**Лекция 1 Введение**

Этот курс посвящен **разработке Android** . Но что такое Android?

Android — это [операционная система](https://en.wikipedia.org/wiki/Operating_system) . То есть, это программное обеспечение, которое соединяет оборудование с программным обеспечением и предоставляет общие услуги. Но более того, это *мобильная* операционная система: ОС, разработанная для работы на *мобильных* (читай: карманных, носимых, переносимых) устройствах.

Обратите внимание, что термин «Android» также используется для обозначения «платформы» (например, устройств, использующих ОС), а также экосистемы, которая ее окружает. Сюда входят производители устройств, использующие платформу, и приложения, которые могут быть созданы и запущены на этой платформе. Таким образом, «разработка Android» технически означает разработку приложений, работающих на определенной ОС, она также обобщается для обозначения разработки любого вида программного обеспечения, взаимодействующего с платформой.

**1.1 История Android**

Если вы собираетесь разрабатывать системы для Android, будет полезно иметь некоторое представление об этой платформе и ее истории, хотя бы для того, чтобы иметь представление о том, как и почему фреймворк разработан именно так:

* **2003** : Платформа была первоначально основана стартапом «Android Inc.», который стремился создать мобильную операционную систему (похожую на то, что в то время делала [Nokia Symbian ).](https://en.wikipedia.org/wiki/Symbian)
* **2005** : Android был приобретен компанией Google, которая хотела выйти на рынок мобильных устройств.
* **2007** : Google объявляет о создании [Open Handset Alliance](https://www.openhandsetalliance.com/) , группы технологических компаний, работающих вместе над разработкой «открытых стандартов» для мобильных платформ. В число участников вошли производители телефонов, такие как HTC, Samsung и Sony; операторы мобильной связи, такие как T-Mobile, Sprint и NTT DoCoMo; производители оборудования, такие как Broadcom и Nvidia; и другие. В настоящее время (2017) Open Handset Alliance включает 86 компаний.

Обратите внимание, это тот же год, когда вышел первый iPhone!

* **2008** : Выпущено первое устройство на базе Android: [HTC Dream](https://en.wikipedia.org/wiki/HTC_Dream) (он же T-Mobile G1)

*Характеристики:* ARM-чип 528 МГц; 256 МБ памяти; емкостный сенсор с разрешением 320x480; выдвижная клавиатура! Мнение автора: забавное маленькое устройство.

* **2010** : Выпущено первое устройство [Nexus :](https://en.wikipedia.org/wiki/Google_Nexus)[Nexus One](https://en.wikipedia.org/wiki/Nexus_One) . Это разработанные Google «флагманские» устройства, призванные продемонстрировать возможности платформы.

*Характеристики:* 1 ГГц Scorpion; 512 МБ памяти; 0,37" при разрешении 480x800 AMOLED емкостный сенсорный экран. Для сравнения, [iPhone 8 Plus](https://en.wikipedia.org/wiki/IPhone_8) (2017) имеет: ~2,54 ГГц шестиядерный A11 Bionic 64bit; 3 ГБ ОЗУ; 5,5" при разрешении 1920x1080 дисплей.

(С 2016 года эта программа была заменена линейкой устройств [Pixel ).](https://en.wikipedia.org/wiki/Pixel_%28smartphone%29)

* **2014** : Анонсирована Android Wear — версия Android для носимых устройств (часов).
* **2016** : Анонсирована Daydream, платформа виртуальной реальности (VR) для Android.

Короче говоря, Google продолжает расширять платформу, чтобы она включала в себя все больше и больше возможностей.

Android невероятно популярен! (см. например, [здесь](https://www.netmarketshare.com/operating-system-market-share.aspx?qprid=8&qpcustomd=1) , [здесь](https://www.theverge.com/2017/2/16/14634656/android-ios-market-share-blackberry-2016) и [здесь](https://www.businessinsider.com/iphone-v-android-market-share-2014-5) )

* В любом из этих анализов есть некоторые вопросы о том, что именно подсчитывается... но нас волнует то, что существует *множество* устройств Android! И более того: существует множество **различных** устройств!

### Версии Android

С момента своего выпуска Android претерпел множество «версий»:

| **Дата** | **Версия** | **Псевдоним** | **Уровень API** |
| --- | --- | --- | --- |
| Сентябрь 2008 г. | 1.0 | андроид | 1 |
| Апрель 2009 г. | 1.5 | Кекс | 3 |
| Сентябрь 2009 г. | 1.6 | Пончик | 4 |
| Октябрь 2009 г. | 2.0 | Эклер | 5 |
| Май 2010 г. | 2.2 | Фройо | 8 |
| Декабрь 2010 г. | 2.3 | Пряники | 9 |
| февраль 2011 г. | 3.0 | Соты | 11 |
| Октябрь 2011 г. | 4.0 | Сэндвич с мороженым | 14 |
| Июль 2012 г. | 4.1 | Жевательные конфеты | 16 |
| Октябрь 2013 г. | 4.4 | КитКат | 19 |
| ноябрь 2014 г. | 5.0 | Леденец | 21 |
| Октябрь 2015 г. | 6.0 | Зефир | 23 |
| авг. 2016 г. | 7.0 | нуга | 24 |
| авг. 2017 г. | 8.0 | Орео | 26 |
| авг. 2018 г. | 9.0 | Пирог | 28 |

Каждая различная «версия» имеет прозвище в честь десерта, в алфавитном порядке. Но как разработчики, мы заботимся об **уровне API** , который указывает, какие различные программные интерфейсы (классы и методы) доступны для использования.

* Вы можете ознакомиться с интерактивной версией истории через Marshmallow по адресу <https://www.android.com/history/>
* Текущую статистику использования см. на [странице https://developer.android.com/about/dashboards/ .](https://developer.android.com/about/dashboards/)

Кроме того, Android — это проект с «открытым исходным кодом», выпущенный через «Android Open Source Project» или [ASOP](https://source.android.com/) . Вы можете найти последнюю версию кода операционной системы на <https://source.android.com/> ; иногда очень полезно покопаться в исходном коде!

Хотя новые версии выпускаются довольно часто, это не означает, что все или даже многие устройства обновляются до последней версии. Вместо этого пользователи исторически получают обновленные телефоны, покупая новые устройства (в среднем каждые 18 минут в США). Кроме того, обновления, включая обновления безопасности, должны поступать от операторов мобильной связи, что означает, что большинство устройств никогда не обновляются выше версии, с которой они были куплены.

* Это проблема с точки зрения потребителя, особенно с точки зрения безопасности! Со стороны Google предпринимаются некоторые усилия, чтобы обойти это ограничение, перемещая все больше и больше сервисов платформы из базовой операционной системы в отдельное «приложение» под названием Google Play Services, а также отделяя ОС от аппаратных требований с помощью нового [проекта Treble](https://android-developers.googleblog.com/2017/05/here-comes-treble-modular-base-for.html) .
* Но для разработчиков это означает, что нельзя ожидать, что устройства будут работать под управлением последней версии операционной системы — диапазон версий, которые вам нужно поддерживать, гораздо шире, чем даже при веб-разработке! Приложения Android должны быть написаны для **разнородных устройств** .

### Юридические баталии

Обсуждая историю Android, было бы упущением не упомянуть некоторые юридические тяжбы вокруг Android. Самая крупная из них — [**Oracle против Google**](https://www.eff.org/cases/oracle-v-google) . Вкратце, Oracle утверждает, что Java API защищен авторским правом (что сами сигнатуры методов и то, как они работают, защищены), поэтому, поскольку Google использует этот API в Android, Google нарушает авторские права. В 2012 году федеральный судья Калифорнии вынес решение в пользу Google (что нельзя защищать авторские права на API). Затем это решение было отменено Федеральным окружным судом в 2014 году. Вердикт был обжалован в Верховном суде США в 2015 году, который отказался рассматривать дело. Затем оно вернулось в окружной суд, который в 2016 году постановил, что использование API компанией Google было добросовестным использованием. Это решение снова находится на рассмотрении. См. <https://www.eff.org/cases/oracle-v-google> для краткого изложения, а также <https://arstechnica.com/series/series-oracle-v-google/>

* Один интересный побочный эффект этой битвы: Android Nougat и более поздние версии используют реализацию Java [OpenJDK](https://openjdk.java.net/) вместо собственной реализации Google in-violation-but-fair-use, [см. здесь](https://venturebeat.com/2015/12/29/google-confirms-next-android-version-wont-use-oracles-proprietary-java-apis/) . Это изменение не должно оказать никакого влияния на вас как разработчика, но стоит следить за потенциальными различиями между Android и Java SE. Это также может мотивировать Android сделать Kotlin своим основным языком.

Были и другие юридические проблемы. Хотя это и не касается непосредственно Android, другое крупное судебное разбирательство — [**Apple против Samsung**](https://arstechnica.com/series/apple-v-samsung/) . В этом случае Apple утверждает, что Samsung нарушила их интеллектуальную собственность (их патенты на дизайн). Это шло туда-сюда в плане убытков и того, что считается нарушением; на момент написания этой статьи последним событием стало то, что Верховный суд рассмотрел дело и [встал на сторону Samsung](https://www.nytimes.com/2016/12/06/technology/samsung-apple-smartphone-patent-supreme-court.html?_r=0) , что нарушение патентов на дизайн не должно приводить к убыткам с точки зрения всего устройства... это сложно (автор не юрист).

Итак, в целом: Android — это растущая, развивающаяся платформа, которая встроена в социальные инфраструктуры вокруг информационных технологий и влияет на них различными способами.

## 1.2 Создание приложений

Хотя приложения Android можно разрабатывать с использованием любой среды программирования, официальной и лучшей IDE для программирования Android является [**Android Studio**](https://developer.android.com/studio/index.html) . Это ответвление приложения IntelliJ IDEA от JetBrain — Java IDE, настроенная для разработки Android. Вам нужно будет загрузить и установить эту IDE.

* Обязательно загрузите пакет Android Studio, который включает **Android SDK** (Standard Development Kit): инструменты и библиотеки, необходимые для разработки Android. В частности, SDK поставляется с рядом полезных [инструментов командной строки](https://developer.android.com/studio/command-line/index.html) . К ним относятся:
	+ **adb**, « **Android Device Bridge** », который является соединением между вашим компьютером и устройством (физическим или виртуальным). Этот инструмент используется для вывода на консоль!
	+ **emulator**, который запускает эмулятор Android: виртуальную машину устройства Android.

Я рекомендую убедиться, что на вашем компьютере есть инструменты SDK (папка toolsи ) , чтобы вы могли использовать их из командной строки. По умолчанию SDK находится в Mac и в Windows. Хотя все эти инструменты встроены в IDE, они могут быть полезными резервными вариантами для отладки или автоматизации.platform-toolsPATH/Users/$USER/Library/Android/sdkC:\Users\$USERNAME\AppData\Local\Android\sdk

### Создание проекта

Чтобы начать свое первое приложение, запустите Android Studio (открытие может занять несколько минут). На экране приветствия выберите «Начать новый проект Android Studio». Это откроет мастер, который проведет вас через настройку проекта.

* «Домен компании» должен быть уникальным для вас. Для этого курса вам следует включить ваш UW NetID, например, joelross.uw.edu.
* Запомните местоположение проекта, чтобы вы могли найти свою работу позже (например, если это Desktopили Documents).
* Если вы выберете «Включить поддержку Kotlin», приложение будет создано с помощью Kotlin, а не Java. Поскольку мы еще не изучали Kotlin, мы начнем с примера Java, чтобы небольшой код, который мы рассмотрим, был более знакомым.
* На следующем экране вам нужно будет выбрать минимальный уровень SDK , который вы хотите поддерживать, то есть, на какой самой старой версии Android ваше приложение сможет работать? Для этого курса, если не указано иное, вы должны ориентироваться как минимум на API 19 KitKat (4.4), что позволит вашему приложению работать практически на любом устройстве Android.

Обратите внимание, что минимальный SDK отличается от **целевого SDK** , который представляет собой версию Android, на которой ваше приложение было протестировано и разработано. Целевой SDK указывает, какой набор функций API вы рассмотрели/кодировали, даже если ваше приложение может откатываться к более старым устройствам, которые не включают эти функции. Во многих отношениях целевой SDK — это «самый высокий SDK, с которым я работал». В большей части этого курса мы будем ориентироваться на API 24 (Nougat).

* На следующем экране выберите, чтобы начать с Empty Activity . **Activity** — это базовый компонент Android, каждый из которых действует как «экран» или «страница» в вашем приложении. Activity более подробно обсуждается в следующей лекции.
* Оставьте имя по умолчанию ( MainActivity) на следующем экране и нажмите «Готово». Android Studio потребуется несколько минут, чтобы создать ваш проект и все настроить. (Следите за нижней строкой состояния, чтобы дождаться завершения всего процесса). После завершения у вас будет готовое (пусть и простое) приложение!

### Запуск приложения

Вы можете запустить свое приложение, нажав кнопку «Play» или «Run» в верхней части IDE. Но вам понадобится устройство Android, чтобы запустить приложение… к счастью, Android Studio поставляется с одним: виртуальным [эмулятором Android](https://developer.android.com/studio/run/emulator.html) . Эта виртуальная машина моделирует универсальное устройство с оборудованием, которое вы можете указать, хотя у него есть некоторые ограничения (например, нет сотовой связи, нет Bluetooth и т. д.).

* Хотя в последнее время эмулятор улучшился, исторически он не очень хорошо работает на Windows — он работает очень медленно (хотя и улучшается!). Лучший способ ускорить эмулятор на любой операционной системе — убедиться, что вы включили [HAXM](https://software.intel.com/en-us/android/articles/intel-hardware-accelerated-execution-manager) (менеджер ускорения Intel, который позволяет эмулятору использовать ваш графический процессор для рендеринга): это значительно ускоряет работу.

Обычно вы можете установить его через Android Studio: перейдите по ссылке, Tools > Android > SDK Managerчтобы открыть менеджер SDK для загрузки различных версий Android SDK и другого вспомогательного программного обеспечения. В разделе «SDK Tools» найдите «Intel x86 Emulator Accelerator (HAXM installer)», отметьте его и нажмите «OK» для загрузки. Обратите внимание, что вам может потребоваться выполнить дополнительную установку/настройку вручную, см. руководства ( [Mac](https://software.intel.com/en-us/android/articles/installation-instructions-for-intel-hardware-accelerated-execution-manager-mac-os-x) , [Windows](https://software.intel.com/en-us/android/articles/installation-instructions-for-intel-hardware-accelerated-execution-manager-windows) ).

* Конечно, также возможно запустить ваше приложение на [физическом устройстве](https://developer.android.com/studio/run/device.html) . Это лучший вариант для разработки (это самый быстрый и простой способ протестировать код), хотя вам понадобится USB-кабель, чтобы подключить ваше устройство к компьютеру. Для этого курса подойдет любое устройство; вам даже не понадобится сотовая связь (достаточно WiFi).

Для установки приложений для разработки на ваше устройство вам потребуется включить [параметры разработчика !](http://developer.android.com/tools/device.html)

Чтобы создать эмулятор для вашего компьютера, перейдите по ссылке, Tools > Android > AVD Managerчтобы открыть Android Virtual Device Manager. Затем вы можете выбрать «Создать виртуальное устройство…», чтобы запустить мастер для указания нового эмулятора.

* Pixel **2** — хороший выбор аппаратного профиля. Nexus 5X также разумен для «старого» устройства.
* На данный момент вам нужно использовать системный образ для Nougat API 24, и почти наверняка на оборудовании x86 (Intel). Убедитесь, что выбрали тот, который включает API Google (чтобы у вас был доступ к специальным библиотекам Google).
* Расширенные настройки можно использовать для указания таких вещей, как камера и принимает ли она ввод с клавиатуры (должно быть включено по умолчанию). Эти настройки всегда можно изменить позже.

После загрузки эмулятора вы можете провести пальцем по экрану, чтобы разблокировать его... и вскоре после этого ваше приложение должно загрузиться и запуститься!

Обратите внимание: если вы не знакомы с устройствами Android, вам следует поэкспериментировать с интерфейсом, чтобы привыкнуть к языку взаимодействия, например, к тому, как нажимать/проводить пальцем/перетаскивать/долго нажимать элементы для использования приложения.

## 1.3 Исходный код приложения

Android Studio по умолчанию создаст кучу файлов проекта — почти все из которых для чего-то используются. По умолчанию он покажет ваш проект, используя представление **Android** , которое организует файлы тематически. Если вместо этого вы перейдете в представление **Project** , вы увидите, как выглядит фактическая файловая система (хотя мы обычно придерживаемся представления Android).



Android Studio. Выбор «вида» отмечен красным.

В представлении Android файлы организованы следующим образом:

* app/папка содержит исходный код нашего приложения
	+ manifests/содержит файлы **манифеста Android** , которые являются своего рода файлом «конфигурации» для приложения
	+ java/содержит исходный код **Java** для вашего проекта. Здесь находится «логика» приложения. (Оно все равно вызывается, java/даже если содержит код Kotlin!)
	+ res/содержит файлы **ресурсов XML** , используемые в приложении. Здесь мы будем размещать информацию о макете/внешнем виде
* Gradle Scriptsсодержит скрипты для инструмента сборки [Gradle](https://gradle.org/) , который используется для компиляции исходного кода для установки на устройство.

Каждый из этих компонентов будет рассмотрен более подробно ниже.

### XML-ресурсы

Папка res/содержит файлы **ресурсов** . Файлы ресурсов используются для определения пользовательского интерфейса и других медиаресурсов (изображений и т. д.) для приложения. Использование отдельных файлов для определения интерфейса приложения, а не тех, которые используются для логики приложения (код Java), помогает разделить внешний вид и поведение. Для сравнения с веб-программированием: ресурсы содержат содержимое HTML/CSS, в то время как код Java будет содержать то, что обычно пишется на JavaScript.

Подавляющее большинство файлов ресурсов указано в [**XML**](https://en.wikipedia.org/wiki/XML) ( **Extensible Markup** Language ). XML имеет точно такой же синтаксис, как и HTML, но вы можете создавать свои собственные теги с любыми семантическими значениями, которые вам нужны. За исключением **того** , что мы будем использовать теги, которые создал и предоставил Android: поэтому определение интерфейса приложения Android будет во многом похоже на определение веб-страницы, но с новым набором элементов. Обратите внимание, что этот курс предполагает, что вы немного знакомы с HTML или XML, но если нет, вы должны быть в состоянии вывести синтаксическую структуру из примеров.

Существует большое количество различных видов ресурсов, которые организованы в различные папки:

* res/drawable/: содержит графику (PNG, JPEG и т. д.), которая будет «нарисована» на экране
* res/layout/: содержит XML-файлы макета пользовательского интерфейса для содержимого приложения
* res/mipmap/: содержит файлы иконок запуска в разных разрешениях для поддержки разных устройств
* res/values/: содержит определения XML для общих констант

Существуют и другие виды ресурсов: подробности см. в разделах [Доступные ресурсы](https://developer.android.com/guide/topics/resources/available-resources.html) или [*Ресурсы и макеты .*](https://info448.github.io/resources-and-layouts.html#resources-and-layouts)

Наиболее распространенным ресурсом, с которым вам придется работать, являются ресурсы **макета** , представляющие собой XML-файлы, определяющие визуальный макет компонента (например, HTML для веб-страницы).

Если вы открываете файл макета (например, activity\_main.xml) в Android Studio, по умолчанию он будет показан в представлении «Дизайн». Это представление позволяет использовать графическую систему для компоновки вашего приложения, аналогично тому, что вы можете сделать со слайдом PowerPoint. Щелкните вкладку «Текст» внизу, чтобы переключиться на представление XML-кода.

* Использование представления дизайна не приветствуется многими разработчиками для исторических ресурсов, даже несмотря на то, что оно становится более мощным с последующими версиями Android Studio. Часто бывает чище и эффективнее писать макеты и содержимое в прямом XML-коде. Это своего рода та же разница между написанием собственного HTML и использованием чего-то вроде DreamWeaver или Wix для создания страницы. Хотя это и легитимные приложения, они считаются менее «профессиональными». Этот курс будет сосредоточен на XML-коде для создания макетов, а не на использовании инструмента дизайна. Подробнее о его функциях см. [здесь .](https://developer.android.com/studio/write/layout-editor)

В представлении кода вы можете увидеть XML: теги, атрибуты, значения. Элементы вложены друг в друга. Предоставленный код XML определяет макет (a <android.support.constraint.ConstraintLayout>) для организации вещей, а внутри него находится <TextView>( [представление](https://developer.android.com/reference/android/view/View.html) , представляющее некоторый текст).

* Обратите внимание, что большинство атрибутов элементов имеют **пространство имен** , например, с android:префиксом, чтобы избежать возможных конфликтов (чтобы мы знали, что говорим об Android, textа не о чем-то другом).

Атрибут android:textсодержит <TextView>некоторый текст. Вы можете изменить его и перезапустить приложение , чтобы увидеть его обновление!

Вы сможете указать, как выглядит ваше приложение, создав эти файлы макета XML. Например, попробуйте заменить <TextView>на <Button>:

**<Button**

 android:id="@+id/my\_button"

 android:layout\_width="wrap\_content"

 android:layout\_height="wrap\_content"

 android:text="Click Me!"

 **/>**

Этот XML определяет кнопку. android:textВ этом случае атрибут указывает, какой текст отображается на кнопке. [*Ресурсы и макеты*](https://info448.github.io/resources-and-layouts.html#resources-and-layouts) более подробно опишут значение других атрибутов, но вы должны быть в состоянии сделать довольно хорошее обоснованное предположение на основе имен.

* (Вы можете сохранить app:атрибуты области действия, если хотите, чтобы кнопка оставалась в центре экрана. Позиционирование будет обсуждаться в [*разделе Ресурсы и макеты*](https://info448.github.io/resources-and-layouts.html#resources-and-layouts) ).

### Манифест

Помимо файлов ресурсов , другой XML, который вам может потребоваться отредактировать, — это **Manifest File** AndroidManifest.xml , который находится в manifest/папке в представлении проекта Android. [Manifest](https://developer.android.com/guide/topics/manifest/manifest-intro.html) действует как файл «конфигурации» для приложения, определяя детали уровня приложения, такие как имя приложения, значок и разрешения.

Например, вы можете изменить отображаемое имя приложения, изменив android:labelатрибут элемента <application>. По умолчанию метка — это **ссылка** на другой ресурс , найденный в res/values/strings.xmlфайле, который содержит определения для строковых «констант». В идеале все строки, с которыми сталкивается пользователь, включая такие вещи, как текст кнопки, должны быть определены как эти константы.

Обычно вам потребуется внести как минимум одно изменение в манифест для каждого приложения (например, изменить отображаемое имя), поэтому вы должны быть с ним знакомы.

### Java-активности

Помимо использования XML для указания макетов, приложения Android пишутся на **Java** , а исходный код находится в java/папке в представлении проекта Android (во вложенной структуре папок на основе имени пакета вашего приложения). Код Java обрабатывает управление программой и логику, а также хранение и обработку данных.

Написание кода Android будет ощущаться во многом как написание любой другой программы Java: вы создаете классы, определяете методы, создаете экземпляры объектов и вызываете методы этих объектов. Но поскольку вы работаете в рамках фреймворка [**,**](https://martinfowler.com/bliki/InversionOfControl.html) есть набор кода, который уже существует для вызова определенных методов. Как разработчик, ваша задача будет заключаться в том, чтобы заполнить то, что делают эти методы, чтобы запустить ваше конкретное приложение.

* С точки зрения веба это ближе к работе с Angular (фреймворком), чем с jQuery (библиотекой).
* Написание кода на Kotlin вместо Java просто означает, что вы используете другой синтаксис для создания по сути того же кода и логики.

Таким образом, хотя вы можете и будете реализовывать «обычные» классы и модели Java в своем коде, чаще всего вы будете использовать определенный набор классов, требуемый фреймворком, что обеспечивает приложениям Android общую структуру.

Самый базовый компонент в программе Android — это **Activity** , который представляет собой один экран в приложении (см. [*Activities*](https://info448.github.io/activities.html#activities) для получения более подробной информации). Предоставленный по умолчанию MainActivityкласс является примером этого: класс расширяет [Activity](http://developer.android.com/reference/android/app/Activity.html) (на самом деле он расширяет подкласс, который поддерживает компоненты Material Design), позволяя вам вносить собственные настройки в поведение приложения в рамках Android.

В этом классе мы переопределяем унаследованный onCreate()метод, который вызывается фреймворком при запуске Activity — таким образом, этот метод действует немного как конструктор для класса (хотя более подробное обсуждение см. [*в Activities ).*](https://info448.github.io/activities.html#activities)

Мы вызываем superметод, чтобы убедиться, что фреймворк делает все необходимое, а затем setContentView()указать, каким должно быть содержимое (внешний вид) Activity. Это передается в значении из чего-то, называемого R. R— это класс, который **генерируется во время компиляции** и содержит константы, которые определяются файлами «ресурсов» XML! Эти файлы преобразуются в переменные Java, к которым мы можем получить доступ через класс R. Таким образом, R.layout.activity\_mainссылается на activity\_mainмакет, найденный в res/layouts/папке. Вот как Android узнает, какой файл макета отображать на экране.

#### Далвик

На настольном компьютере код Java необходимо скомпилировать в байт-код и запустить на виртуальной машине ( [виртуальной машине Java (JVM)](https://en.wikipedia.org/wiki/Java_virtual_machine) ). До Lollipop (5.0) код Android запускался на виртуальной машине под названием [**Dalvik**](https://en.wikipedia.org/wiki/Dalvik_%28software%29) .

* Интересный факт для людей с образованием в области компьютерных наук: Dalvik использует [архитектуру на основе регистров,](https://en.wikipedia.org/wiki/Register_machine) а не стека!

Разработчик писал код Java , который затем компилировался в байт-код JVM , который затем транслировался в байт-код DVM (виртуальная машина Dalvik), который можно было запустить на устройствах Android. Этот байт-код DVM хранится в файлах .dexили .odex(«[Optimized] Dalvik Executable»), которые загружались на устройство. Процесс преобразования кода Java в dexфайлы называется **«dexing»** (поэтому скомпилированный и преобразованный код называется «dexed»).

Dalvik включает JIT («Just In Time») компиляцию в машинный код, который работает намного быстрее, чем код, интерпретируемый виртуальной машиной, подобно [Java HotSpot](http://www.oracle.com/technetwork/articles/javase/index-jsp-136373.html) . Этот машинный код быстрее, поскольку для взаимодействия с реальным оборудованием (через ОС) не требуется шага трансляции.

Однако, начиная с Lollipop (5.0) , Android вместо этого использует [Android Runtime (ART)](https://source.android.com/devices/tech/dalvik/) для запуска кода. Самое большое преимущество ART заключается в том, что он компилирует .dexбайт-код в машинный код при установке с помощью компиляции AOT («Ahead of Time»). ART продолжает принимать .dexбайт-код для обратной совместимости (поэтому происходит тот же процесс dexing), но код, который фактически устанавливается и запускается на устройстве, является машинным. Это позволяет приложениям выполняться быстрее, но за счет более длительного времени установки — и поскольку вы устанавливаете приложение только один раз, это довольно выгодная сделка.

(Kotlin также компилируется в .dexбайт-код, поэтому оказывается в том же месте, что и Java).

После сборки приложение Android (исходный код, dexed bytecode и любые некодовые ресурсы, такие как изображения) упаковываются в файл **.apk**. По сути, это zip-файлы (используется то же сжатие gzip); если вы переименуете файл, .zipто сможете его распаковать! .apkЗатем файл [криптографически подписывается,](https://en.wikipedia.org/wiki/Digital_signature) чтобы указать его подлинность, и либо «загружается» на устройство, либо загружается в App Store для развертывания.

* Короче говоря: подписанный .apkфайл — это по сути «исполняемая» версия вашей программы!
* Google находится в процессе прекращения поддержки .apkфайлов и замены их на [Android App Bundles](https://developer.android.com/platform/technology/app-bundle/) . Они содержат весь скомпилированный исходный код, но переносят генерацию APK и подписание приложений в Play Store.
* Обратите внимание, что код фреймворка приложения Android (например, базовый Activityкласс) на самом деле «предварительно DEXed» (предварительно скомпилирован) на устройстве; когда вы пишете код, вы компилируете с пустыми заглушками кода (вместо того, чтобы включать эти классы в ваш .apk)! При этом любые другие сторонние библиотеки, которые вы включите, будут скопированы в ваше собранное приложение, что может увеличить размер его файла как для установки, так и на устройстве.
* Полезно то, что, поскольку код Android в любом случае пишется для виртуальной машины, приложения Android можно разрабатывать и создавать на любой операционной системе компьютера (в отличие от некоторых других мобильных ОС…).

### Скрипты Gradle

Подводя итог, после написания исходного кода Java и XML, для «создания» и запуска приложения вам необходимо:

1. Генерация исходных файлов Java (например, R) из файлов ресурсов XML
2. Скомпилируйте код Java (или Kotlin) в байт-код JVM
3. «dex» байт-код JVM в байт-код Dalvik
4. Упакуйте код и другие активы в.apk
5. Криптографически подпишите .apkфайл, чтобы авторизовать его
6. Перенесите его .apkна свое устройство, установите и запустите!

Это много шагов! К счастью, IDE делает это за нас с помощью автоматизированного инструмента сборки Gradle [**.**](https://gradle.org/) Такие инструменты позволяют вам, по сути, указать одну команду, которая выполнит все эти шаги одновременно.

Можно настроить скрипт сборки, изменив файлы скрипта Gradle, которые находятся в Gradle Scriptsпапке в представлении проекта Android. По умолчанию их много:

* build.gradle: Сборка Gradle верхнего уровня; уровень проекта (для сборки!)
* app/build.gradle: Сборка Gradle, специфичная для приложения. **Используйте ее для настройки вашего проекта!** , например, для добавления зависимостей или внешних библиотек.
	+ Например, здесь мы можем изменить Target SDK .
* proguard-rules.pro: конфигурация для релизной версии (минимизация, обфускация и т. д.).
* gradle.properties: Настройки сборки, специфичные для Gradle, общие
* local.properties: настройки, локальные только для этого компьютера
* settings.gradle: Настройки сборки, специфичные для Gradle, общие

Обратите внимание, что старые приложения Android разрабатывались с использованием [Apache ANT](http://ant.apache.org/) . Скрипт сборки хранился в build.xmlфайле, build.propertiesсодержащем local.propertiesглобальные и локальные настройки сборки. Хотя Gradle в наши дни более распространен, вам следует знать об ANT для устаревших целей.

Также можно использовать Gradle для [сборки и установки вашего приложения из командной строки,](http://developer.android.com/tools/building/building-cmdline.html) если вы хотите. Вам нужно убедиться, что у вас есть подключенное и работающее устройство (физическое или виртуальное). Затем из папки проекта вы можете собрать и установить ваше приложение с помощью

*# use the provided Gradle wrapper to run the `installDebug` script*

./gradlew installDebug

Вы также можете запустить приложение из командной строки с помощью команды

*# use adb to start*

adb shell am start -n package.name/.ActivityName

Вы можете запустить обе эти команды последовательно, соединив их с помощью &&(который замыкает команду, поэтому она запустится только в случае успешной сборки).

## 1.4 Ведение журнала и ADB

В Android мы не можем использовать, System.out.println()потому что у нас на самом деле нет консоли для печати! Точнее, устройство (на котором запущено приложение) не имеет доступа к [стандартному выводу](https://en.wikipedia.org/wiki/Standard_streams#Standard_output_.28stdout.29) ( stdout), что Java и подразумевает под System.out.

* Получить доступ к можно stdoutс adbпомощью adb shell stop; adb shell setprop log.redirect-stdio true; adb shell start, но это определенно не идеальный вариант.

Вместо этого Android предоставляет систему [ведения журнала](https://developer.android.com/studio/debug/am-logcat.html) , которую мы можем использовать для записи отладочной информации, и которая автоматически доступна через adb(Android Debugging Bridge). Зарегистрированные сообщения можно фильтровать, категоризировать, сортировать и т. д. Ведение журнала также можно отключить в производственных сборках по соображениям производительности (хотя часто этого не происходит, потому что люди совершают ошибки).

Для выполнения этой регистрации мы будем использовать класс [1.](https://info448.github.io/introduction.html#fn1) Этот класс включает в себя ряд методов, которые в основном предназначены для печати в файле журнала устройства, который затем доступен через .[android.util.Log](https://developer.android.com/reference/android/util/Log.html)staticprintlnadb

* Вам понадобится **import**класс Log!

Вы можете сделать так, чтобы Android Studio автоматически добавляла importдля класса, выбрав имя этого класса и нажав alt-return(вам будет предложено, если имя класса неоднозначно). Для лучшего результата включите «Добавлять недвусмысленные импорты на лету» в настройках IDE.

Файл журнала устройства хранится постоянно… в некотором роде. Это файл размером 16 КБ, но он общий для всей системы. Поскольку каждое приложение и часть системы пишет в него, он быстро заполняется. Поэтому фильтрация/поиск становятся важными, и вы пытаетесь просматривать журнал (и отлаживать свое приложение) в режиме реального времени!

### Методы ведения журнала

Logпредоставляет методы, соответствующие разному уровню приоритета (важности) записываемых сообщений. От низкого к высокому приоритету:

* **Log.v()**: VERBOSE output. Это наиболее подробный вывод для повседневных сообщений. Это часто используемый по умолчанию уровень для ведения журнала. В идеале Log.v()вызовы должны компилироваться в приложение только во время разработки и удаляться для производственных версий.
* **Log.d()**: DEBUG вывод. Это предназначено для низкоуровневых, менее подробных сообщений (но все еще уровня кода, то есть ссылающихся на определенные программные сообщения). Эти сообщения могут быть скомпилированы в код, но удаляются во время выполнения в производственных сборках через Gradle.
* **Log.i()**: INFO-вывод. Это предназначено для «высокоуровневой» информации, например, на уровне пользователя (а не для конкретики кода).
* **Log.w()**: WARN вывод. Для предупреждений
* **Log.e()**: Вывод ERROR. Для ошибок
* А если посмотреть на [API](https://developer.android.com/reference/android/util/Log.html) … Log.wtf()!

Эти разные уровни используются для того, чтобы помочь «отфильтровать шум». Таким образом, вы можете просматривать только ошибки, ошибки и предупреждения, ошибки, предупреждения и информацию... вплоть до просмотра всего с подробным описанием. Регистрируется огромное количество информации, поэтому фильтрация действительно помогает!

Каждый Logметод принимает два Stringsпараметра. Второй — это сообщение для печати. ​​Первый — это «тег» — строка, которая добавляется к выводу, по которому можно выполнять поиск и фильтрацию. Обычно этот тег — имя приложения или класса (например, «AndroidDemo», «MainActivity»). Распространенной практикой является объявление константы, которую TAGможно использовать во всем классе:

**private** static final String TAG = "MainActivity";

### ЛогКэт

Вы можете просматривать журналы через adb(отладочный мост) и службу, называемую Logcat(от «log» и «conCATenation», поскольку она объединяет журналы). Самый простой способ проверить Logcat — использовать Android Studio. Панель браузера Logcat обычно находится в нижней части экрана после запуска приложения. Она «следует» за журналом, показывая последний вывод по мере его появления.

Вы можете использовать раскрывающийся список для фильтрации по приоритету и поле поиска для поиска (например, по тегу, если хотите). Android Studio также позволяет вам фильтровать, чтобы показывать только текущее приложение, что очень здорово. Обратите внимание, что вы можете увидеть много журналов, которые вы не создавали, включая возможные предупреждения (например, я вижу много всего о том, как OpenGL подключается к видеокарте). Это нормально !

Также возможно просматривать Logcat через командную строку с помощью adb, и включает сложные аргументы фильтрации. Подробнее см. [в Logcat Command-line Tool .](https://developer.android.com/studio/command-line/logcat.html)

* Что-то еще для проверки: заставить приложение выдать runtime Exception! Например, вы можете создать новый локальный массив и попытаться получить доступ к элементу за пределами границ. Или просто throw new RuntimeException()(что немного менее интересно). Вы можете увидеть ***трассировку стека*** в журналах?

Логирование — это фантастика и отличный метод отладки, как в том, как используются Activity, так и для любого вида ошибок (также RuntimeExceptions). Это возвращает нас к отладке printline, которая по-прежнему является законным процессом отладки, спасибо большое.

Обратите внимание, что Android Studio имеет встроенный [отладчик](https://developer.android.com/studio/debug/index.html) , если вам удобно работать с такими системами. Мы поговорим об этом подробнее в ходе будущей лабораторной работы.

### Тост

Журналы отлично подходят для отладки выходных данных, но помните, что они видны только разработчикам (ваш телефон должен быть подключен к IDE или SDK!). Если вы хотите вывести сообщение об ошибке или предупреждение для пользователя , вам нужно использовать другой метод.

Один простой и быстрый способ дать краткую визуальную обратную связь — использовать тост [**.**](http://developer.android.com/guide/topics/ui/notifiers/toasts.html) Это маленькое текстовое поле, которое на мгновение появляется в нижней части экрана, чтобы быстро отобразить сообщение.

* Это называется «Тост», потому что он всплывает!

Реализовать тосты довольно просто, как в следующем примере (из официальной документации):

Toast toast = Toast.makeText(**this**, "Hello toast!", Toast.LENGTH\_SHORT);

toast.show();

*//as one line. Don't forget to show()!*

Toast.makeText(**this**, "Hello toast!", Toast.LENGTH\_SHORT).show();

Тосты создаются с помощью Toast.makeText()метода factory (вместо вызова конструктора). Этот метод принимает три параметра: Context или то, что создает тост (см. Главу 3), текст для отображения и intконстанту, представляющую длину, на которую должен отображаться тост.

Всплывающие уведомления предназначены для предоставления информации пользователю (например, для быстрой обратной связи), но они также могут быть полезны для тестирования (хотя в конечном итоге Logcat станет вашим лучшим выбором для отладки, особенно при попытке устранения сбоев или просмотра более сложных выходных данных).

## 1.5 Добавление взаимодействия

Наконец, мы создали кнопку и обсудили, как показать визуальную информацию пользователю... так давайте же объединим это!

Как и в JavaScript, чтобы наша кнопка что-то сделала, нам нужно зарегистрировать функцию обратного вызова , которая может быть выполнена при нажатии кнопки. В Java эти функции обратного вызова предоставляются объектами «слушателя», которые могут реагировать на события (более подробное обсуждение см. [в Приложении B](https://info448.github.io/java-swing.html#java-swing) ).

Во-первых, нам нужно получить доступ к переменной, которая представляет собой , которую Buttonмы определили в XML — аналогично тому, что вы делаете document.getElementById()в JavaScript. Метод для доступа к элементу в Android называется [findViewById()](https://developer.android.com/reference/android/app/Activity.html#findViewById(int)), и может быть вызван непосредственно в Activity:

Button button = (Button)findViewById(R.id.my\_button);

В качестве аргумента мы передаем значение, определенное в автоматически сгенерированном R классе, представляющем значение кнопки id— оно основано на том, что мы поместили в <Button>атрибут android:id. Точный формат обсуждается в [*разделе Ресурсы и макеты*](https://info448.github.io/resources-and-layouts.html#resources-and-layouts) .

* Обратите внимание, что метод возвращает View(суперкласс Button), поэтому мы почти всегда приводим результат к типу . Подробнее о классе см. в [*разделе Ресурсы и макеты*](https://info448.github.io/resources-and-layouts.html#resources-and-layouts)View .

Затем мы можем зарегистрировать прослушиватель (обратный вызов), вызвав setOnClickListener()метод и передав **анонимный класс** , который будет действовать как прослушиватель:

button.setOnClickListener(**new** View.OnClickListener() {

 **public** void onClick(View v) {

 *// Perform action on click*

 }

});

**Автодополнение Tab** — ваш друг!! Попробуйте просто ввести button., а затем выбрать имя метода из предоставленного списка. Аналогично, вы можете начать вводить new OnClickи затем дополнить Tab-комплетом оставшуюся часть определения класса. Android Studio IDE упрощает создание этого вездесущего шаблонного кода.

Наконец, мы можем заполнить метод, чтобы он выходил из системы или выводил сообщение при нажатии!

## 1.6 Котлин

Как упоминалось выше, Android Studio теперь предоставляет возможность писать «логику» для ваших приложений на [**Kotlin**](https://kotlinlang.org/) , а не на Java. Kotlin будет обсуждаться более подробно в следующей лекции, но этот раздел даст вам быстрое представление о том, как использовать Kotlin в приложении Android.

Есть два способа включить Kotlin в ваше Android-приложение:

1. Вы можете использовать Kotlin при первом создании проекта в Android Studio, нажав на опцию «Включить поддержку Kotlin». Это приведет к тому, что ваша начальная активность станет файлом Kotlin (например, MainActivity.kt) вместо файла Java.
2. Также можно использовать Android Studio для преобразования файла Java в файл Kotlin! Для этого можно использовать команду Code > Convert Java File to Kotlin Fileиз главного меню или вызвать [действие Find](https://www.jetbrains.com/idea/help/navigating-to-action.html) с помощью shift-cmd-a. Это заставит Android Studio выполнить рефакторинг вашего кода Java в код Kotlin, заменив весь синтаксис Java на синтаксис Kotlin.

Вы заметите, что большая часть кода выглядит так же — некоторые знаки препинания сместились, и несколько слов отсутствуют, но в остальном это та же самая настройка. Конкретные различия и детали языка обсуждаются в лекции [*Kotlin*](https://info448.github.io/introduction.html#kotlin-setup) .

1. <http://developer.android.com/reference/android/util/Log.html>[↩](https://info448.github.io/introduction.html#fnref1)