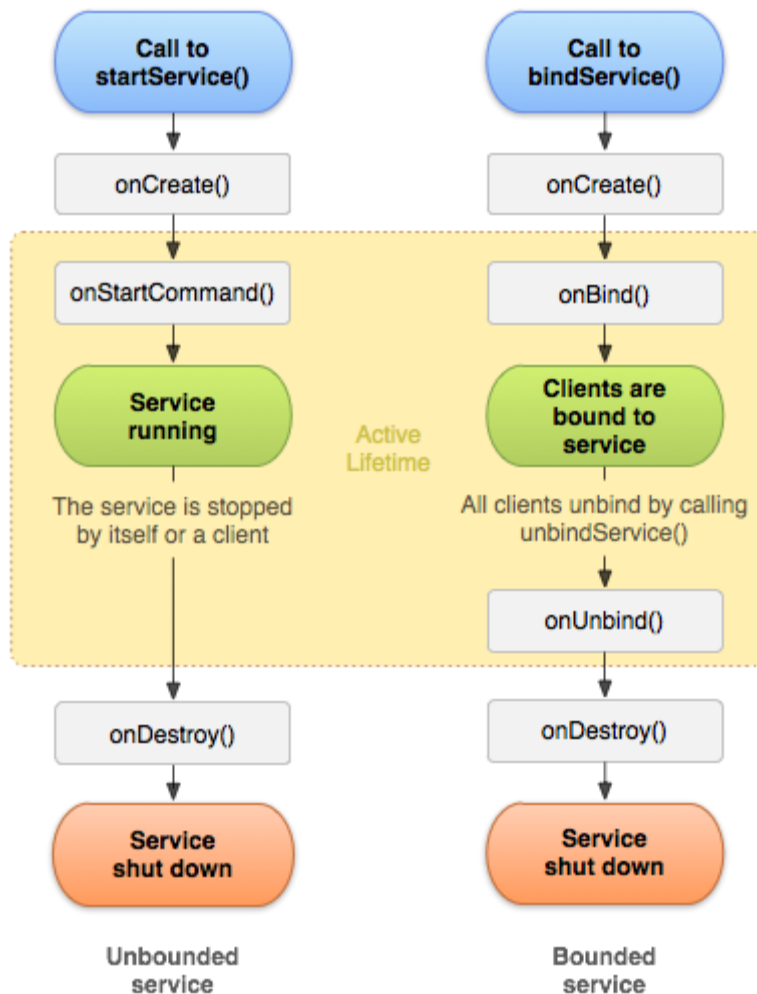


Рисунок 2. Жизненный цикл службы. На схеме слева показан жизненный цикл, когда служба создана посредством метода `startService()`, а на

схеме справа показан жизненный цикл, когда служба создана посредством метода bindService().

С помощью реализации этих методов можно отслеживать два вложенных цикла в жизненном цикле службы:

Весь жизненный цикл службы происходит между вызовом метода onCreate() и возвратом из метода onDestroy(). Подобно операции, служба выполняет начальную настройку в методе onCreate() и освобождает все оставшиеся ресурсы в методе onDestroy(). Например, служба воспроизведения музыки может создать поток для воспроизведения музыки в методе onCreate(), затем остановить поток в методе onDestroy().

Методы onCreate() и onDestroy() вызываются для всех служб, независимо от метода создания: startService() или bindService().

Активный жизненный цикл службы начинается с вызова метода onStartCommand() или onBind(). Каждый метод направляется намерением Intent, которое было передано методу startService() или bindService(), соответственно.

Если служба запущена, активный жизненный цикл заканчивается одновременно с окончанием всего жизненного цикла (служба активна даже после возврата из метода onStartCommand()). Если служба является привязанной, активный жизненный цикл заканчивается, когда возвращается метод onUnbind().

Примечание. Хотя запущенная служба останавливается посредством вызова метода stopSelf() или stopService(), для службы не существует соответствующего обратного вызова (нет обратного вызова onStop()). Поэтому, если служба не привязана к клиенту, система уничтожает ее при остановке службы — метод onDestroy() является единственным получаемым методом обратного вызова.

Рисунок 2 иллюстрирует типичные методы обратного вызова для службы. Хотя на рисунке отделены службы, созданные посредством метода startService(), от служб, созданных посредством метода bindService(), помните, что любая служба, независимо от способа запуска, позволяет клиентам выполнять привязку к ней. Поэтому служба, изначально созданная посредством метода onStartCommand() (клиентом, который вызвал startService()), может получать вызов метода onBind() (когда клиент вызывает метод bindService()).

Дополнительные сведения о создании службы, которая обеспечивает привязку, см. в документе Привязанные службы, который содержит дополнительную информацию о методе обратного вызова onRebind() в разделе Управление жизненным циклом привязанной службы.

ЛЕКЦИЯ №62-№65. ДАТЧИКИ МОБИЛЬНЫХ УСТРОЙСТВ.
УПРАВЛЕНИЕ ДАТЧИКАМИ В ПРИЛОЖЕНИИ.

Платформа Android идеально подходит для создания инновационных приложений с использованием аппаратных датчиков, особенно Java-программистами. Мы рассмотрим некоторые варианты интерфейсов для приложений Android, включая использование подсистемы датчиков и записи аудиофрагментов.

Какое приложение лучше построить, чтобы задействовать аппаратные возможности устройства на платформе Android? Подойдет любое, в котором требуются электронные глаза и уши. На ум приходит электронная няня, система безопасности или даже сейсмограф. Хотя с метафизической точки зрения одновременно присутствовать в двух местах невозможно, Android может предложить некоторые практические способы преодоления этого ограничения. В рамках этой статьи под Android-устройством понимается не просто "мобильный телефон", а установленное в определенном месте устройство с беспроводным соединением с сетью, таким как EDGE или Wi-Fi. Загрузите исходный код примера приложения для этой статьи.

Сенсорные возможности Android

Один из приятных аспектов работы с платформой Android заключается в возможности получить доступ к некоторым полезным компонентам самого устройства. До сих пор разработчиков мобильных устройств разочаровывала невозможность доступа к их внутреннему оборудованию. Хотя между вами и металлом все же остается прослойка Java-среды Android, команда разработчиков Android вывела многие возможности аппаратуры на поверхность. А так как это платформа с открытым исходным кодом, можно засучить рукава и написать собственный код для решения своих задач.

Загрузите SDK Android, если вы этого еще не сделали. Рекомендуем также изучить содержимое пакета `android.hardware` и следить за примерами этой статьи. Пакет `android.media` содержит классы, которые предоставляют разработчикам новые полезные функции.

Ниже описаны некоторые аппаратно-ориентированные функции, содержащиеся в SDK Android.

Таблица 1. Аппаратно-ориентированные функции SDK Android

Функция
<code>android.hardware.Camera</code>
<code>android.hardware.SensorManager</code>
<code>android.hardware.SensorListener</code>
<code>android.media.MediaRecorder</code>

android.FaceDetector

android.os.*

java.util.Date
java.util.Timer
java.util.TimerTask

Android.hardware.SensorManager содержит несколько констант, которые характеризуют различные аспекты системы датчиков Android, в том числе:

Тип датчика

Ориентация, акселерометр, свет, магнитное поле, близость, температура и т.д.

Частота измерений

Максимальная, для игр, обычная, для пользовательского интерфейса. Когда приложение запрашивает конкретное значение частоты отсчетов, с точки зрения сенсорной подсистемы это лишь рекомендация. Никакой гарантии, что измерения будут производиться с указанной частотой, нет.

Точность

Высокая, низкая, средняя, ненадежные данные.

Центром сенсорных приложений служит интерфейс `SensorListener`. Он включает в себя два необходимых метода:

Метод `onSensorChanged(int sensor, float values[])` вызывается всякий раз, когда изменяется значение датчика. Этот метод вызывается только для датчиков, контролируемых данным приложением (подробнее об этом ниже). В число аргументов метода входит целое, которое указывает, что значение датчика изменилось, и массив значений с плавающей запятой, отражающих собственно значение датчика. Некоторые датчики выдают только одно значение данных, тогда как другие предоставляют три значения с плавающей запятой. Датчики ориентации и акселерометр дают по три значения данных каждый.

Метод `onAccuracyChanged(int sensor, int accuracy)` вызывается при изменении точности показаний датчика. Аргументами служат два целых числа: одно указывает датчик, а другое соответствует новому значению точности этого датчика.

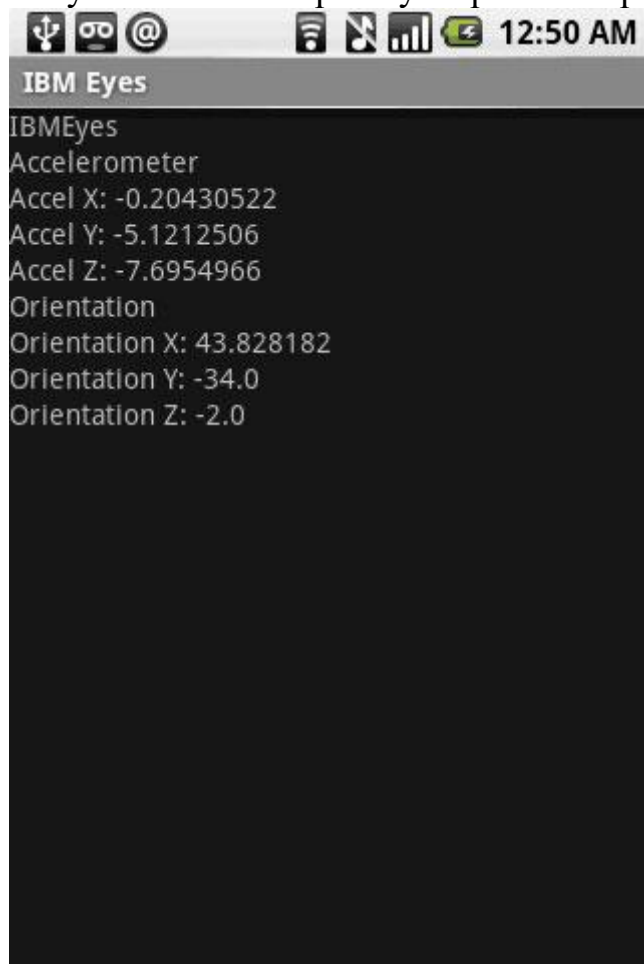
Для взаимодействия с датчиком приложение должно зарегистрироваться на прием действий, связанных с одним или несколькими датчиками. Регистрация осуществляется с помощью метода `registerListener` класса `SensorManager`. Пример кода для этой статьи демонстрирует, как приложение регистрируется и отменяет регистрацию с помощью `SensorListener`.

Помните, что не каждое устройство Android поддерживает тот или иной датчик, указанный в SDK. Если на конкретном устройстве тот или иной датчик отсутствует, ваше приложение должно обрабатывать эту ситуацию аккуратно.

Пример работы с датчиком

Этот пример приложения просто следит за изменениями значений датчиков ориентации и акселерометра (исходный код приведен в разделе [Загрузки](#)). При изменении значений датчиков они отображаются на экране в виджете TextView. Рисунок 1 демонстрирует приложение в действии.

Рисунок 1. Мониторинг ускорения и ориентации



Приложение создано в среде Eclipse с плагином Android Developer Tools. (Более подробную информацию о разработке Android-приложений с помощью Eclipse см. в разделе [Ресурсы](#).) В листинге 1 приведен код этого приложения.

Листинг 1. IBMEyes.java

```
package com.msi.ibm.eyes;
import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
import android.widget.TextView;
import android.hardware.SensorManager;
import android.hardware.SensorListener;
public class IBMEyes extends Activity implements SensorListener {
    final String tag = "IBMEyes";
```



```

SensorManager sm = null;
TextView xViewA = null;
TextView yViewA = null;
TextView zViewA = null;
TextView xViewO = null;
TextView yViewO = null;
TextView zViewO = null;

/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // get reference to SensorManager
    sm = (SensorManager) getSystemService(SENSOR_SERVICE);
    setContentView(R.layout.main);
    xViewA = (TextView) findViewById(R.id.xbox);
    yViewA = (TextView) findViewById(R.id.ybox);
    zViewA = (TextView) findViewById(R.id.zbox);
    xViewO = (TextView) findViewById(R.id.xboxo);
    yViewO = (TextView) findViewById(R.id.yboxo);
    zViewO = (TextView) findViewById(R.id.zboxo);
}

public void onSensorChanged(int sensor, float[] values) {
    synchronized (this) {
        Log.d(tag, "onSensorChanged: " + sensor + ", x: " +
values[0] + ", y: " + values[1] + ", z: " + values[2]);
        if (sensor == SensorManager.SENSOR_ORIENTATION) {
            xViewO.setText("Orientation X: " + values[0]);
            yViewO.setText("Orientation Y: " + values[1]);
            zViewO.setText("Orientation Z: " + values[2]);
        }
        if (sensor == SensorManager.SENSOR_ACCELEROMETER) {
            xViewA.setText("Accel X: " + values[0]);
            yViewA.setText("Accel Y: " + values[1]);
            zViewA.setText("Accel Z: " + values[2]);
        }
    }
}

public void onAccuracyChanged(int sensor, int accuracy) {
    Log.d(tag,"onAccuracyChanged: " + sensor + ", accuracy: " + accuracy);
}

@Override
protected void onResume() {
    super.onResume();

```

```

        // register this class as a listener for the orientation and accelerometer
sensors
        sm.registerListener(this,
            SensorManager.SENSOR_ORIENTATION
|SensorManager.SENSOR_ACCELEROMETER,
            SensorManager.SENSOR_DELAY_NORMAL);
    }

    @Override
    protected void onStop() {
        // unregister listener
        sm.unregisterListener(this);
        super.onStop();
    }
}

```

Это обычное приложение на основе действий, поскольку оно просто выводит на экран данные, получаемые от датчиков. Для ситуаций, когда устройство должно одновременно выполнять другие высокоприоритетные действия, приложение лучше было бы реализовать в форме сервиса.

Метод действия onCreate получает ссылку на SensorManager, где расположены все связанные с датчиками функции. Кроме того, метод onCreate создает ссылки на шесть виджетов TextView, в которые будут выводиться результаты измерений.

Метод onResume(), используя ссылку на SensorManager, регистрируется на прием обновлений данных датчика посредством метода registerListener:

Первый параметр представляет собой экземпляр класса, реализующий интерфейс SensorListener.

Второй параметр - это битовая маска желаемых датчиков. В данном случае приложение запрашивает данные из SENSOR_ORIENTATION и SENSOR_ACCELEROMETER.

Третий параметр указывает системе, как быстро должны обновляться значения датчиков для данного приложения.

Когда работа приложения приостанавливается, нужно отменить регистрацию приемника, чтобы больше не получать обновления значений датчика. Это делается с помощью метода unregisterListener класса SensorManager. Единственным параметром является экземпляр SensorListener.

При вызове обоих методов registerListener и unregisterListener приложение использует ключевое слово this. Обратите внимание на ключевое слово implements в определении класса, которое декларирует, что этот класс реализует интерфейс SensorListener. Вот почему в registerListener и unregisterListener передается this.

SensorListener должен реализовать два метода: onSensorChange и onAccuracyChanged. Наш пример приложения не

заботится о точности датчиков, а лишь вводит текущие значения X, Y и Z этих датчиков. Метод `onAccuracyChanged`, по существу, ничего не делает; он просто добавляет запись в журнал при каждом вызове.

Кажется, что метод `onSensorChanged` вызывается постоянно, так как акселерометр и датчик ориентации передают данные быстро. Чтобы определить, какой датчик передает данные, мы смотрим на первый параметр. Когда передающий датчик выявлен, соответствующие элементы пользовательского интерфейса обновляются данными, содержащимися в массиве значений с плавающей запятой, принятом в качестве второго аргумента метода. Хотя пример просто отображает эти значения, в более сложных приложениях они могут анализироваться, сравниваться с предыдущими значениями или использоваться в некоем алгоритме распознавания образов для определения того, что делает пользователь (или что происходит во внешней среде).

Теперь, когда мы рассмотрели подсистему датчиков, в следующем разделе приведем пример кода, который записывает звук на телефон Android. Этот пример работает на устройстве для разработчиков Dev1.

Применение MediaRecorder

Пакет `Android.media` содержит классы для взаимодействия с мультимедийной подсистемой. Класс `android.media.MediaRecorder` используется для записи медиафрагментов, включая аудио и видео. `MediaRecorder` действует как конечный автомат. Вы задаете различные параметры, такие как устройство-источник и формат. После установки запись может выполняться как угодно долго, пока не будет остановлена.

В листинге 2 приведен код для записи звука на устройство Android. Этот код не включает элементы пользовательского интерфейса приложения (полный исходный код см. в разделе загрузок).

Листинг 2. Запись аудиофрагмента

```
MediaRecorder mrec ;
File audiofile = null;
private static final String TAG="SoundRecordingDemo";
protected void startRecording() throws IOException
{
    mrec.setAudioSource(MediaRecorder.AudioSource.MIC);
    mrec.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
    mrec.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);
    if (mSampleFile == null)
    {
        File sampleDir = Environment.getExternalStorageDirectory();
        try
        {
            audiofile = File.createTempFile("ibm", ".3gp", sampleDir);
        }
        catch (IOException e)
```

```

        {
            Log.e(TAG,"sdcard access error");
            return;
        }
    }
    mrec.setOutputFile(audiofile.getAbsolutePath());
    mrec.prepare();
    mrec.start();
}
protected void stopRecording()
{
    mrec.stop();
    mrec.release();
    processaudiofile(audiofile.getAbsolutePath());
}
protected void processaudiofile()
{
    ContentValues values = new ContentValues(3);
    long current = System.currentTimeMillis();
    values.put(MediaStore.Audio.Media.TITLE, "audio" + audiofile.getName());
    values.put(MediaStore.Audio.Media.DATE_ADDED, (int) (current / 1000));
    values.put(MediaStore.Audio.Media.MIME_TYPE, "audio/3gpp");
    values.put(MediaStore.Audio.Media.DATA, audiofile.getAbsolutePath());
    ContentResolver contentResolver = getContentResolver();

    Uri base = MediaStore.Audio.Media.EXTERNAL_CONTENT_URI;
    Uri newUri = contentResolver.insert(base, values);

    sendBroadcast(new
Intent(Intent.ACTION_MEDIA_SCANNER_SCAN_FILE, newUri));
}

```

В методе startRecording создается и инициализируется экземпляр MediaRecorder.

В качестве источника данных выбирается микрофон (MIC).

Выходной формат устанавливается в 3GPP (файлы *.3gp) - медиаформат, ориентированный на мобильные устройства.

Кодер настроен на AMR_NB - аудиоформат с частотой дискретизации 8 кГц. NB означает узкую полосу частот. Различные форматы данных и имеющиеся кодеры рассматриваются в документации SDK.

Аудиофайл хранится не во внутренней памяти, а на отдельной карте. External.getExternalStorageDirectory() возвращает имена карты памяти и временного файла, созданного в этом каталоге. Затем этот файл связывается с экземпляром MediaRecorder, обращаясь к методу setOutputFile. Аудиоданные будут храниться в этом файле.

Вызов метода `prepare` завершает инициализацию `MediaRecorder`. Когда нужно начать процесс записи, вызывается метод `start`. Запись в файл на карте памяти ведется до тех пор, пока не будет вызван метод `stop`. Этот метод освобождает ресурсы, выделенные экземпляру `MediaRecorder`.

Когда аудиофрагмент записан, можно выполнить несколько действий:

Добавить аудиозапись в медиатеку на устройстве.

Выполнить некоторые шаги по распознаванию звука:

Не плач ли это ребенка?

Это голос хозяина, и нужно разблокировать телефон?

Или это фраза "Сезам, откройся", которая отпирает потайную дверь?

Автоматически загрузить звуковой файл в сетевую папку для обработки.

В примере кода метод `processaudiofile` добавляет аудиозапись в медиатеку. Для уведомления встроенного приложения о том, что доступна новая информация, используется `Intent`.

И последнее замечание об этом фрагменте кода: сразу после создания он не будет записывать аудио. Вы увидите созданный файл, но не услышите звука. Нужно добавить разрешение в файл `AndroidManifest.xml`:

```
<uses-permission
```

```
android:name="android.permission.RECORD_AUDIO"></uses-permission>
```

Теперь вы кое-что знаете о взаимодействии с датчиками Android и записи аудио. В следующем разделе приводится более широкий обзор архитектуры приложений, связанных со сбором данных и системами передачи сообщений.

Android как измерительная платформа

Платформа Android располагает широкими возможностями по измерению параметров окружающей среды. При таком многообразии входных параметров и датчиков в сочетании с мощными вычислительными и сетевыми функциями Android становится привлекательной платформой для создания систем взаимодействия с реальным миром. На рисунке 2 упрощенно показано соотношение между входами, логикой приложений и методами уведомления или выходами.

Рисунок 2. Блок-схема Android-ориентированной системы датчиков



Это очень гибкая архитектура; логику приложения можно распределять между локальным Android-устройством и серверными ресурсами, которые могут подключаться к более крупным базам данных и вычислительным мощностям. Например, аудиотрек, записанный на локальном Android-устройстве, можно отправить методом POST на Web-сервер, где данные сопоставляются с базой данных образцов голоса. Ясно, что это только самое

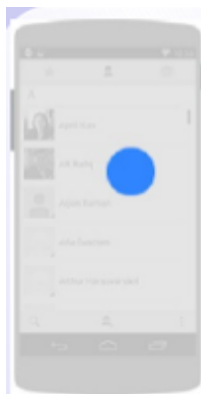
поверхностное знакомство с возможностями платформы. Надеюсь, что вы заинтересовались и начнете копать глубже, чтобы использовать платформу Android не только для целей мобильной телефонии.

Заключение

Из этой статьи вы получили первое представление о работе с датчиками Android. Приведенные примеры приложений определяют ориентацию и ускорение, а также взаимодействуют со средствами звукозаписи с использованием класса MediaRecorder. Android - гибкая, удобная платформа для создания систем взаимодействия с реальным миром. Сфера влияния Android очень быстро расширяется, и эта платформа осваивает все новые области применения. Не упускайте ее из вида.

Сенсорное (touch) управление

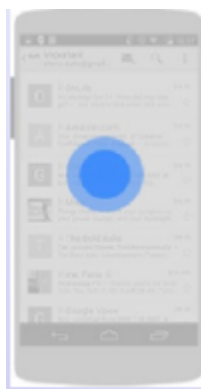
В этом разделе лекции рассмотрим возможности добавления сенсорного управления в мобильные приложения под Android. Сенсорное управление подразумевает использование сенсорных жестов для взаимодействия с приложением. Ниже представлен набор жестов, поддерживаемый системой Android.



Касание (touch).

Использование: Запуск действия по умолчанию для выбранного элемента.

Выполнение: нажать, отпустить.



Длинное касание (long touch).

Использование: Выбор элемента. Не стоит использовать этот жест для вызова контекстного меню.

Выполнение: нажать, ждать, отпустить.



Скольжение или перетаскивание (swipe or drag).

Использование: Прокрутка содержимого или навигация между элементами интерфейса одного уровня иерархии.

Выполнение: нажать, переместить, отпустить.



Скольжение после длинного касания (long press drag).

Использование: Перегруппировка данных или перемещение в контейнер.

Выполнение: длительное касание, переместить, отпустить.



Двойное касание (double touch).

Использование: Увеличение масштаба, выделение текста.

Выполнение: быстрая последовательность двух касаний.



Перетаскивание с двойным касанием (double touch drag).

Использование: Изменение размеров: расширение или сужение по отношению к центру жеста.

Выполнение: касание, следующее за двойным касанием со смещением вверх или вниз при этом:

- смещение вверх уменьшает размер содержимого;
- смещение вниз увеличивает размер содержимого.



Сведение пальцев (pinch close).

Использование: уменьшение содержимого, сворачивание.

Выполнение: касание экрана двумя пальцами, свести, отпустить.



Разведение пальцев (pinch open).

Использование: увеличение содержимого, разворачивание.

Выполнение: касание экрана двумя пальцами, развести, отпустить.

О возможности управлять приложением с помощью сенсорных жестов можно говорить в том случае, когда приложение способно распознать, что под набором касаний экрана скрывается некоторый жест и выполнить соответствующее действие. Процесс распознавания жеста обычно состоит из двух этапов: сбор данных и распознавание жеста. Рассмотрим эти этапы подробнее.

Сбор данных о сенсорных событиях

Основные действия, которые может произвести пользователь при взаимодействии с сенсорным экраном: коснуться экрана пальцем, переместить палец по экрану и отпустить. Эти действия распознаются системой Android, как сенсорные события (touch-события).

Каждый раз при появлении сенсорного события инициируется вызов метода `onTouchEvent()`. Обработка события станет возможной, если этот метод реализован в классе активности или некоторого компонента, иначе событие просто игнорируется.

Жест начинается, при первом касании экрана, продолжается пока система отслеживает положение пальцев пользователя и заканчивается получением финального события, состоящего в том, что ни один палец не касается экрана. Объект `MotionEvent`, передаваемый в метод `onTouchEvent()`, предоставляет детали каждого взаимодействия. Рассмотрим основные константы класса `MotionEvent`, определяющие сенсорные события:

- `MotionEvent.ACTION_DOWN` - касание экрана пальцем, является начальной точкой для любого сенсорного события или жеста;
- `MotionEvent.ACTION_MOVE` - перемещение пальца по экрану;
- `MotionEvent.ACTION_UP` - поднятие пальца от экрана.

Приложение может использовать предоставленные данные для распознавания жеста.

Можно реализовать свою собственную обработку событий для распознавания жеста, таким образом можно работать с произвольными жестами в приложении. Если же в приложении необходимо использовать стандартные жесты, описанные выше, можно воспользоваться классом `GestureDetector`. Этот класс позволяет распознать стандартные жесты без обработки отдельных сенсорных событий.

Распознавание жестов

Android предоставляет класс `GestureDetector` для распознавания стандартных жестов. Некоторые жесты, которые он поддерживает включают: `onDown()`, `onLongPress()`, `onFling()` и т.д. Можно использовать класс `GestureDetector` в связке с методом `onTouchEvent()`. Подробно распознавание поддерживаемых жестов рассмотрено в первой части лабораторной работы в этой теме.

Начиная с версии 1.6, Android предоставляет API для работы с жестами, который располагается в пакете `android.gesture` и позволяет сохранять, загружать, создавать и распознавать жесты. Виртуальное устройство Android (AVD), начиная все с той же версии 1.6, содержит предустановленное приложение, которое называется `Gesture Builder` и позволяет создавать жесты. После создания жесты сохраняются на SD карте виртуального устройства и могут быть добавлены в приложение в виде бинарного ресурса.

Для распознавания жестов необходимо добавить компонент `GestureOverlayView`, в XML файл активности. Этот компонент может быть добавлен как обычный элемент графического интерфейса пользователя и встроен в компоновку, например `RelativeLayout`. С другой стороны он может быть использован, как прозрачный слой поверх других компонентов, в этом случае в XML файле активности он должен быть записан, как корневой элемент.

Кроме всего вышеперечисленного, для использования собственных жестов в приложении необходимо реализовать интерфейс `OnGesturePerformedListener` и его метод `onGesturePerformed()`.

Другие сенсоры и датчики

Большинство устройств, работающих под управлением Android, укомплектованы встроенными сенсорами, которые предоставляют исходные данные высокой точности. Сенсоры могут быть полезны в том случае, если необходимо регистрировать положение и перемещения, повороты устройства в трехмерном пространстве, а также изменения параметров окружающей среды.

Платформа Android поддерживает три категории сенсоров:

Датчики движения

Эти сенсоры измеряют силы ускорения и вращательные силы по трем осям. Эта категория включает акселерометры, гироскопы, датчики вектора вращения и сенсоры силы тяжести.

Датчики окружающей среды

Эти сенсоры измеряют различные параметры окружающей среды, такие как температура воздуха и давление, освещенность и влажность. Эта категория включает барометры, термометры и датчики освещенности.

Датчики положения

Эти сенсоры измеряют физическое положение устройства. Эта категория включает магнитометры и датчики ориентации устройства в пространстве.

Сенсоры могут быть реализованы аппаратно или программно.

Аппаратно-реализованные датчики являются физическими элементами встроенными в мобильное устройство, они получают данные путем прямых измерений некоторых свойств, таких как ускорение, сила геомагнитного поля или изменение углов. Программно-реализованные датчики получают свои данные с одного или нескольких физических датчиков и вычисляют значение, которое от них ожидается.

Android предоставляет набор классов и интерфейсов для работы с сенсорами. Эти классы и интерфейсы являются частью пакета `android.hardware` и позволяют выполнять следующие задачи:

- определять какие сенсоры доступны на устройстве;
- определять индивидуальные возможности сенсоров, такие как максимальное значение, производитель, требования к потребляемой энергии и разрешения;
- собирать данные с сенсоров и определять минимальную частоту, с которой выполняется сбор данных;
- подключать и отключать слушателей событий от датчиков, события состоят в изменении значений датчиков. Для работы с датчиками Android предоставляет следующие классы и интерфейсы:

SensorManager -Этот класс может использоваться для создания экземпляра сервиса, связанного с сенсором. Также он предоставляет различные методы для доступа и составления списка сенсоров, подключения и отключения слушателей событий от сенсоров, сбора информации. Этот класс содержит константы, которые используются для задания точности сенсора, частоты получения данных и настройки датчиков.

Sensor -Этот класс используется для создания экземпляра датчика, предоставляет методы, позволяющие определить свойства сенсора.

SensorEvent -Система использует этот класс для создания объекта, соответствующего событию датчика и предоставляющего следующую

информацию: данные сенсора; тип сенсора, который породил событие, точность данных и время появления события.

SensorEventListener - Данный интерфейс может использоваться для реализации двух методов, получающих уведомления (события датчиков), когда меняется значение сенсора или когда меняется точность сенсора.

ЛЕКЦИЯ №66-№67. ПРОГРАММНЫЙ ДОСТУП К ДИСПЛЕЮ УСТРОЙСТВА.

При использовании смартфона на платформе Android нередко приходится передавать данные между двумя и более устройствами с одинаковой операционной системой, чтобы получить доступ к определенной информации, и в этом, как правило, нет ничего сложного. В то же время, если требуется передать с одного телефона на другой не просто файлы, а изображение в реальном времени, может возникнуть масса вопросов. По ходу данной статьи мы расскажем о нескольких более или менее универсальных решениях.

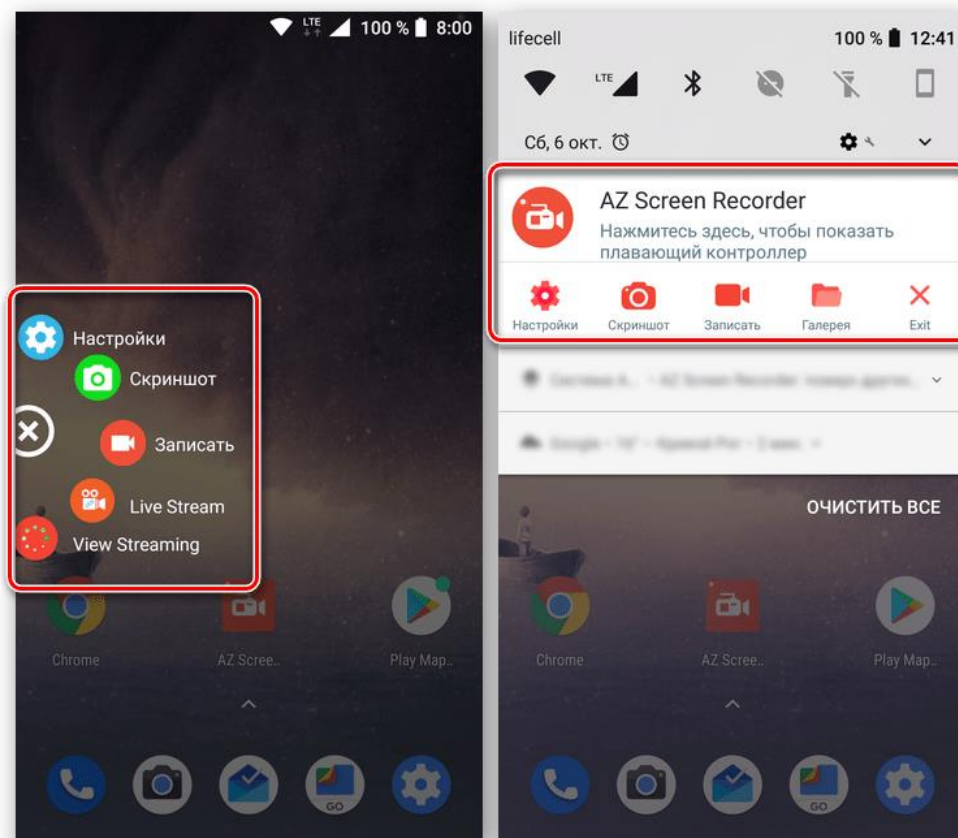
Вывод изображения с Android на Android

Для реализации поставленной задачи существует несколько основных методов, однако чаще всего так или иначе придется воспользоваться сторонними программами. Далее будут рассмотрены лишь наиболее известные примеры, но в Плей Маркете существуют и альтернативные решения.

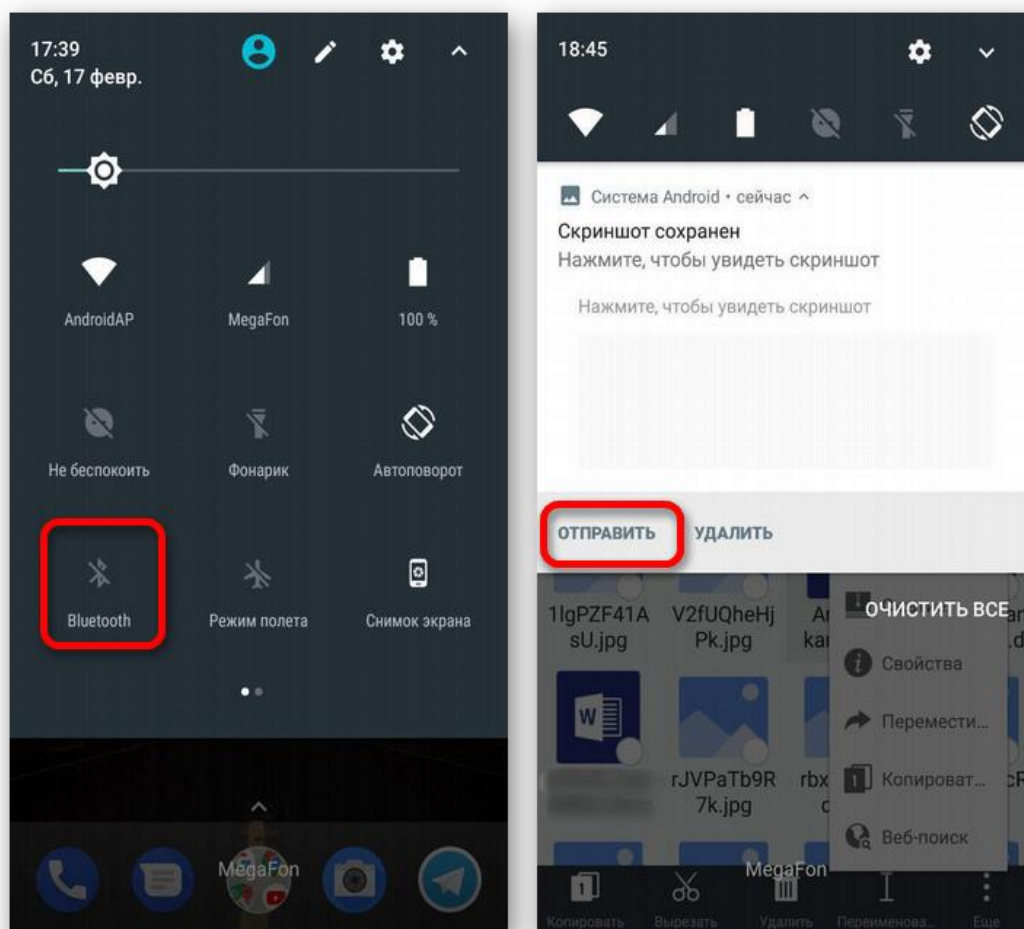
Способ 1: Запись и передача

Проще всего произвести передачу изображения с экрана телефона на другое такое же устройство путем записи и последующей загрузки получившегося видео через Bluetooth или любой другой вид соединения. В данном случае получится продемонстрировать все, что было на экране во время записи, но только в прошедшем времени. Таким образом, если вам не требуется прямая трансляция со смартфона, способ является отличным вариантом.

Чтобы выполнить описанную выше процедуру, первым делом выберите приложение для записи видео с экрана смартфона и произведите его настройку. Более подробно процесс создания роликов был нами описан в отдельной инструкции. Кроме того, стоит учитывать, что некоторые модели смартфонов по умолчанию предоставляют нужные инструменты, равно как и в ситуации со скриншотами.



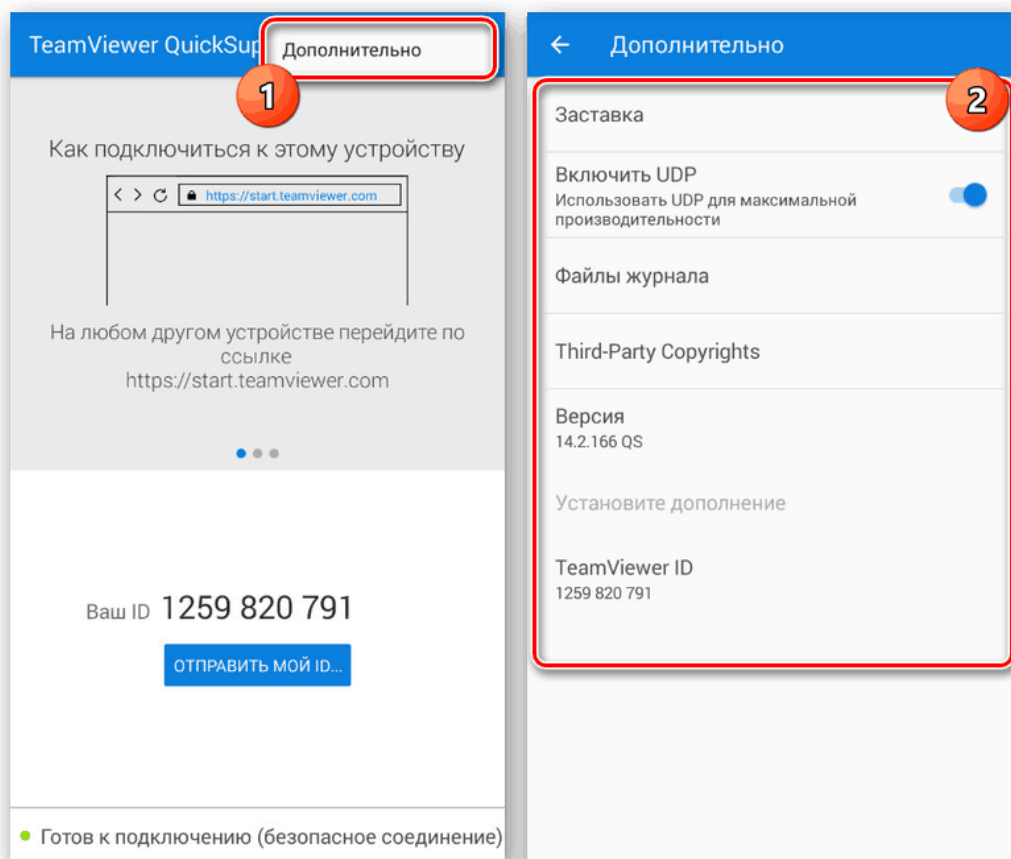
Завершив подготовку видеоряда, можно сразу же приступить к отправке видео через один из возможных типов соединения. В целом большинство актуальных вариаций было упомянуто в другой общей статье на нашем сайте и сводится к использованию Bluetooth, Android Beam или облака.



Представленных действий будет достаточно, чтобы произвести запись и передать готовый материал на другое устройство. Учтите, что зачастую получающиеся видео имеют слишком большой вес для некоторых видов соединения, из-за чего дополнительно придется выполнить сжатие или воспользоваться менее удобными способами вроде карты памяти.

Способ 2: Удаленное управление

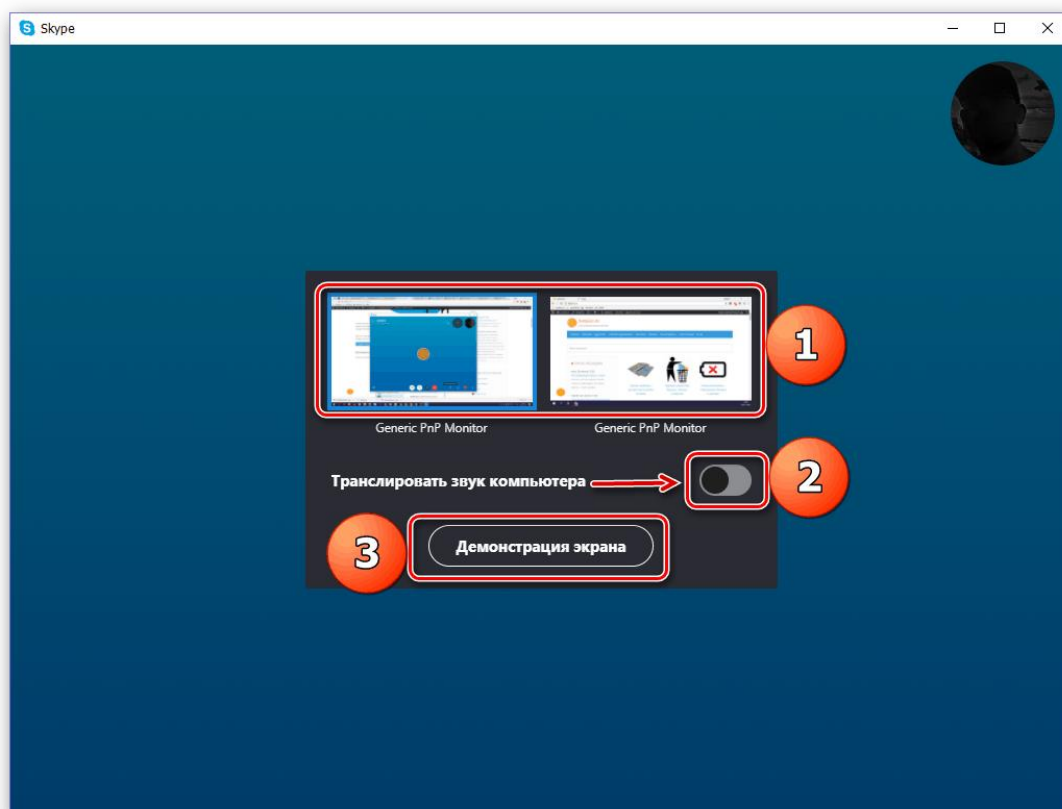
По аналогии с предыдущим методом удаленное управление изначально предназначено несколько для других целей, однако по необходимости вполне может быть использовано для трансляции экрана с одного Android-устройства на другое. К тому же многие из соответствующих приложений предоставляют не только функцию удаленного управления, но и позволяют ограничить действия подключившегося пользователя одной лишь демонстрацией.



Лучшим вариантом из существующих станет TeamViewer, давно пользующееся популярностью среди пользователей на разных платформах. Его значимым преимуществом выступает бесплатный доступ к полному набору функций во время подключения, но с некоторыми временными ограничениями. К тому же в отличие от большинства аналогов программой гарантируется безопасное и достаточно устойчивое соединение.

Способ 3: Демонстрация экрана в Skype и его аналогах

Один из достаточно удобных методов передачи изображения заключается в использовании программы Skype, выпущенной для большинства платформ, включая Android. Главные же преимущества метода сводятся к возможности голосового и текстового общения во время трансляции, высокое качество соединения и отсутствие каких-либо ограничений по времени. В целом действия практически идентичны с версией для ПК.



По сути данный метод лишь пример, так как существуют и другие, во многом более удобные, программы, точно так же поддерживающие трансляцию экрана со смартфона. Отличным примером будет Viber и более известный среди геймеров Discord.

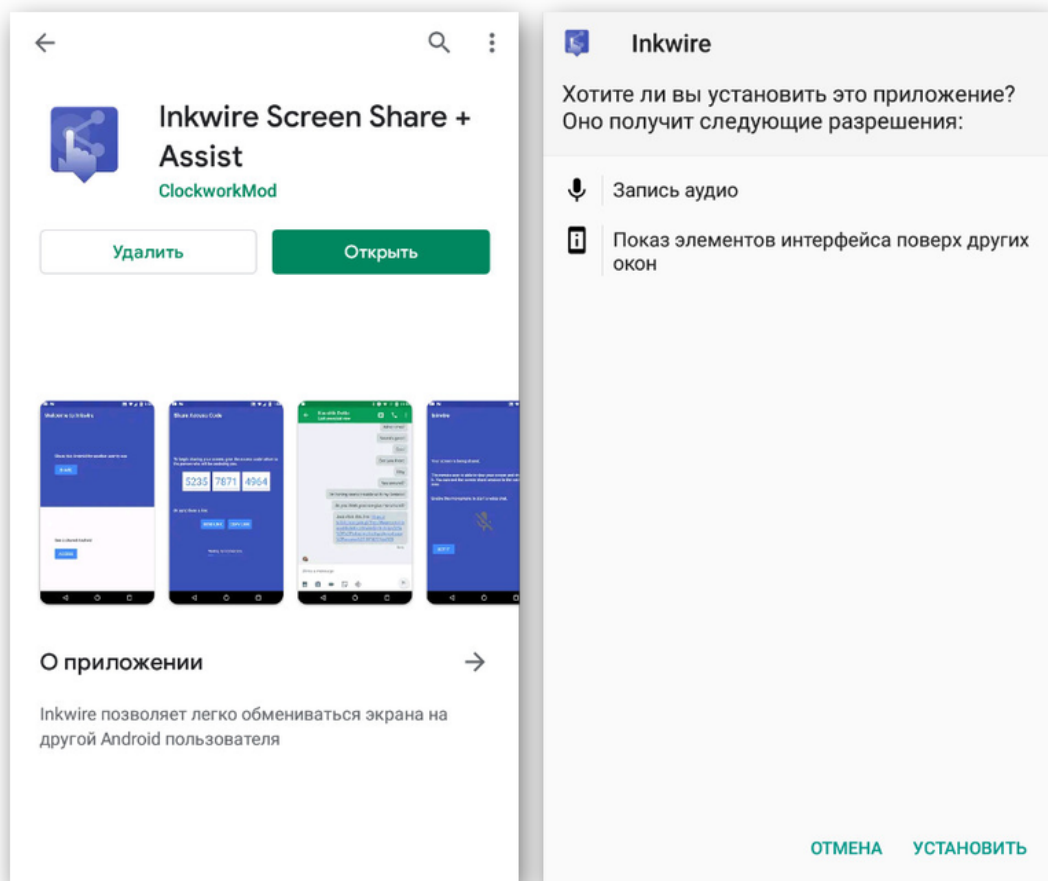
Способ 4: Inkwire Screen Share

Данное приложение является одним из лучших сторонних средств для передачи изображения с экрана одного смартфона на другой, будь то устройства на разных операционных системах или только Android. ПО имеет много общего с другой известной и ранее упомянутой программой TeamViewer, тем самым немного упрощая работу и позволяя использовать удаленный контроль на рутованных девайсах.

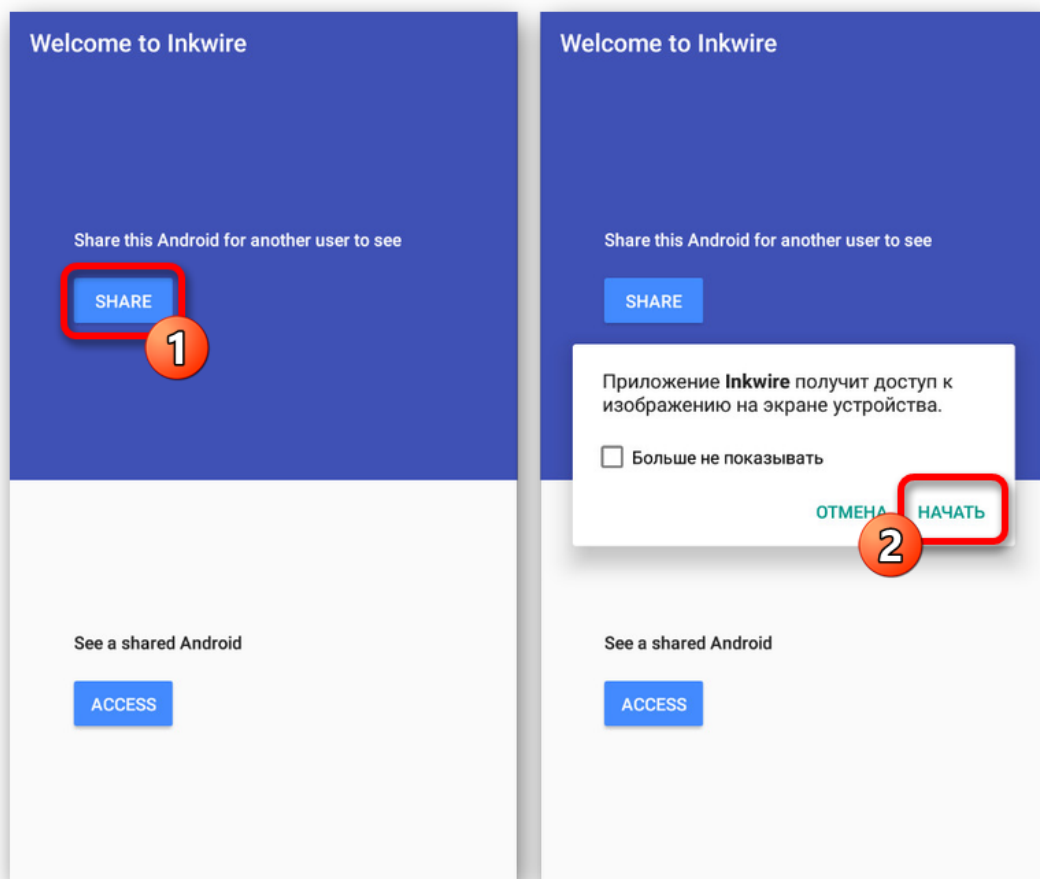
Скачать Inkwire Screen Share из Google Play Маркета

Шаг 1: Подготовка клиента

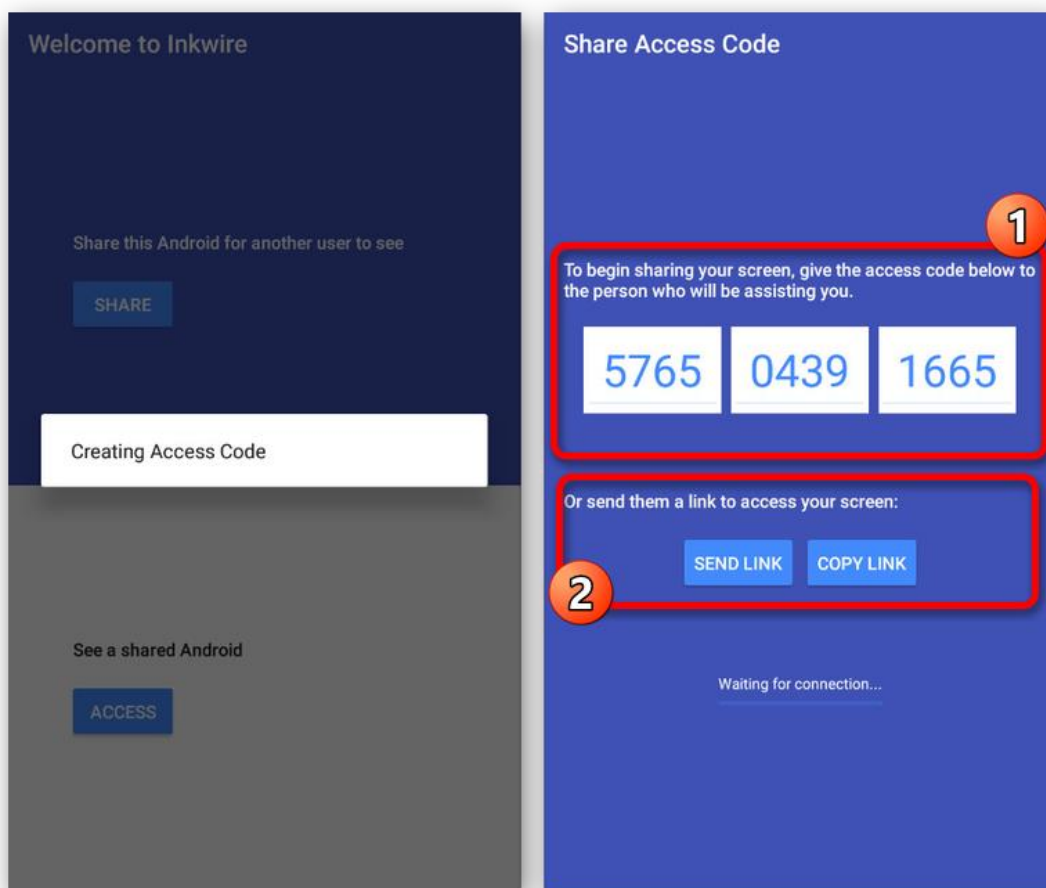
Чтобы выполнить передачу изображения с экрана, первым делом следует установить и настроить приложение на телефоне, с которого впоследствии будет проводиться трансляция. Обратите внимание, что для установки ПО необходимы минимальные разрешения из возможных.



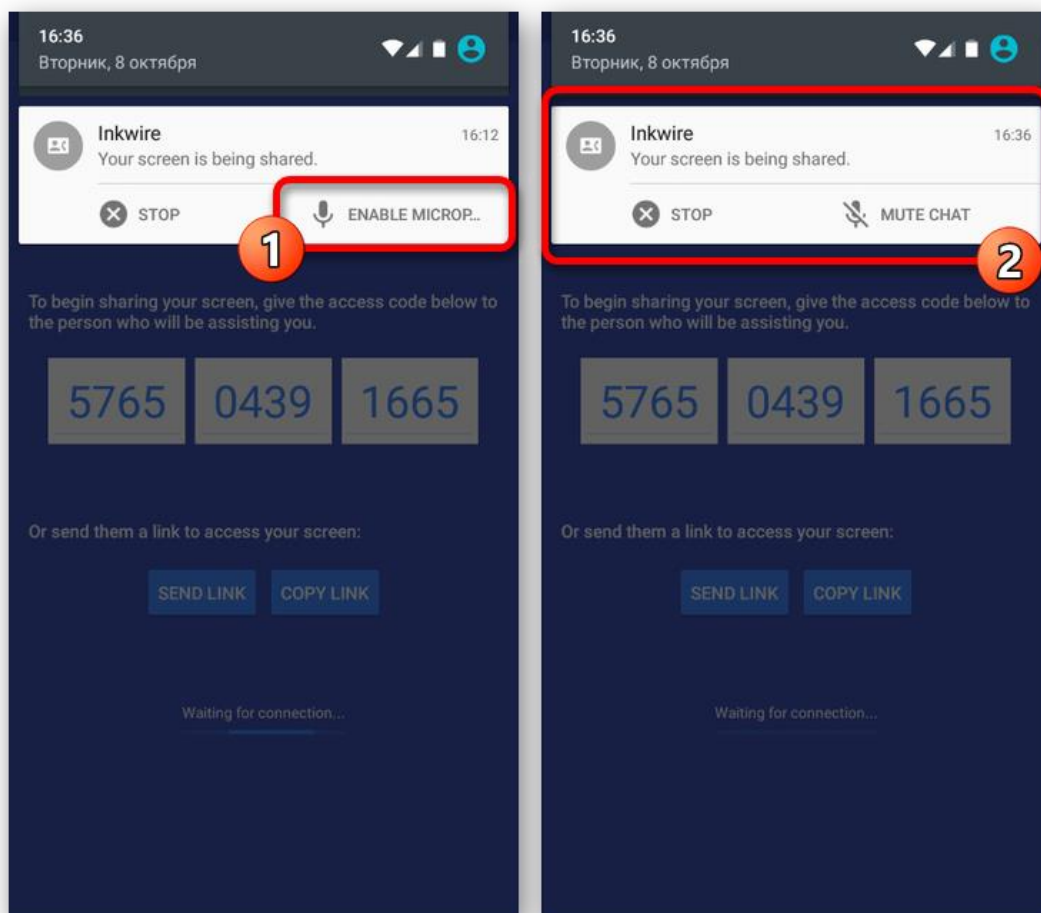
После появления стартовой страницы отобразится всего два варианта без каких-либо параметров. Коснитесь кнопки «Share» в верхнем блоке и подтвердите предварительный запрос программы на доступ к изображению с экрана смартфона.



Дождитесь завершения процедуры создания уникального идентификатора в сети приложения и, при появлении номера на экране, передайте код на другой телефон. Произвести передачу можно с помощью ссылок «Send Link» или «Copy Link».

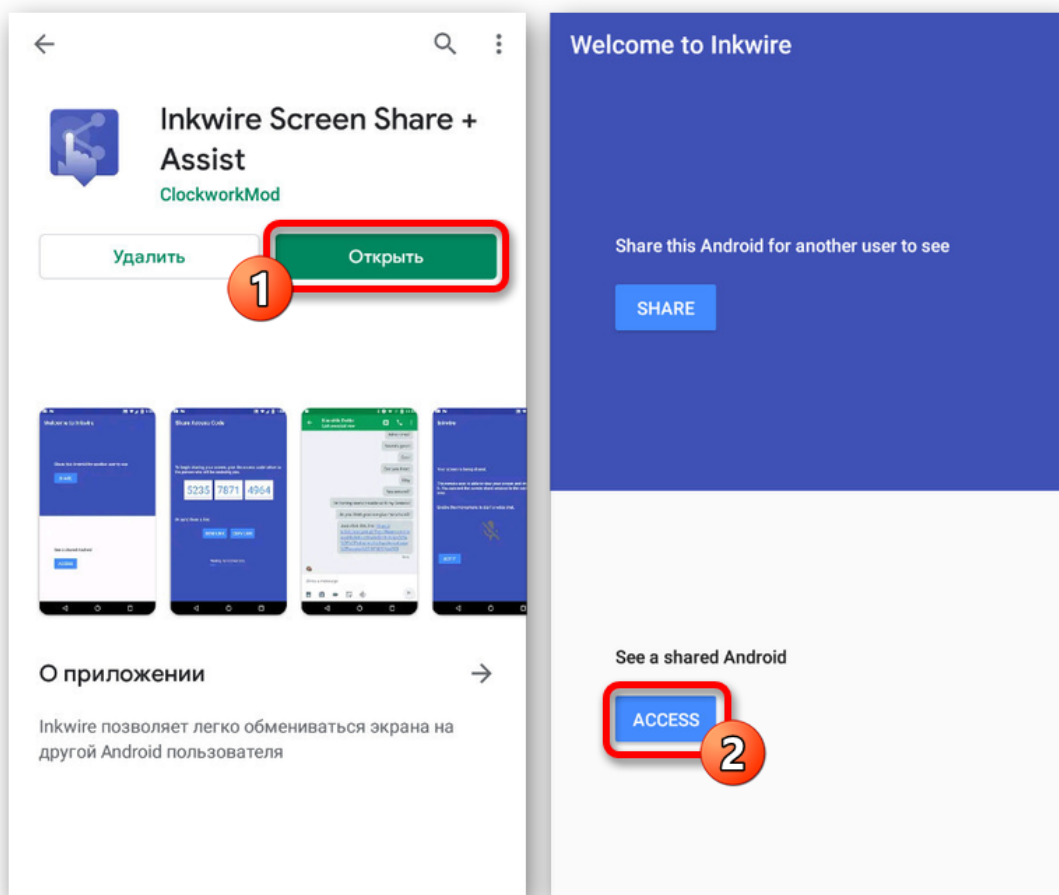


Кроме прочего, пока активен созданный код подключения, через область уведомлений можно включить микрофон. Это пригодится для обратной связи с владельцем второго смартфона.

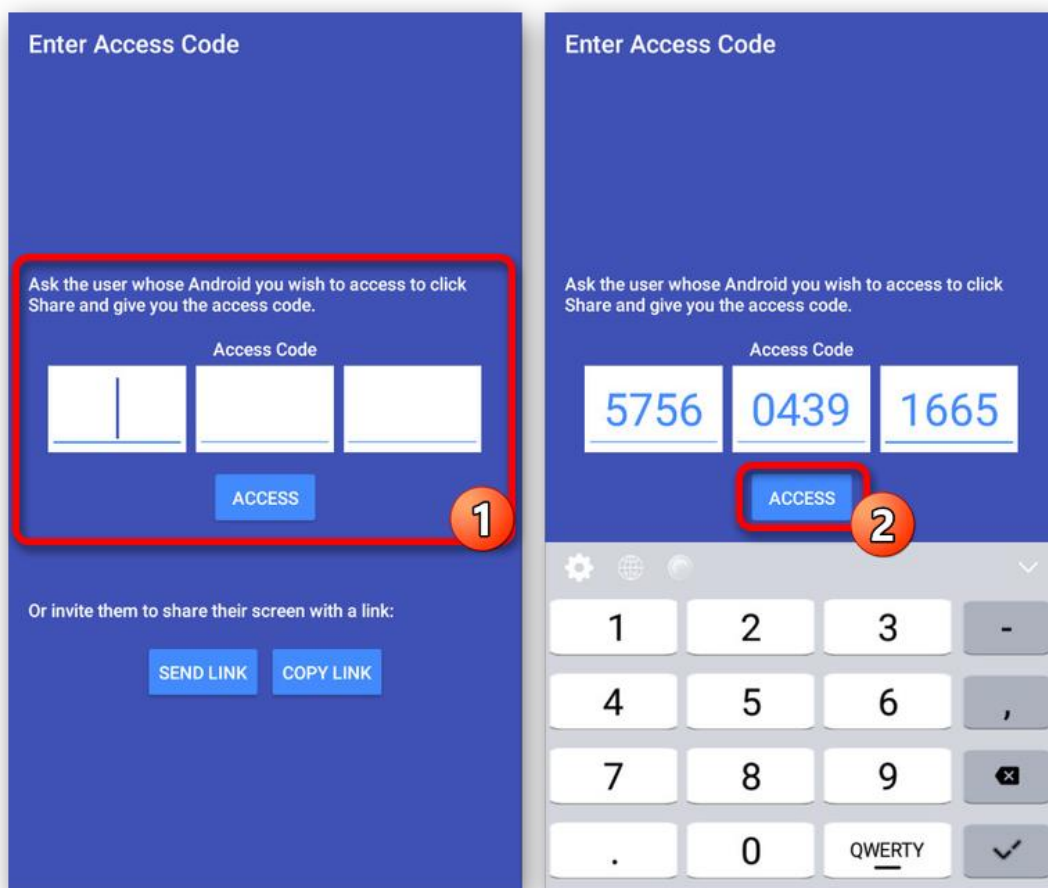


Шаг 2: Получение трансляции

Теперь необходимо снова установить приложение, но уже на другой Android-аппарат. Чтобы установить соединение с телефоном из предыдущего шага, на главной странице в нижней части воспользуйтесь кнопкой «Access».



В представленные на странице поля введите ранее полученные номера и тапните по кнопке «Access». Для успешной установки подключения нужно стабильное, хоть даже и медленное, интернет-соединение.



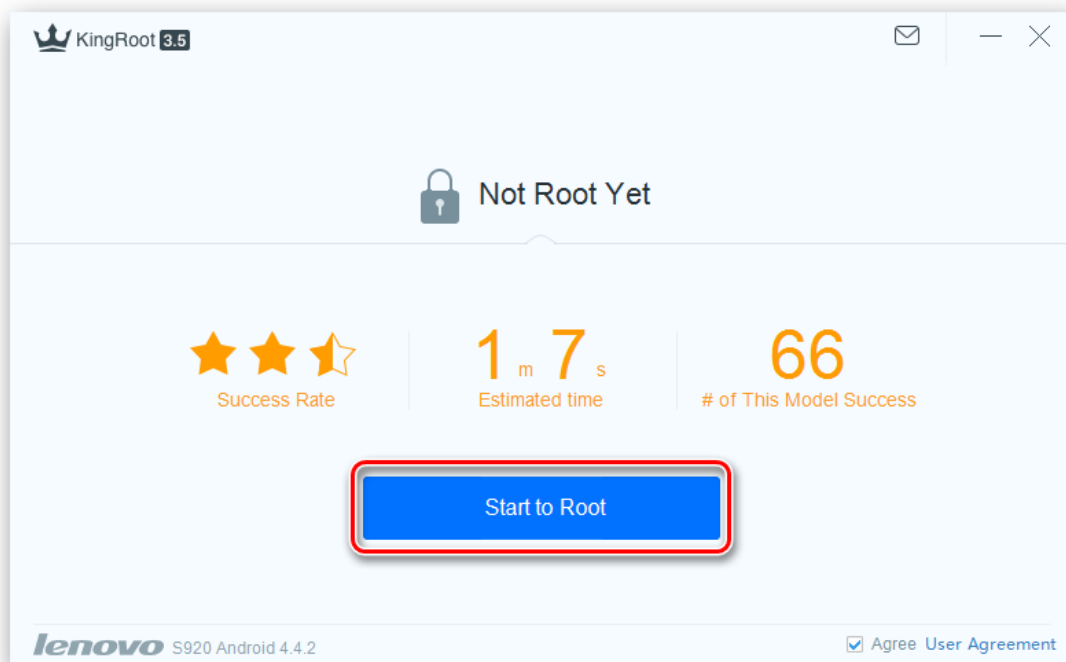
При успешном подключении появится область, отображающая содержимое на экране транслирующего устройства, включая панель уведомлений и любые открытые программы. Изучить все дальнейшие тонкости работы вроде возможности рисования на экране или голосового и текстового общения вы можете сами в процессе использования.

Как можно заметить, Inkwire Screen Share предоставляется полностью бесплатно и при этом не содержит внутренней рекламы, что позволяет с удобством транслировать изображение с экрана. Вместе с тем, к сожалению, обновления происходят довольно редко и потому на самых новых версиях ОС могут возникать неполадки.

Способ 5: Screen Stream Mirroring

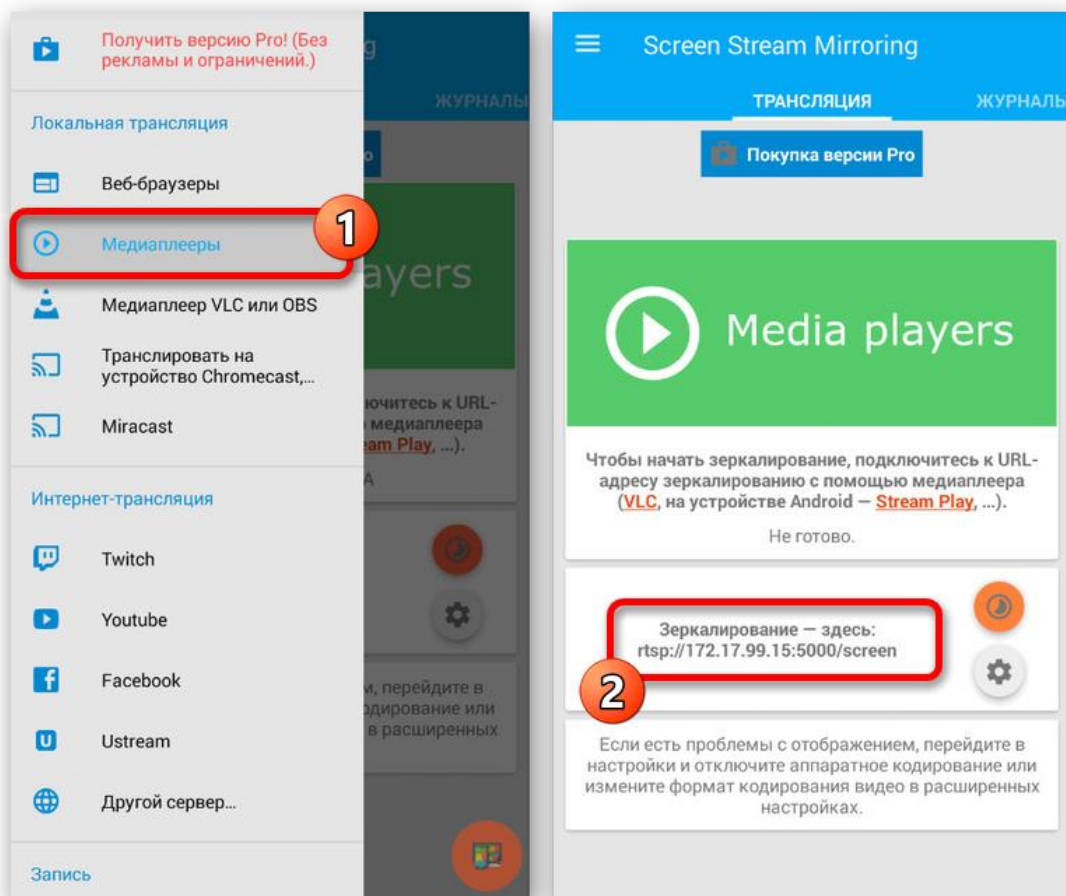
Программа Screen Stream Mirroring, как видно из названия, предназначена для создания прямых трансляций, но для куда большего количества платформ. Таким образом, например, можно создать стрим на Twitch или YouTube с Android-устройства, что отчасти так же может стать дополнительным вариантом решения. Нами же будет рассмотрена трансляция изображения между одинаковыми гаджетами по локальной сети с помощью медиаплеера.

Сперва понадобится добавить на телефон рут-права, так как без них приложение не будет работать должным образом.

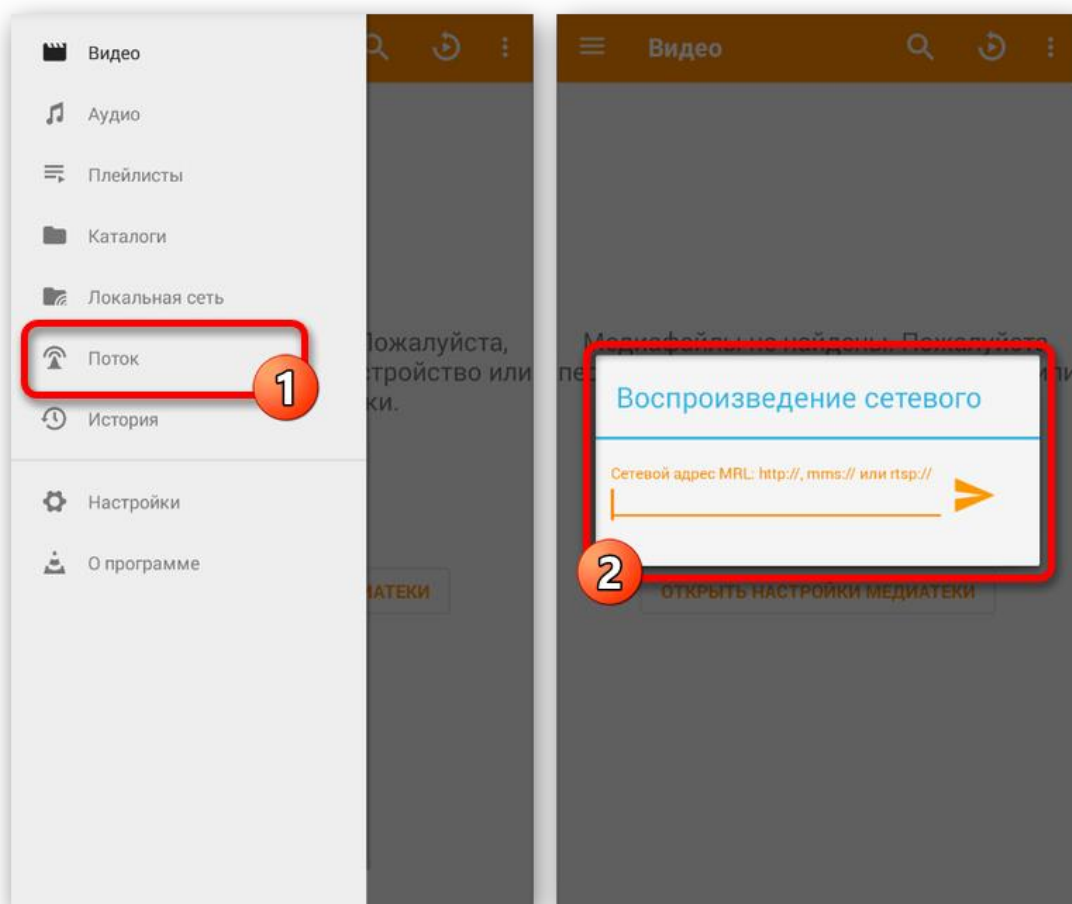


Подробнее: Получение рут-прав на Android

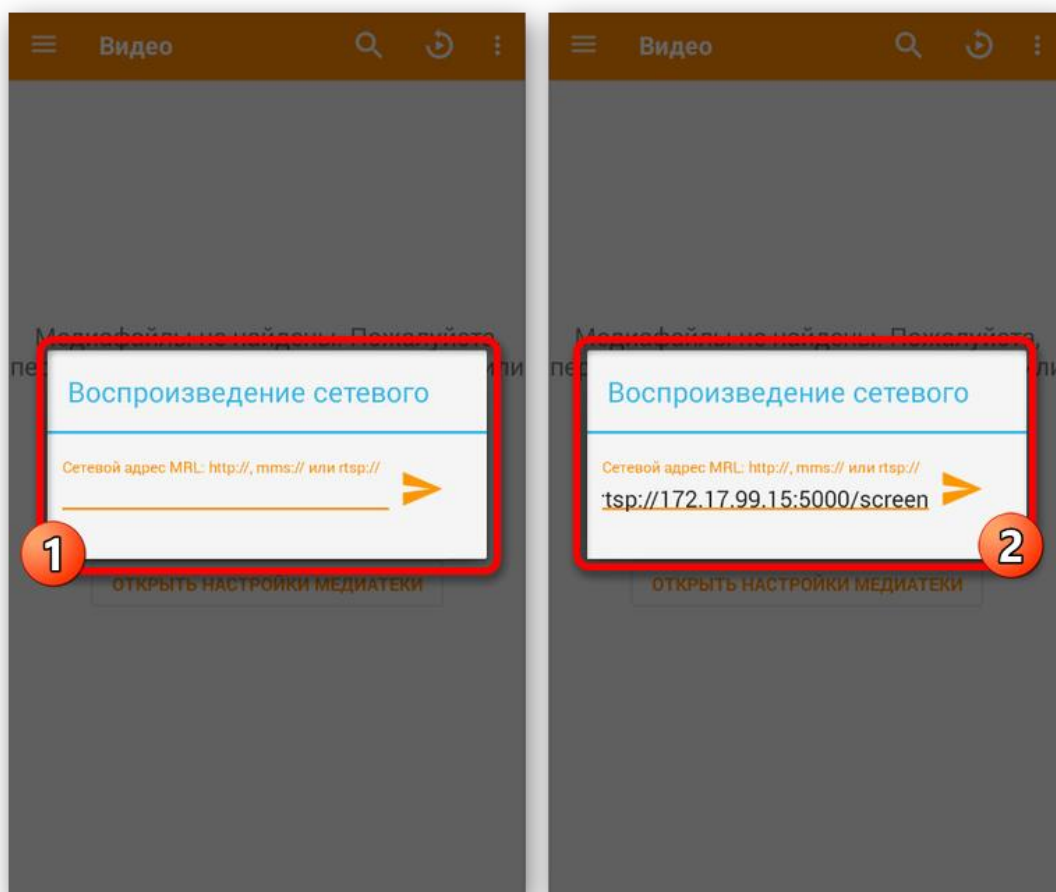
Дальше установите и запустите Screen Stream Mirroring на транслирующем устройстве, после чего через главное меню перейдите на вкладку «Медиаплееры». В результате поступит запрос на предоставление рут-прав и трансляция будет запущена.



На другой телефон установите одно из рекомендованных приложений типа Stream Play от того же разработчика или VLC Media Player. Во втором случае откройте главное меню и выберите вкладку «Поток».



В представленное текстовое поле вставьте ссылку «Зеркалирование» из Screen Stream Mirroring на транслирующем устройстве и подтвердите подключение. На этом процедура завершается, так как далее откроется медиаплеер с трансляцией экрана.



Данный метод, несмотря на необходимость покупки платной версии для доступа к полному ассортименту функций и отключения рекламы, по праву является одним из самых безопасных. К тому же, это единственный оптимальный вариант для трансляции экрана сразу многим людям.

Способ 6: BBQScreen

Как и в прошлых способах, приложение BBQScreen позволяет транслировать изображение с экрана одного девайса на другое, но уже с дополнительными параметрами. Для успешного же подключения следует заранее создать локальную сеть между смартфонами, так как здесь используется IP-адрес роутера.

Шаг 1: Скачивание и установка

Для работы уже установленного приложения потребуются рут-права. Про его наиболее удобное получение было нами сказано в прошлом способе.

Софт, к сожалению, отсутствует в Плей Маркете, из-за чего скачивание придется произвести со стороннего ресурса. Наиболее доверенным источником считается форум 4PDA.

На устройство, с которого будет произведена трансляция, необходимо установить обычное приложение «BBQScreen» по первой ссылке.

4pda.ru/forum/index.php?showto

super easy to setup, and can work through Wi-Fi, USB Tethering, or even Bluetooth tethering (best experienced with Wi-Fi or USB Tethering).

Don't hesitate to reduce screen scale for more stability, or if no frames are received.

Get and install this app on your Android device, and get the latest client app on your computer at <http://screen.bbqdroid.org>. You just have to enter your device's IP address, and it's working immediately.

Русский интерфейс: Неизвестно

Разработчик: Team BBQ

Домашняя страница: <http://screen.bbqdroid.org/>

Google Play: <https://play.google.com/store/apps/details?...droid.bbqscreen>

Google Play (клиент для связи Android-Android): <https://play.google.com...qdroid.bbqscreenclient>

Видео YouTube: <https://www.youtube.com/watch?feature=playe...p;v=Gz3fJ1ZDCQ>

Ссылки:

версия: 2.3.1 @ BBQScreen (Пост #37103021) 1

BBQScreen Android Client (BBQScreen) - клиент для управления андроид устройством с другого андроид устройства

Спойлер (+) (Прошлые версии)

Сообщение отредактировал vadeus - 09.01.15, 18:30

Причина редактирования: новая версия

4pda.ru/forum/index.php?showto

Сообщение отредактировал REF LUKK - 10.01.15, 23:25

09.01.15, 18:05 | #94

• Alex0047 ▾

Друзья 4PDA

Pen: ☐ (63834) ☐

Жалоба Цитата

BBQScreen Remote Control v2.3.1

What's New

2.3.1:

- Fixed regression on Android 4.0 to 4.2

 BBQScreen_2.3.1.apk (1,11 МБ)

18.01.15, 12:15 | #95

• Kuk07 ▾

Активные пользователи

Pen: ☐ (0) ☐

Жалоба Цитата

А звук идет на комп в этой проге ? А то у меня изображение смарта на компе есть, а вот звук только на смарте.

20.01.15, 13:00 | #96

• sinyak71 ▾

Пользователи

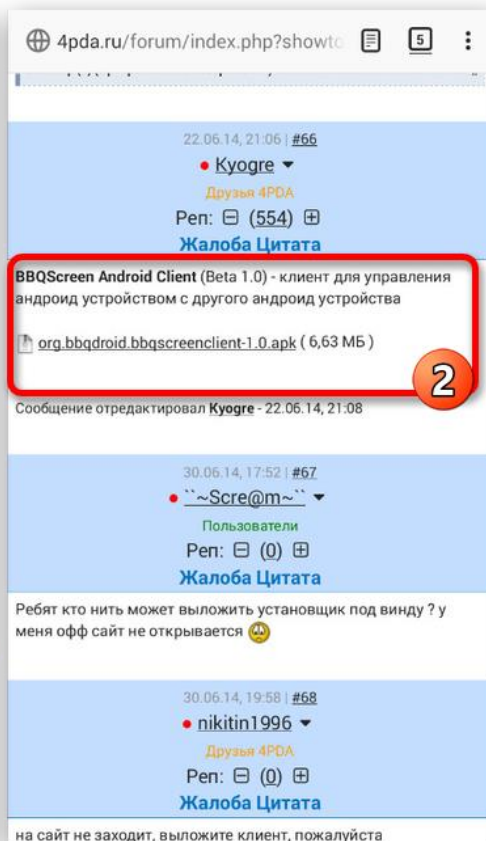
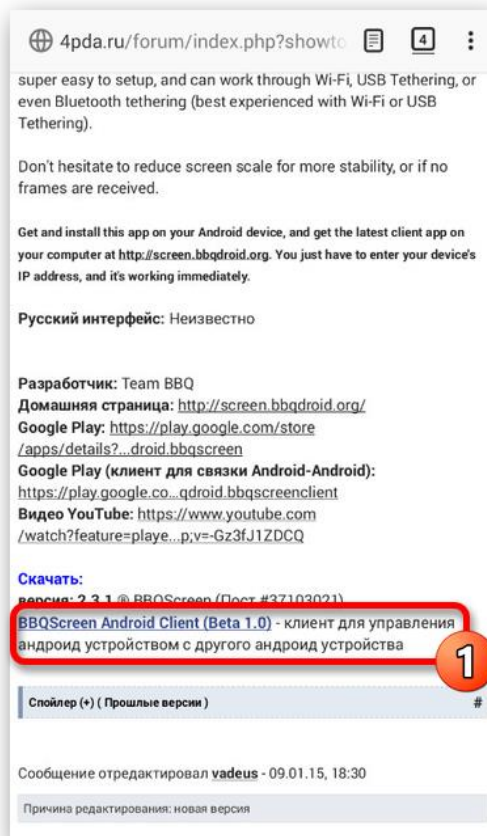
Pen: ☐ (0) ☐

Жалоба Цитата

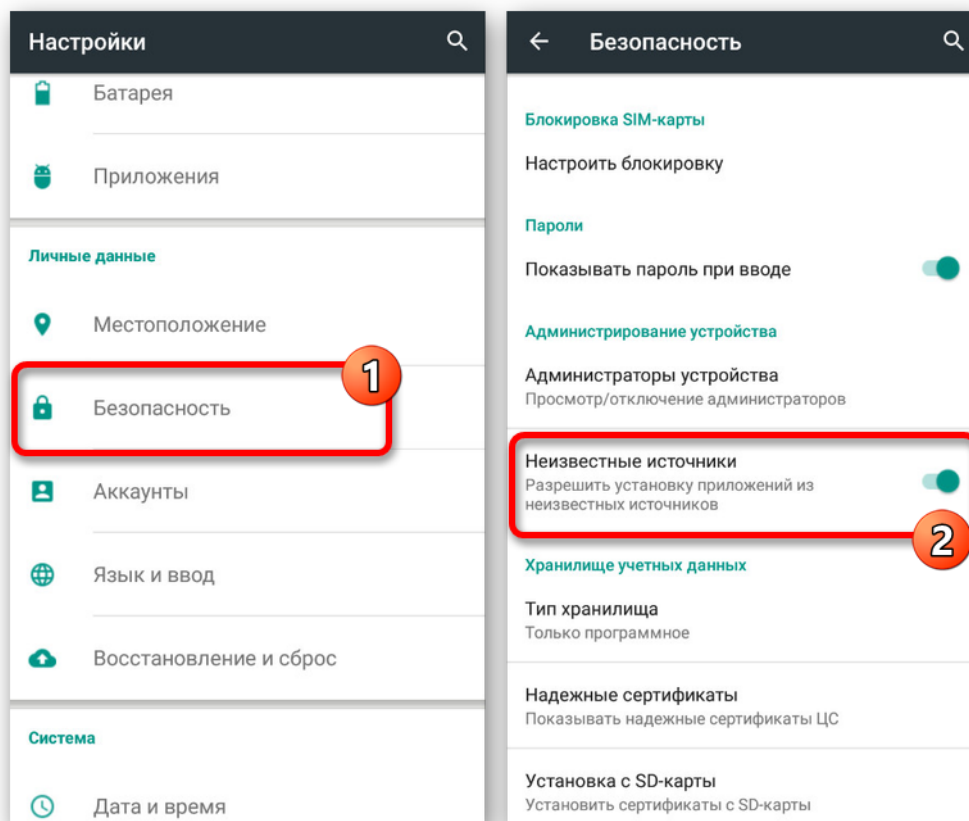
как подключить по usb

2

Другой же смартфон нужно оснастить программой «BBQScreen Android Client», так же доступной для скачивания на 4PDA.

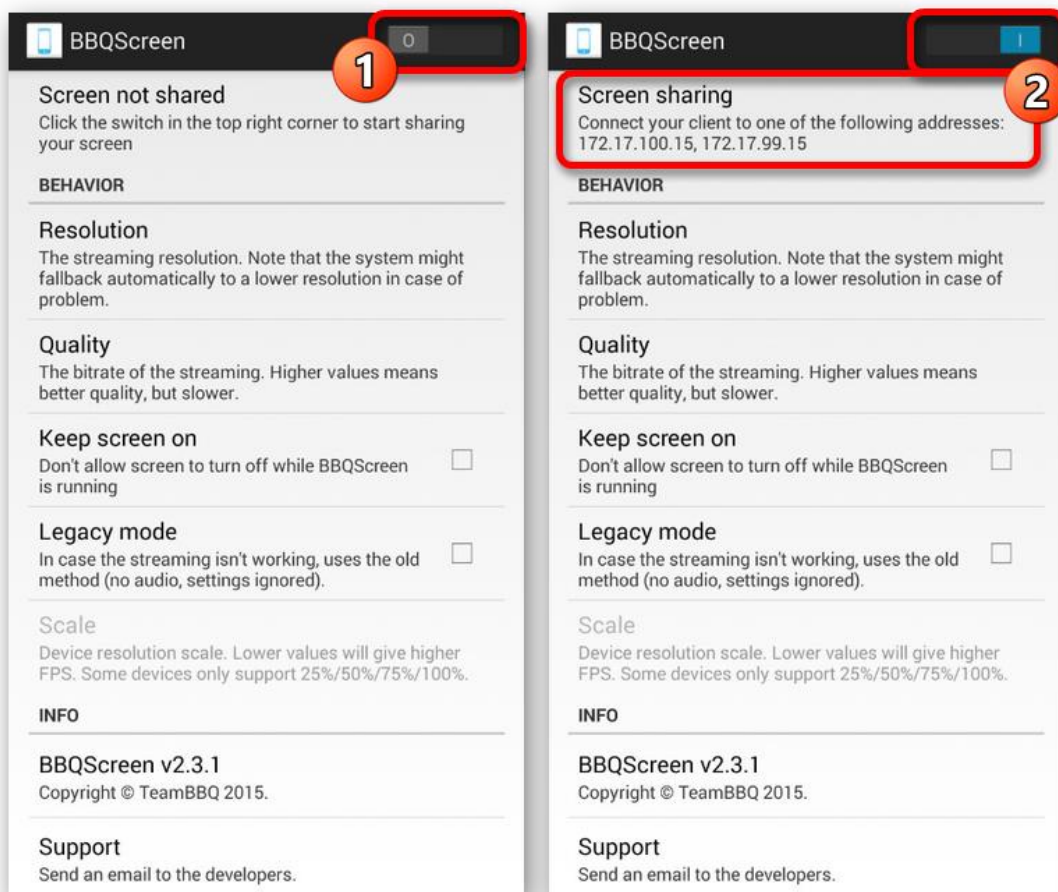


Чтобы установить загруженное ПО в обоих случаях, надо включить функцию «Неизвестные источники» в «Настройках» или разрешить распаковку для отдельного браузера. Процедура была нами описана отдельно и обязательна к ознакомлению.



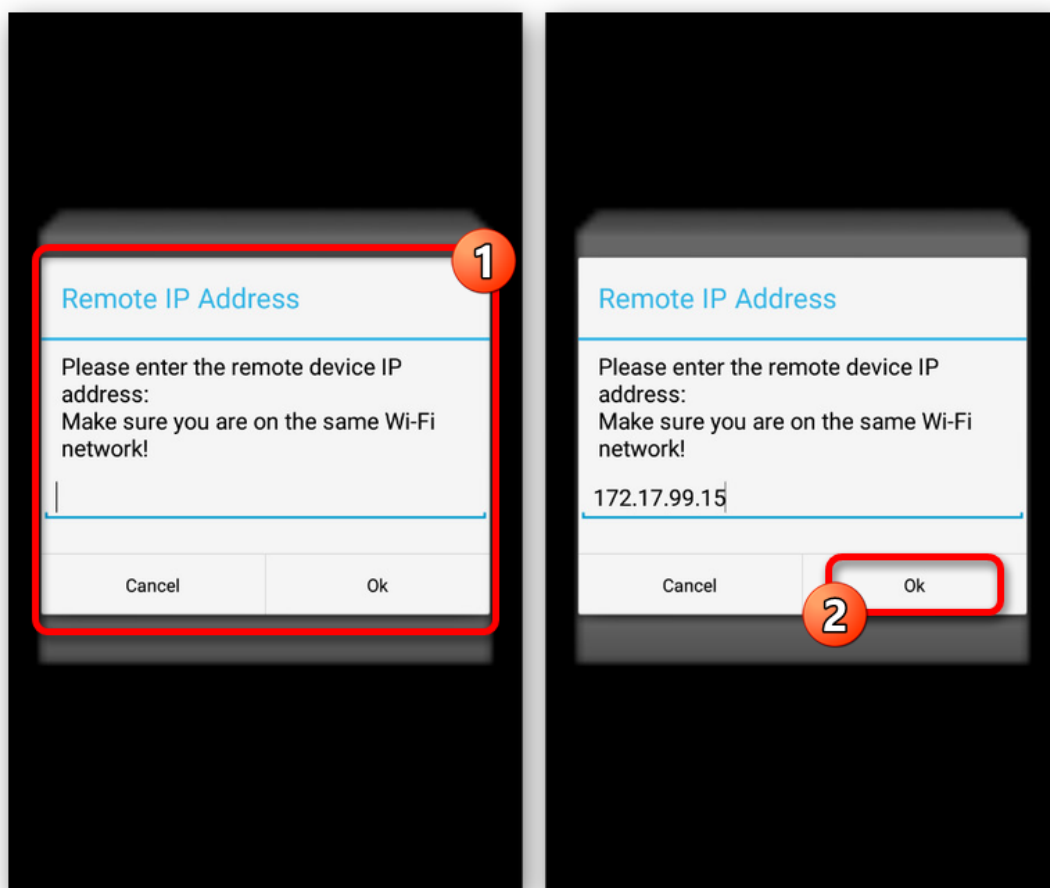
Шаг 2: Работа с приложением

Разобравшись с загрузкой, откройте приложение на транслирующем устройстве и воспользуйтесь ползунком в правом верхнем углу страницы. В результате смартфон станет доступен в локальной сети как для другого управляющего смартфона, так и для компьютера.



Тут же обратите внимание на блок «Screen sharing», доступный сразу после предыдущего шага. Цифры отсюда представляют собой локальный IP-адрес, который будет использоваться далее – запишите или запомните.

На втором устройстве запустите приложение и в автоматически открывшемся окне «Remote IP Address» укажите цифры, упомянутые ранее. Для подключения нажмите кнопку «ОК».

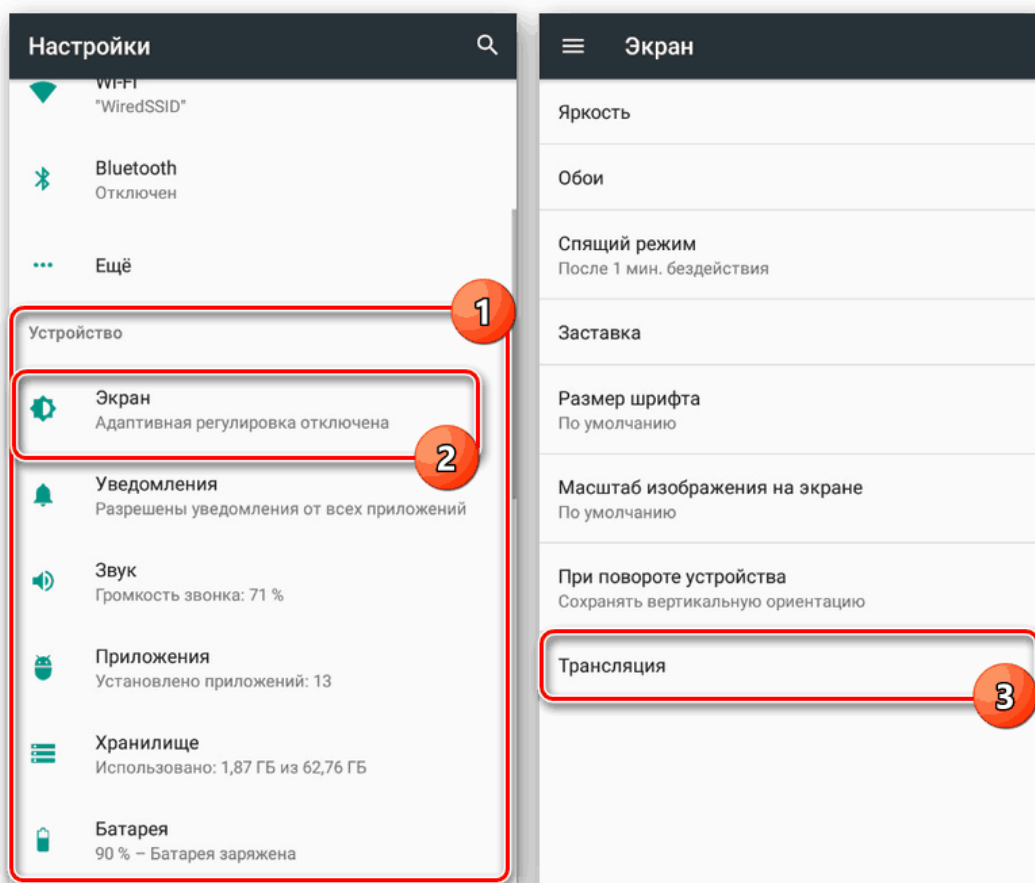


При успешной установке соединения должно будет отобразиться изображение с экрана второго смартфона. Параметры отображения необходимо регулировать именно на транслирующем устройстве, в то время как управляющее более ограничено.

Метод является отличной, хотя и частично устаревшей из-за редких обновлений, альтернативой прочим вариантам. Ко всему прочему ПО не требовательно к характеристикам системы и стабильно работает на Android 4.4+.

Способ 7: Функция трансляции

На многих смартфонах с рассматриваемой операционной системой по умолчанию присутствует функция трансляции Miracast. С ее помощью можно передать изображение с экрана девайса на большинство других девайсов, включая платформу Android, но при этом в большинстве ситуаций она используется в связке с телевизором. Более подробно данная функция была нами рассмотрена в отдельной инструкции на сайте.



Учитывайте, что далеко не все смартфоны оснащены нужной возможностью, из-за чего потребуется поиск альтернативы. Выступать в качестве замены могут специальные приложения из Плей Маркета и прочих источников, представленные выше. Мы же на этом завершаем данную статью и надеемся, что у вас получилось провести передачу изображения.

СПИСОК ЛИТЕРАТУРЫ

Список литературы согласно программе обучения:

Основная литература:

1. Дейтел, П. Android для разработчиков/ П. Дейтел, Х. Дейтел, А. Уолд. - 3-е изд. — СПб.: Питер, 2016. — 512 с.

Дополнительная литература:

1. Филлипс, Б. Android. Программирование для профессионалов/ Б. Филлипс, К. Стюарт, К. Марсикано.- 3-е изд. — СПб.: Питер, 2017. — 688 с.

Периодически – печатные издания:

1. Прикладная информатика, 2016, Том 11, №1 (61) / Прикладная информатика, Том 11, №1 (61), 2016 - Текст: электронный. - URL: <https://new.znanium.com/catalog/product/896486>