# Лекция №2
# Создание приложений на платформе Android

**Android** — операционная система для смартфонов, планшетов, электронных книг, цифровых проигрывателей, наручных часов, фитнесс-браслетов, игровых приставок, ноутбуков, Очков Google Glass, телевизоров и других устройств (бытовых роботов) (Википедия).

Основана на ядре Linux и собственной реализации виртуальной машины Java от Google. Изначально разрабатывалась компанией Android, Inc., которую затем купила Google.

### *Версии ОС*

Изначально Google рассчитывала давать версиям *Android* имена известных роботов, но потом отказалась из-за проблем с авторскими правами. Каждая версия системы начиная с версии 1.5 получает собственное кодовое имя на тему сладостей. Кодовые имена присваиваются в алфавитном порядке латинского алфавита.

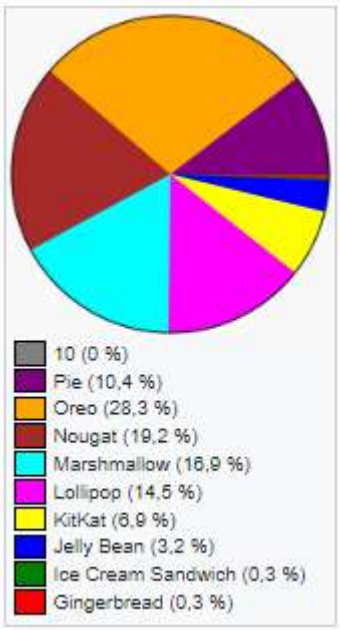
**Статистика распространения версий на 7 мая 2019 года[2]:**

| Версия | Название | Год | Доля |
|---|---|---|---|
| 2.3 | *Gingerbread* | 2010 | 0,3 % |
| 4.0 | *Ice Cream Sandwich* | 2011 | 0,3 % |
| 4.1 | *Jelly Bean* | 2012 | 1,2 % |
| 4.2 | | 2012 | 1,5 % |
| 4.3 | | 2013 | 0,5 % |
| 4.4 | *KitKat* | 2013 | 6,9 % |
| 5.0 | *Lollipop* | 2014 | 3 % |
| 5.1 | | 2015 | 11,5 % |
| 6.0 | *Marshmallow* | 2015 | 16,9 % |
| 7.0 | *Nougat* | 2016 | 11,4 % |
| 7.1 | | 2016 | 7,8 % |
| 8.0 | *Oreo* | 2017 | 12,9 % |
| 8.1 | | 2017 | 15,4 % |
| 9.0 | *Pie* | 2018 | 10,4 % |
| 10.0 | | 2019 | < 0,1 % |

*Примечание: Учитываются только те устройства, владельцы которых запускали Google Play на своём аппарате хотя бы раз за период сбора информации. Версии, имеющие менее 0,1 %, не учитываются статистикой.*

Последняя версия ОС Android на данный момент — Android 10, выпущенная 3 сентября 2019 года.

На настоящий момент выпущено 15 версий системы. Исходя из статистики на май 2019 года, доли версий распределились следующим образом:




***Среды разработки***

Нужно скачать и установить соответствующее SDK, называемое еще JDK (если, конечно, оно уже не установлено).

Сейчас существует несколько сред разработки **:**

**NetBeans** — не самая популярная среда разработки. Разрабатывается компанией Oracle.

**Eclipse** — менее удобная, чем Intellij IDEA, но зато менее прожорлива.

**Intellij IDEA** — также одна из самых популярных IDE, конкурирует с Eclipse. Более удобная и умная. На слабых машинах разработка в ней практически нереальна.

**Android Studio** — ответвление от Intellij IDEA, разрабатываемое компанией Google. Данная IDE ориентирована именно под Android, в отличие от Intellij IDEA, которая также может использоваться для разработки серверных приложений. Также Android Studio не требует установки дополнительных плагинов.

***Языки разработки***

*Java*

Официальный язык программирования, поддерживаемый средой разработки Android Studio. В 2018 году Java вошёл в пятёрку самых популярных языков программирования.

На Java ссылается большинство официальной документации Google.

*Kotlin*

Язык был официально представлен в мае 2017 года на Google I/O и позиционируется Google как второй официальный язык программирования под Android после Java, только чуть более простой для понимания. Знания Java необходимы здесь, чтобы понимать принципы работы Kotlin, общую структуру языка и его особенности.

В настоящее время Kotlin используется для создания Android-приложений такими лидерами бизнеса как Pivotal, Atlassian, Pinterest, Evernote и Uber.

Последняя статистика App brain показывает, что в сегменте топовых приложений 2018 года Kotlin занимает 25,3% рынка. При этом 40,76% новых инсталляций приложений также приходятся на приложения, написанные на Kotlin.

Почему Kotlin это ведущий язык программирования для разработки Android-приложений?

1) Улучшенная производительность

Последовательный и интуитивный синтаксис Kotlin обеспечивает повышение производительности команд разработчиков. Для написания и деплоймента программы требуется меньше времени и меньше строк кода. В результате вы получаете готовое приложение быстрее.

2) 100% совместимости с Java

Другими словами, методы Java можно вызывать из Kotlin. Это является преимуществом не только для разработчиков, но и для компаний, имеющих большую кодовую базу на Java.

3)Легкость поддержки

Android-разработчикам легко поддерживать код на Kotlin, поскольку этот язык поддерживается во многих IDE, включая Android studio, и в нескольких других SDK. Кроме того, разработчики могут работать с любым привычным набором инструментов.

4) Надежность

Последняя версия Kotlin имеет обратную совместимость со всеми предыдущими версиями. Это избавляет Android-разработчиков от огромной головной боли – работы с разными версиями.

5) Легкость изучения

Kotlin изучать легче, чем Java, потому что для этого не требуется никаких знаний в сфере разработки мобильных приложений.

6) Поддержка Android Studio

Android Studio предоставляет расширенную поддержку Kotlin и инструменты адаптации. Разработчики могут работать одновременно на Kotlin и на Java.

*Недостатки Kotlin:*

1) Скорость компиляции

Разработчики жалуются на колебания скорости компиляции кода на Kotlin. В некоторых случаях она происходит очень быстро, а в других заметно медленнее.

2) Меньшая поддержка сообщества

У Kotlin маленькое сообщество разработчиков, в связи с чем количество ресурсов для изучения этого языка ограничено. Вам встретится много вопросов без ответов.

В чем Java все еще лучше, чем Kotlin?

1) Портируемость

С помощью виртуальной машины Java программы на этом языке могут запускаться практически в любой системе. В результате Java получает лидерство среди Android-приложений.

2) Большое сообщество

Благодаря обширным сообществам на GitHub и Stack Overflow разработчики могут получить помощь практически по любой проблеме.

3) Кроссплатформенность

Применение этого языка не ограничивается лишь разработкой Android-приложений. Java прекрасно подходит и для разработки кроссплатформенных приложений.

4) Ресурсы

Поскольку Android разработан на Java, этот язык имеет уже готовые библиотеки и SDK для облегчения процесса разработки.

В чем Java отстает?

1) Скорость

Java требует больше памяти и, по сравнению с другими языками, работает намного медленнее.

2) На Java тяжелее писать код

Код на Java длинный, а следовательно, он требует больше времени для написания, в нем больше ошибок и багов.

Что есть в Java, чего нет в Kotlin

- Статические члены
- Примитивные типы, не являющиеся классами
- Частные поля
- Wildcard-типы
- Отмеченные исключения

Что есть в Kotlin, чего нет в Java

- Шаблоны строк
- Синглтоны
- Null безопасность
- Функции расширения
- Умные приведения типов (smart casts)

*C/C++*

Более низкоуровневые языки поддерживаются Android Studio с использованием Java NDK (Native Development Kit). Это позволяет писать нативные приложения, что может пригодиться для создания игр или других ресурсоёмких программ. Код будет запускаться не через Java Virtual Machine, а непосредственно через девайс, что даст вам больше контроля над такими элементами системы, как память, сенсоры, жесты и т. д., а также возможность выжать из Android-устройств максимум ресурсов. Это также означает, что пользоваться вам придётся библиотеками, написанными на C или C++.

*Python*

Любители этого языка разработали множество инструментов, позволяющих скомпилировать код на Python в требуемое состояние, а наличие различных библиотек позволит строить даже нативные интерфейсы. Самым популярным фреймворком является Kivy.

*BASIC*

Он не поддерживается Android Studio и не подходит для сред Unity и Xamarin. Для BASIC есть специальная среда разработки B4A, в которой можно создавать Android-приложения.

*Lua (с использованием Corona SDK)*

На языке Lua основан кроссплатформенный графический движок Corona. LUA значительно проще Java. Он поддерживает все нативные библиотеки, позволяя тем самым писать под множество платформ.

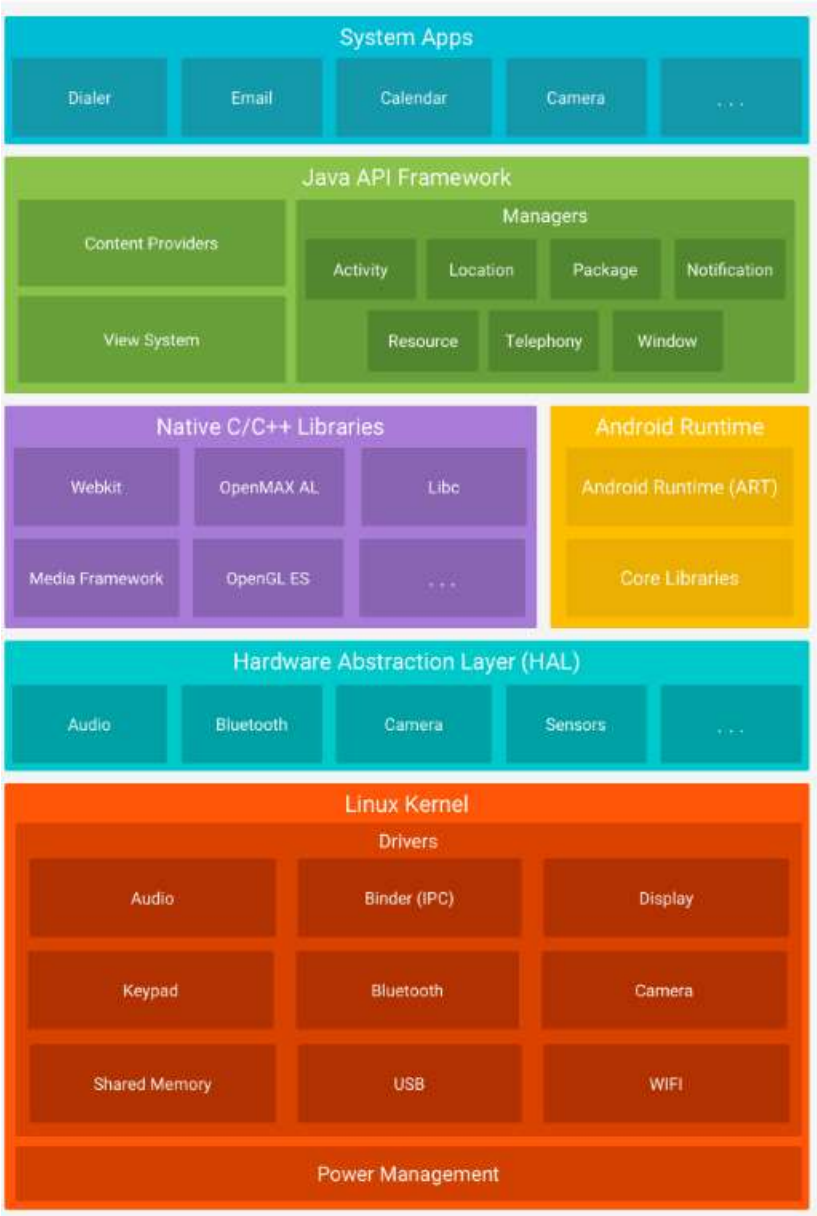
*PhoneGap*

PhoneGap позволяет разрабатывать Android-приложения силами веб-разработки. Ваше приложение будет отображаться через WebView, но как бы в обёртке мобильного приложения. Для разработчиков PhoneGap — это что-то вроде моста для доступа к нативным функциям смартфона или планшета вроде акселерометра или камеры.

***Литература***

http://developer.android.com

## Архитектура ОС Android




### *Ядро Linux*

Основой платформы Android является ядро Linux. Например, Android Runtime (ART) полагается на ядро Linux для базовых функций, таких как потоковая обработка и управление уровнем низкого уровня памяти. Использование ядра Linux позволяет Android использовать ключевые функции безопасности и позволяет производителям устройств разрабатывать аппаратные драйверы для хорошо известного ядра.

### *Уровень абстракции оборудования (HAL)*

Уровень абстракции аппаратного обеспечения (HAL) предоставляет стандартные интерфейсы, которые предоставляют аппаратные возможности устройства для платформы Java API более высокого уровня. HAL состоит из нескольких библиотечных модулей, каждый из которых реализует интерфейс для

определенного типа аппаратного компонента, такого как камера или модуль bluetooth. Когда API-интерфейс инфраструктуры выполняет вызов для доступа к аппаратным средствам устройства, система Android загружает библиотечный модуль для этого аппаратного компонента.

### *Android Runtime*

Для устройств под управлением Android версии 5.0 (API уровня 21) или выше каждое приложение запускается в собственном процессе и с собственным экземпляром Android Runtime (ART). ART записывается для запуска нескольких виртуальных машин на устройствах с низкой памятью путем выполнения DEX-файлов, формат байт-кода, разработанный специально для Android, который оптимизирован для минимального объема памяти.

До версии Android 5.0 (уровень API 21) Dalvik был временем выполнения Android. Если ваше приложение хорошо работает с ART, то оно должно работать и на Дальвике, но обратное может быть неверным. Android также включает в себя набор основных библиотек времени выполнения, которые обеспечивают большую часть функциональных возможностей языка программирования Java, включая некоторые языковые функции Java 8, которые использует инфраструктура Java API.

### *Нативные библиотеки C / C ++*

Многие базовые компоненты и сервисы Android, такие как ART и HAL, построены из собственного кода, для которого требуются собственные библиотеки, написанные на C и C ++. Платформа Android предоставляет API-интерфейсы Java framework, чтобы выявить функциональность некоторых из этих родных библиотек для приложений. Например, вы можете получить доступ к OpenGL ES через Java OpenGL API платформы Android, чтобы добавить поддержку для рисования и управления 2D и 3D-графикой в вашем приложении. Если вы разрабатываете приложение, для которого требуется код C или C ++, вы можете использовать Android NDK для доступа к некоторым из этих родных библиотек плат непосредственно из вашего собственного кода.

### *Java API Framework*

Весь набор функций Android OS доступен вам через API, написанные на языке Java. Эти API-интерфейсы образуют строительные блоки, необходимые для создания приложений для Android, упрощая повторное использование основных, модульных системных компонентов и сервисов, которые включают в себя следующее:

- расширяемая система просмотра, которую вы можете использовать для создания пользовательского интерфейса приложения, включая списки, сетки, текстовые поля, кнопки и даже встраиваемый веб-браузер

- Диспетчер ресурсов, обеспечивающий доступ к ресурсам без кода, таким как локализованные строки, графики и файлы макета
- Диспетчер уведомлений, который позволяет всем приложениям отображать пользовательские предупреждения в строке состояния
- Диспетчер активности, который управляет жизненным циклом приложений и предоставляет общий стек возврата навигации
- Поставщики контента, которые позволяют приложениям получать доступ к данным из других приложений, таких как приложение «Контакты», или предоставлять свои собственные данные .

Разработчики имеют полный доступ к тем же API-интерфейсам платформы, которые используют системные приложения для Android.

Системные приложения Android поставляется с набором основных приложений для электронной почты, SMS-сообщений, календарей, просмотра интернет-страниц, контактов и т. Д. Приложения, входящие в состав платформы, не имеют особого статуса среди приложений, которые пользователь хочет установить. Таким образом, стороннее приложение может стать веб-браузером пользователя по умолчанию, SMS-мессенджером или даже клавиатурой по умолчанию (применяются некоторые исключения, например, приложение «Настройки системы»). Системные приложения функционируют как приложения для пользователей, а также предоставляют ключевые возможности, которые разработчики могут получать из своего собственного приложения. Например, если ваше приложение хочет доставить SMS-сообщение, вам не нужно самостоятельно создавать эту функцию - вместо этого вы можете использовать любое приложение для SMS, которое уже установлено для доставки сообщения получателю, которого вы указываете.

### *Поддерживаемые API-интерфейсы и возможности языка Java*

В настоящее время Android поддерживает не все возможности языка Java. Однако при разработке приложений для Android N Preview доступны следующие функции.

- Статические и заданные по умолчанию методы интерфейсов
- Лямбда-выражения
- Повторяющиеся примечания

Кроме того, доступны следующие API-интерфейсы для реализации возможностей языка Java.

*API-интерфейсы отражения и языковых функций:*

java.lang.FunctionalInterface;

java.lang.annotation.Repeatable;

java.lang.reflect.Method.isDefault();

а также API-интерфейсы отражения, связанные с повторяющимися примечаниями, например,AnnotatedElement.getAnnotationsByType(Class).

*Вспомогательные API:*

java.util.function.

### *Активация возможностей Java и набора инструментов Jack*

Для использования новых возможностей языка Java необходимо также использовать новый набор инструментов Jack. С его помощью Android компилирует языковой источник Java в считываемый Android байткод Dalvik Executable (dex). В Jack предусмотрен собственный формат библиотеки .jack, большинство функциональных возможностей набора инструментов предоставляется в рамках одного инструмента: перекомпоновка, сжатие, обфускация и использование нескольких файлов DEX.

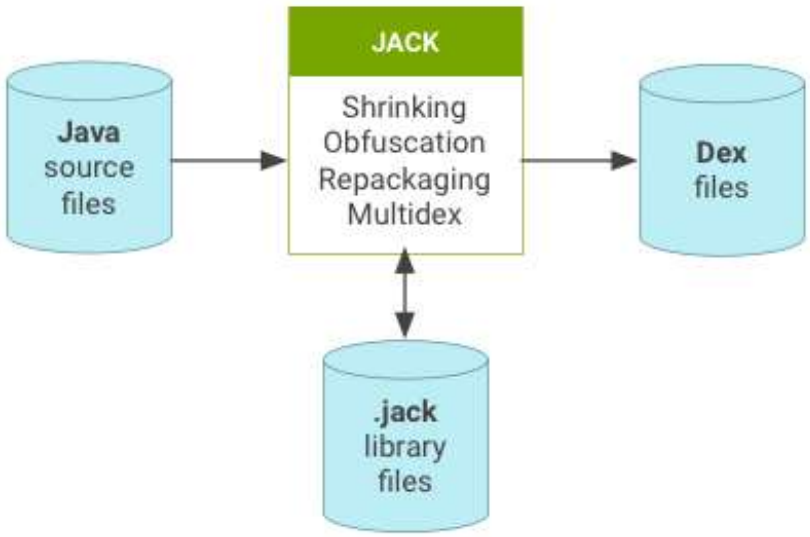
Ниже представлено сравнение двух наборов инструментов, используемых для сборки файлов DEX в Android.

Старый набор инструментов javac:

javac (.java --> .class) --> dx (.class --> .dex)

Новый набор инструментов Jack:

Jack (.java --> .jack --> .dex)




### *Процесс сборки Android-приложения*

Для начала о самом процессе сборки apk.

Когда вы запускаете сборку, первым делом читается AndroidManifest.xml, в нём есть важные параметры, такие как package (например, com.example.app) и targetSdkVersion.

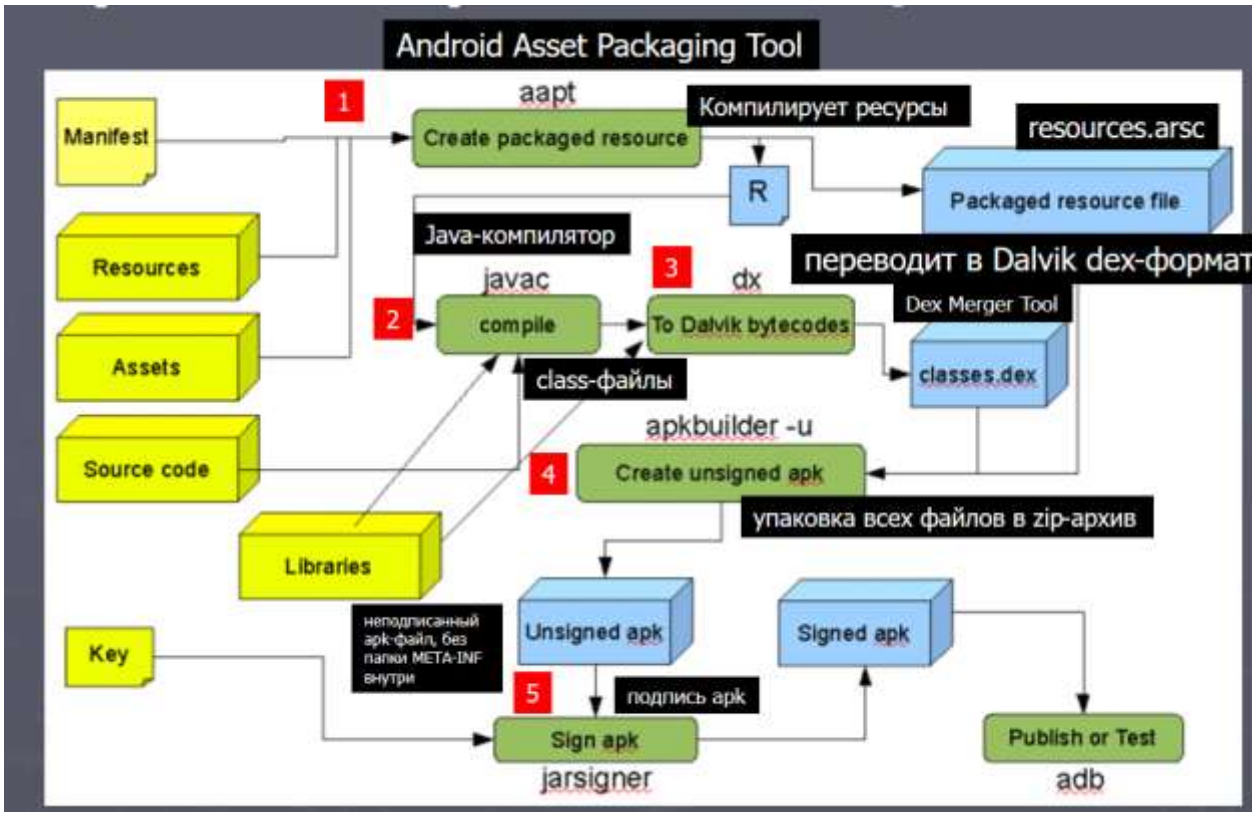
Затем вызывается программа aapt (Android Asset Packaging Tool), которой передаётся AndroidManifest.xml, папка с ресурсами res/, assets/, путь к android.jar нужной target-версии. aapt проверяет ресурсы и компилирует их, создавая при этом класс R.java в котором содержатся идентификаторы ресурсов и файл resources.arsc в котором содержится информация об xml-ресурсах и их атрибутах.

Далее подхватываются все библиотеки, которые используются в проекте и запускается Java-компилятор javac. Полученные class-файлы передаются в

программу dx, которая переводит их в dex-формат. Причём для оптимизации, готовые библиотеки дексируются отдельно, а классы проекта отдельно (оптимизация в том, что дексированные библиотеки можно закешировать). Если собралось несколько dex-файлов, то они все объединяются при помощи Dex Merger Tool. В конечном итоге мы получаем единственный файл classes.dex (или несколько, если используется multidex).

Теперь у нас есть все компоненты и можно собирать apk. По сути это просто упаковка всех файлов в zip-архив, но используется специальная программа apkbuilder. После её выполнения получаем неподписанный apk-файл, то есть без папки META-INF внутри.

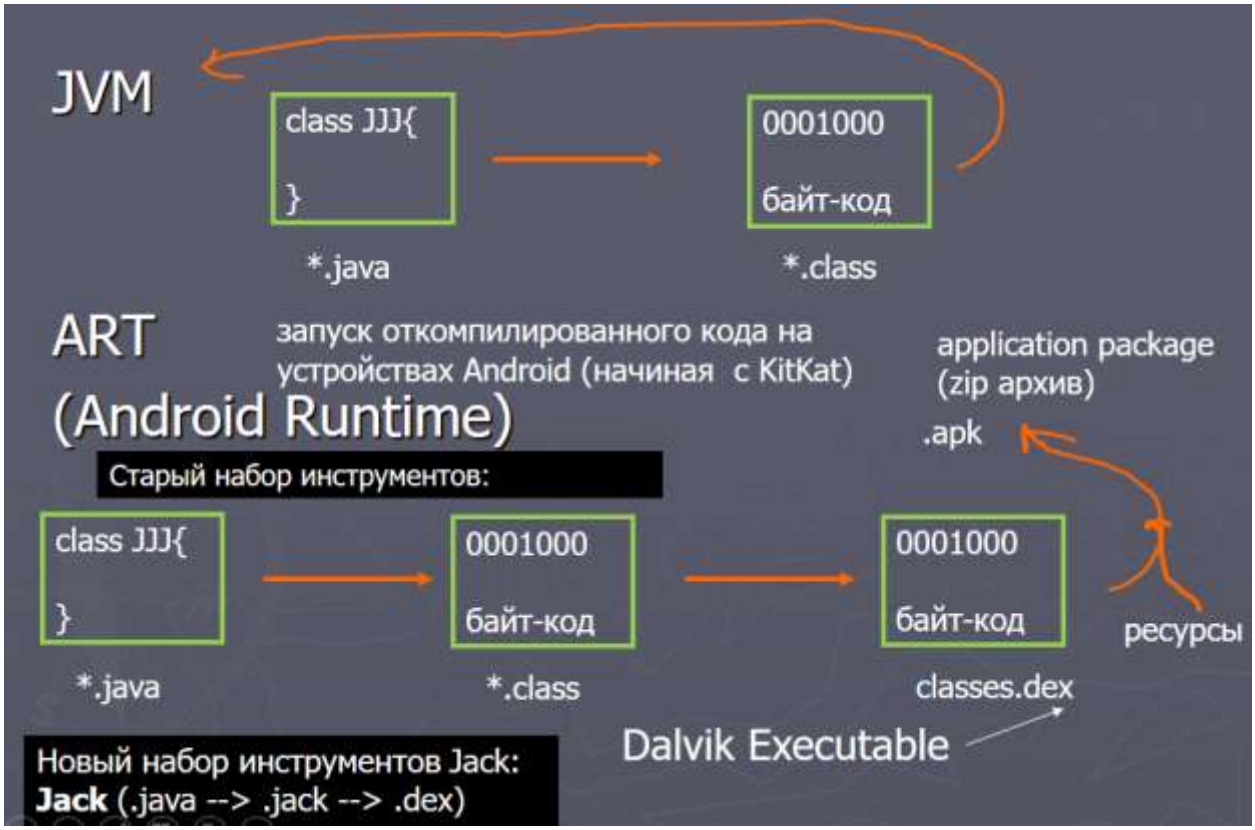
Последний этап - подпись apk. Берётся заранее сгенерированный ключ и передаётся в jarsigner вместе с неподписанным apk-файлом. На выходе имеем готовое приложение, которое можно устанавливать.




***Виртуальная машина Android RunTime***

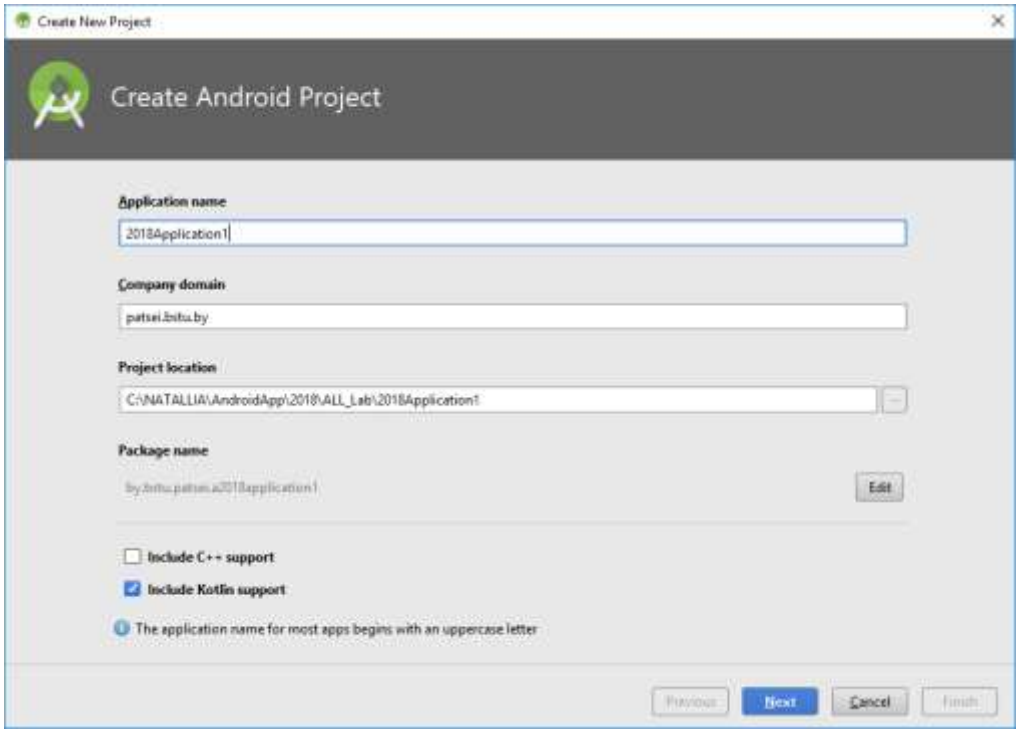
Начиная с версии Android 4.4. KitKat в ОС Android была добавлена вторая виртуальная машина - Android RunTime. Хотя в той же версии Android 4.4 Dalvik по умолчанию является предпочтительной средой для выполнения кода приложения, но уже в версии Android 5.0 Lollipop виртуальная машина Android RunTime полностью заменила Dalvik.

Работа Android RunTime коренным образом отличается от Dalvik: Android RunTime использует предварительную компиляцию приложения при его установке на мобильное устройство, а не JIT-компиляцию, что увеличивает производительность приложения.

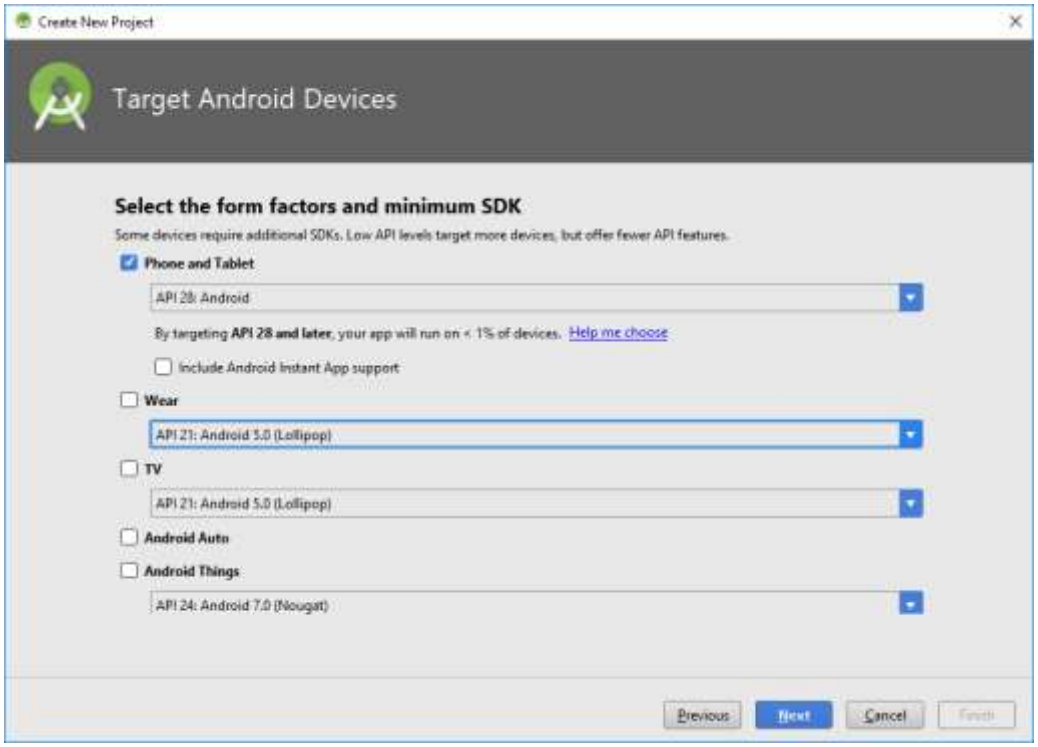



***Создание Android проекта***

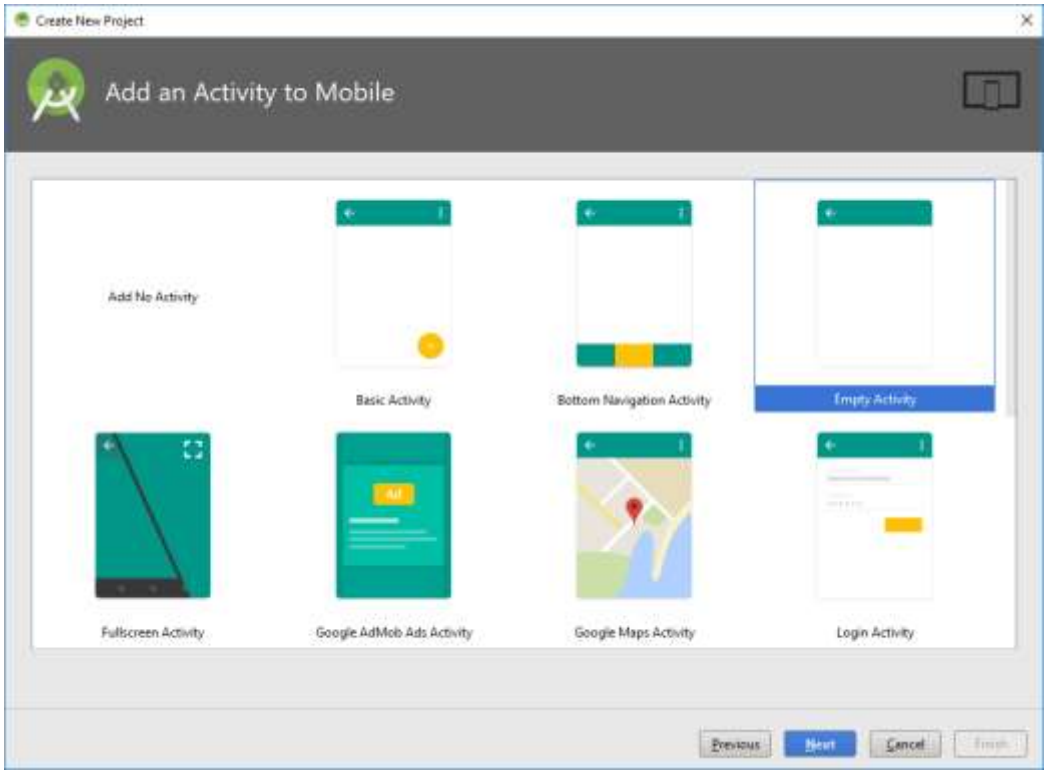
Создаем новый проект.



Как видите нужно установить версию желаемого при разработке API. Есть проблема. При выборе более нового API получаем не большой процент поддерживаемых устройств.




Следующий шаг – шаблон приложения



***Компоненты приложения***

Компоненты приложения являются кирпичиками, из которых состоит приложение для Android. Каждый компонент представляет собой отдельную точку, через которую система может войти в приложение. Не все компоненты являются точками входа для пользователя, а некоторые из них зависят друг от друга. При этом каждый компонент является самостоятельной структурной единицей и играет определенную роль — каждый из них представляет собой уникальный элемент структуры, который определяет работу приложения в целом.

Компоненты приложения можно отнести к одному из четырех типов. Компоненты каждого типа предназначены для определенной цели, они имеют собственный жизненный цикл, который определяет способ создания и прекращения существования компонента.

**Операция** (Activity, активность) представляет собой один экран с пользовательским интерфейсом. Например, в приложении для работы с электронной почтой одна активность может служить для отображения списка новых сообщений, другая – для составления сообщения и третья операция – для чтения сообщений. Несмотря на то, что активности совместно формируют взаимодействие пользователя с приложением, каждая из них не зависит от других. Любые из активностей могут быть запущены другим приложением. Например, приложение для камеры может запустить операцию в приложении по работе с электронной почтой, которая составляет новое сообщение, чтобы пользователь мог отослать фотографию. Операция относится к подклассу Activity [2].

**Служба** (Service) представляет собой компонент, который работает в фоновом режиме и выполняет длительные операции, связанные с работой удаленных процессов [2]. Служба не имеет пользовательского интерфейса. Например, она может воспроизводить музыку в фоновом режиме, пока пользователь работает в другом приложении, или может получать данные по сети, не блокируя взаимодействие пользователя с активностью. Служба может быть запущена другим компонентом, который затем будет взаимодействовать с ней. Служба относится к подклассу *Service*.

**Поставщик контента** (Content provider*)* управляет общим набором данных приложения. Данные можно хранить в файловой системе, базе данных SQLite, в Интернете или любом другом месте хранения, к которому у приложения есть доступ. Посредством поставщика контента другие приложения могут запрашивать или изменять данные (если поставщик контента позволяет делать это). Например, в системе Android есть поставщик контента, который управляет информацией контактов пользователя. Любое приложение, получившее соответствующие разрешения, может запросить часть этого поставщика контента для чтения и записи. Поставщики контента также используются для чтения и записи данных, доступ к которым внешним компонентам приложение не предоставляет. Он относится к подклассу *ContentProvider* и должен реализовывать стандартный набор API-интерфейсов [2].

**Приемник широковещательных сообщений** (Broadcast receiver) представляет собой компонент, который реагирует на объявления,

распространяемые по всей системе. Многие из этих объявлений рассылает система – например уведомление о том, что экран выключился, аккумулятор разряжен или был сделан фотоснимок. Объявления также могут рассылаться приложениями. Например, можно сообщить другим приложениям о том, что какие-то данные были загружены на устройство и теперь готовы для использования. Несмотря на то что приемники широковещательных сообщений не имеют пользовательского интерфейса, они могут создавать уведомления в строке состояния, чтобы предупредить пользователя о событии «рассылка объявления» [2]. Чаще всего они являются просто шлюзом для других компонентов и предназначены для выполнения минимального объема работы. Приемник широковещательных сообщений относится к подклассу *BroadcastReceiver*.

Уникальной особенностью системы Android является то, что любое приложение может запустить компонент другого приложения. Для пользователя это будет выглядеть как одно приложение. Когда система запускает компонент, она запускает процесс для этого приложения (если он еще не был запущен) и создает экземпляры классов, которые требуются этому компоненту. Поэтому, в отличие от приложений для большинства других систем, в приложениях для Android отсутствует единая точка входа (в них нет функции *main*()).

Поскольку система выполняет каждое приложение в отдельном процессе с такими правами доступа к файлам, которые ограничивают доступ в другие приложения, приложение не может напрямую вызвать компонент из другого приложения. Но это может сделать система Android. Поэтому, чтобы вызвать компонент в другом приложении, необходимо сообщить системе о своем **намерении** (*Intent*) запустить определенный компонент. После этого система активирует для вас этот компонент.

***Структура проекта***


Здесь также мы увидим единственный модуль проекта - модуль app. Собственно весь код, с которым мы будем работать, располагается внутри этого модуля.

Все модули в проекте описываются файлом *setting.gradle*. По умолчанию он имеет следующее содержимое

```
include ':app'
```

Файл *build.gradle* содержит информацию, которая используется при построении проекта.

Каждый модуль имеет свой файл build.gradle, который определяет конфигурацию построения проекта, специфичную для данного модуля. Так, если мы посмотрим на содержимое папки app, то как раз найдем в ней такой файл. На начальном этапе данные файлы не столь важны, достаточно лишь понимать, для чего они нужны.

```
apply plugin: 'com.android.application'

apply plugin: 'kotlin-android'

apply plugin: 'kotlin-android-extensions'

android {
    compileSdkVersion 26
    defaultConfig {
        applicationId "by.bstu.patsei.hellow"
```

```
        minSdkVersion 28
        targetSdkVersion 26
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
'proguard-rules.pro'
        }
    }
}

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation"org.jetbrains.kotlin:kotlin-stdlib-jre7:$kotlin_version"
    implementation 'com.android.support:appcompat-v7:26.1.0'
    implementation 'com.android.support.constraint:constraint-layout:1.1.3'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'com.android.support.test:runner:1.0.2'
    androidTestImplementation 'com.android.support.test.espresso:espresso-
core:3.0.2'
}
```

В модуле *app* мы можем увидеть несколько папок и файлов:

каталог src - предназначен для хранения исходного кода. Он содержит ряд подкаталогов. Каталоги androidTest и test предназначены для хранения файлов тестов приложения. А собственно исходные коды располагаются в папке java.

```
package by.bstu.patsei.hellow

import android.support.v7.app.AppCompatActivity
import android.os.Bundle

class HelloWActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_hello_w)
    }
}
```

*Файл манифеста*

AndroidManifest.xml представляет файл манифеста, который описывает фундаментальные характеристики приложения, его конфигурацию и определяет каждый из компонентов данного приложения.

Для запуска компонента приложения системе Android необходимо знать, что компонент существует. Для этого она читает файл AndroidManifest.xml приложения (файл манифеста). В этом файле, который должен находиться в корневой папке приложения, должны быть объявлены все компоненты приложения.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="by.bstu.patsei.hellow">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".HelloWActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Объявляет имя Java-пакета данного приложения (строка 3). Имя пакета служит уникальным идентификатором приложения.

Атрибут android:icon в элементе <application> указывает на ресурсы для значка, который обозначает приложение.

Атрибут android:name в элементе <activity> указывает полное имя класса подкласса Activity, а атрибут android:label указывает строку, которую необходимо использовать в качестве метки операции, отображаемой для пользователя.

Описывает компоненты приложения – Activity,Service, Broadcast Receiver и Content Provider, из которых состоит данное приложение. Эти объявления позволяют Андроид знать чем компоненты являются и при каких условиях они могут быть запущены. Все компоненты приложения необходимо объявлять следующим образом:

элементы <activity> для операций;

элементы <service> для служб;

элементы <receiver> для приемников широковещательных сообщений;

элементы <provider> для поставщиков контента

Системе не видны операции, службы и поставщики контента, которые имеются в исходном коде, но не объявлены в манифесте, поэтому они не могут быть запущены.

Помимо объявления компонентов приложения, манифест служит и для других целей, среди которых:

- указание всех полномочий пользователя, которые требуются приложению, например разрешения на доступ в Интернет или на чтение контактов пользователя;
- объявление минимального уровня API, требуемого приложению, с учетом того, какие API-интерфейсы оно использует;

- объявление аппаратных и программных функций, которые нужны приложению или используются им, например камеры, службы Bluetooth или сенсорного экрана;
- указание библиотек API, с которыми необходимо связать приложение (отличные от API-интерфейсов платформы Android), например библиотеки Google Maps ;
- и многое другое

И все же основная задача манифеста – это информировать систему о компонентах приложения.

Подробные сведения о структуризации файла манифеста для приложения см. в документе Файл AndroidManifest.xml .

*Ресурсы приложения*

Папка res содержит каталоги с ресурсами, в частности она содержит следующие каталоги:

- папка drawable предназначена для хранения изображений, используемых в приложении
- папка layout предназначена для хранения файлов, определяющих графический интерфейс. По умолчанию здесь есть файл activity_main.xml, который определяет интерфейс для единственной в проекте activity - MainActivity
- папки mipmap-xxxx содержат файлы изображений, которые предназначены для создания иконки приложения при различных разрешениях экрана. Соответственно для каждого вида разрешения здесь имеется свой каталог
- папка values хранит различные xml-файлы, содержащие коллекции ресурсов - различных данных, которые применяются в приложении

Используя ресурсы приложения, можно без труда изменять его различные характеристики, не меняя код, а, кроме того, —путем предоставления наборов альтернативных ресурсов — можно оптимизировать свое приложение для работы с различными конфигурациями устройств (например, для различных языков или размеров экрана).

Для каждого ресурса, включаемого в проект Android, инструменты SDK задают уникальный целочисленный идентификатор, который может использоваться, чтобы сослаться на ресурс из кода приложения или из других ресурсов, определенных в XML. Например, если в вашем приложении имеется файл изображения с именем logo.png (сохраненный в папке res/drawable/), инструменты SDK сформируют идентификатор ресурса под именем R.drawable.logo, с помощью которого на изображение можно будет ссылаться и вставлять его в пользовательский интерфейс.

Один из наиболее важных аспектов предоставления ресурсов отдельно от исходного кода заключается в возможности использовать альтернативные ресурсы для различных конфигураций устройств. Например, определив строки пользовательского интерфейса в XML, вы сможете перевести их на другие языки и сохранить эти переводы в отдельных файлах. Затем по квалификатору языка ,

добавленному к имени каталога ресурса (скажем res/values-fr/ для строк на французском языке), и выбранному пользователем языку система Android применит к вашему пользовательскому интерфейсу строки на соответствующем языке.

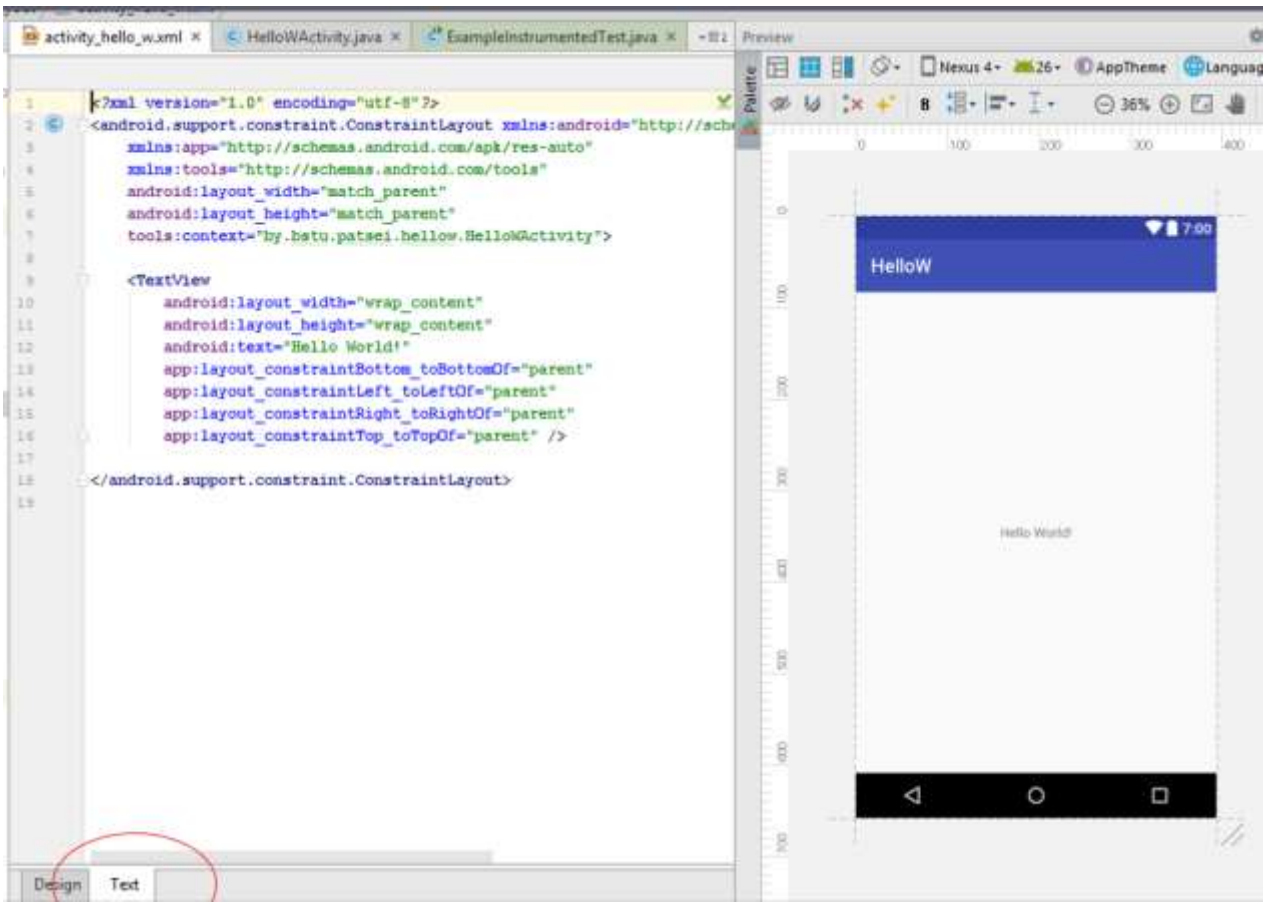
Android поддерживает разные квалификаторы для соответствующих ресурсов. Квалификатор представляет собой короткую строку, которая включается в имена каталогов ресурсов с целью определения конфигурации устройства, для которой эти ресурсы следует использовать. В качестве другого примера можно сказать, что для своих операций следует создавать разные макеты, которые будут соответствовать размеру и ориентации экрана устройства. Например, когда экран устройства имеет книжную ориентацию (расположен вертикально), кнопки в макете можно также размещать по вертикали, а когда экран развернут горизонтально (альбомная ориентация), кнопки следует размещать по горизонтали. Чтобы при изменении ориентации экрана изменялся макет, можно определить два разных макета и применить соответствующий квалификатор к имени каталога каждого макета. После этого система будет автоматически применять соответствующий макет в зависимости от ориентации устройства.

Подробные сведения о различных видах ресурсов, которые можно включить в приложение, а также о том, как создавать альтернативные ресурсы для разных конфигурацией устройств, см. в разделе Предоставление ресурсов.

### *Разработка интерфейса приложения*

В студии должен быть открыт файл *activity_hello_w.xml* (по умолчанию - *activity_main.xml*), который содержит определение графического интерфейса приложения.

Если файл открыт в режиме дизайнера, а в центре отображается дизайн приложения, то надо переключить вид файла в текстовый. Для переключения режима из текстового в графический и обратно внизу есть две кнопки *Design* и *Text* (рис. 1.5).

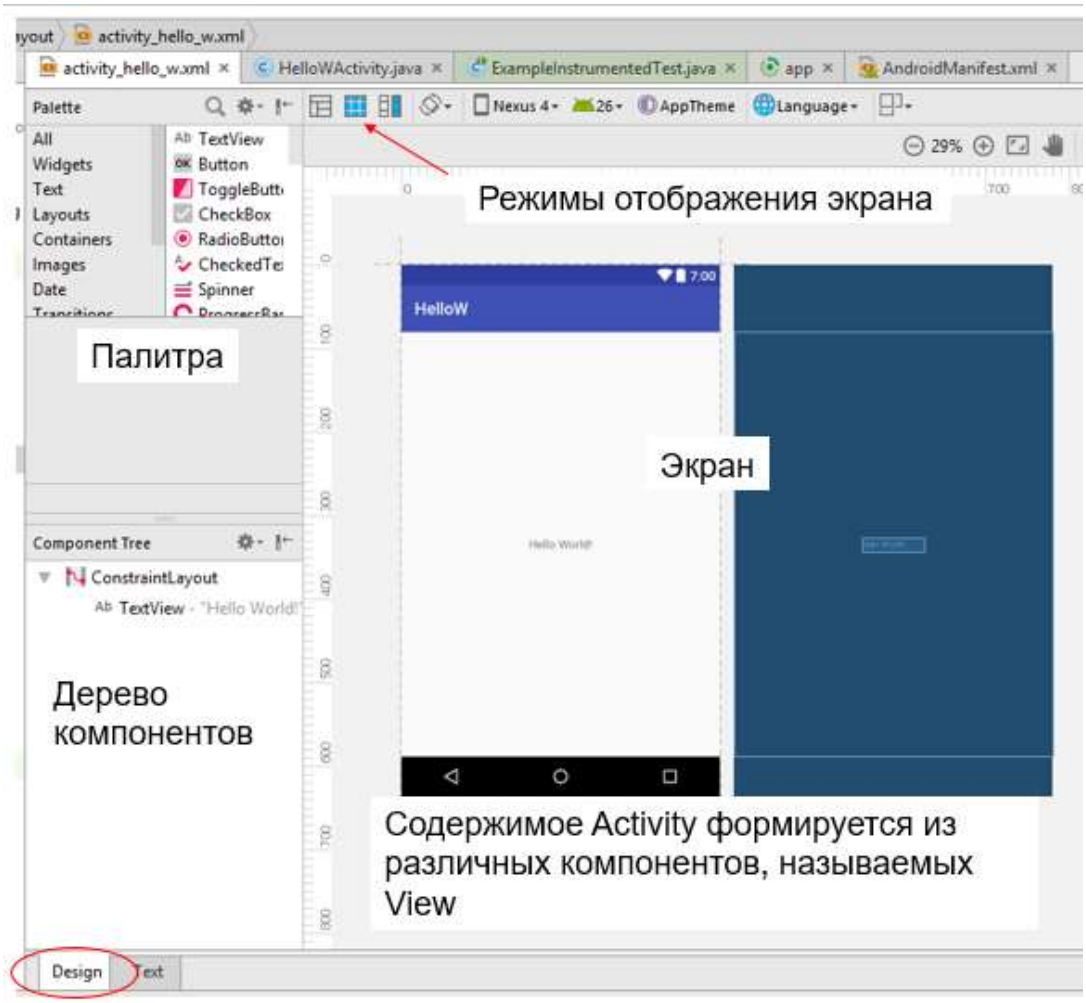



Режимы редактирования Activity

Изменим код приложения так, чтобы оно выводило на экран строку "Hello World!". В файле *activity_hello_w.xml* определение элемента *TextView* отвечает за вывод текстовой информации на экран. Выводимый текст задается с помощью атрибута*android:text*.

После сохранения файла мы можем переключиться в графический режим (рис.).

После переключения режима вы увидите обычный белый и рядом с ним синий вид (рис. далее). Это один и тот же экран, но он отображен в двух разных режимах: *Design* – в нем мы видим *View* компоненты так, как они обычно выглядят на экране; *Blueprint* – отображаются только контуры *View* компонентов.



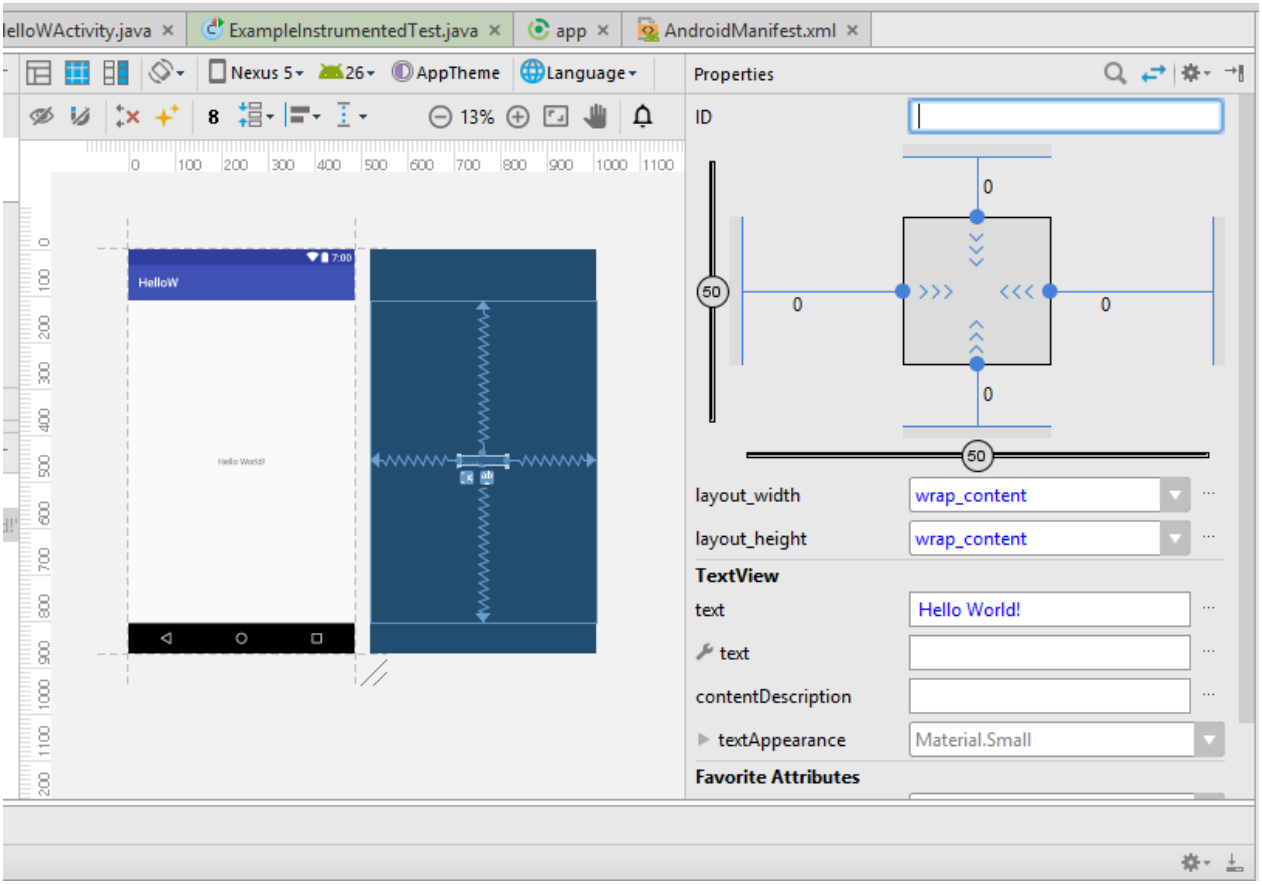

Структура окон разработки и редактирования экрана приложения

Кнопки на панели (рис) позволяют переключать режимы: *Design; Blueprint; Design + Blueprint*.

Окно *Палитра (Palette)* представляет список всех *View* компонентов, которые можно добавлять на экран: кнопки, поля ввода, чекбоксы, прогрессбары и т.д (рис).

Окно *Дерево компонентов (Component Tree)* отображает иерархию *View* компонентов. Например, на рис. корневой элемент – это *ConstraintLayout,* а в него вложен *TextView*.

В окне *Свойства (Properties)* при работе с *View* компонентом будут отображаться его свойства (рис).




Окно свойств

Activity является классом, который по сути представляет отдельный экран (страницу) приложения или его визуальный интерфейс. Приложение может иметь одну activity, а может и несколько. Каждая отдельная активность задает окно для отображения.

Рассмотрим код активности, которая генерируется автоматически в Android Studio (файл кода можно найти в проекте в папке *src/main/java*):

```
import android.support.v7.app.AppCompatActivity
import android.os.Bundle

class HelloWActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_hello_w)
    }
}
```

Класс *HelloWActivity* представляет обычный java класс, в начале которого идут определения пакета и импорта внешних пакетов. По умолчанию он содержит только один метод *onCreate()*, в котором фактически и создается весь интерфейс приложения.

В методе *OnCreate()* идет обращение к методу родительского класса и установка ресурса разметки дизайна:

```
super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_hello_w);
```

Чтобы установить ресурс разметки дизайна, вызывется метод *setContentView*, в который передается идентификатор ресурса. Идентификатор ресурса выглядит следуюшим образом: *R.layout.activity_hello_w*. Это и есть ссылка на файл *activity_hello_w.xml*, который находится в каталоге *res/layout*. Таким образом, при запуске приложения сначала запускается класс *HelloWActivity*, который в качестве графического интерфейса устанавливает разметку из файла *activity_hello_w.xml*. Однако в классе *HelloWActivity* мы используем не файлы, а идентификаторы ресурсов: *R.layout. activity_hello_w*.

Все идентификаторы ресурсов определены в классе *R*, который автоматически создается утилитой *appt* и находится в файле *R.java* в каталоге *build/generated/source/r/debug* (рис.) Класс *R* содержит идентификаторы для всех ресурсов. Для каждого типа ресурсов в классе *R* создается внутренний класс (например, для всех графических ресурсов из каталога *res/drawable* создается класс *R.drawable*) и каждому ресурсу присваивается идентификатор (рис.).

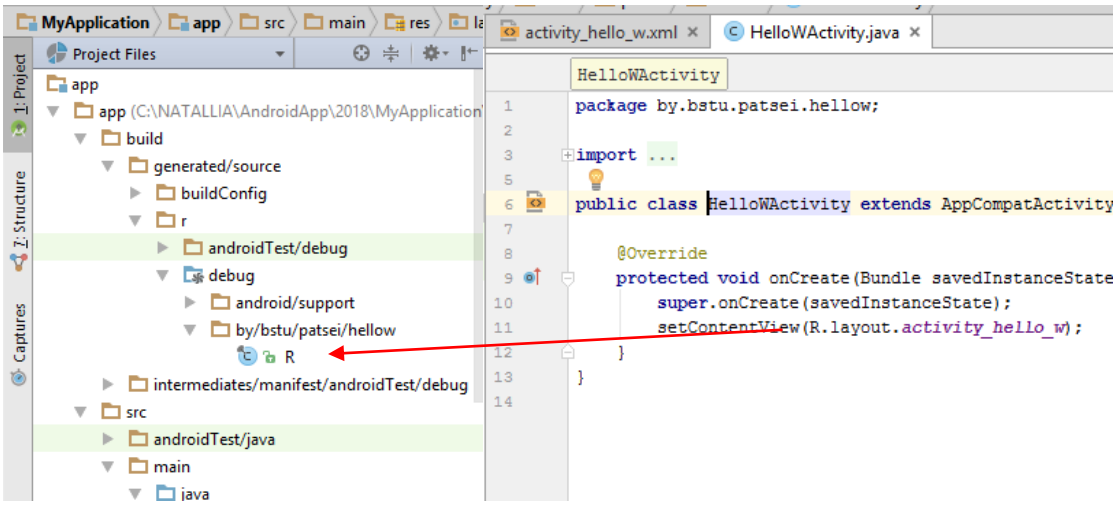



Рис. Обращение и физическое размещение файла ресурсов

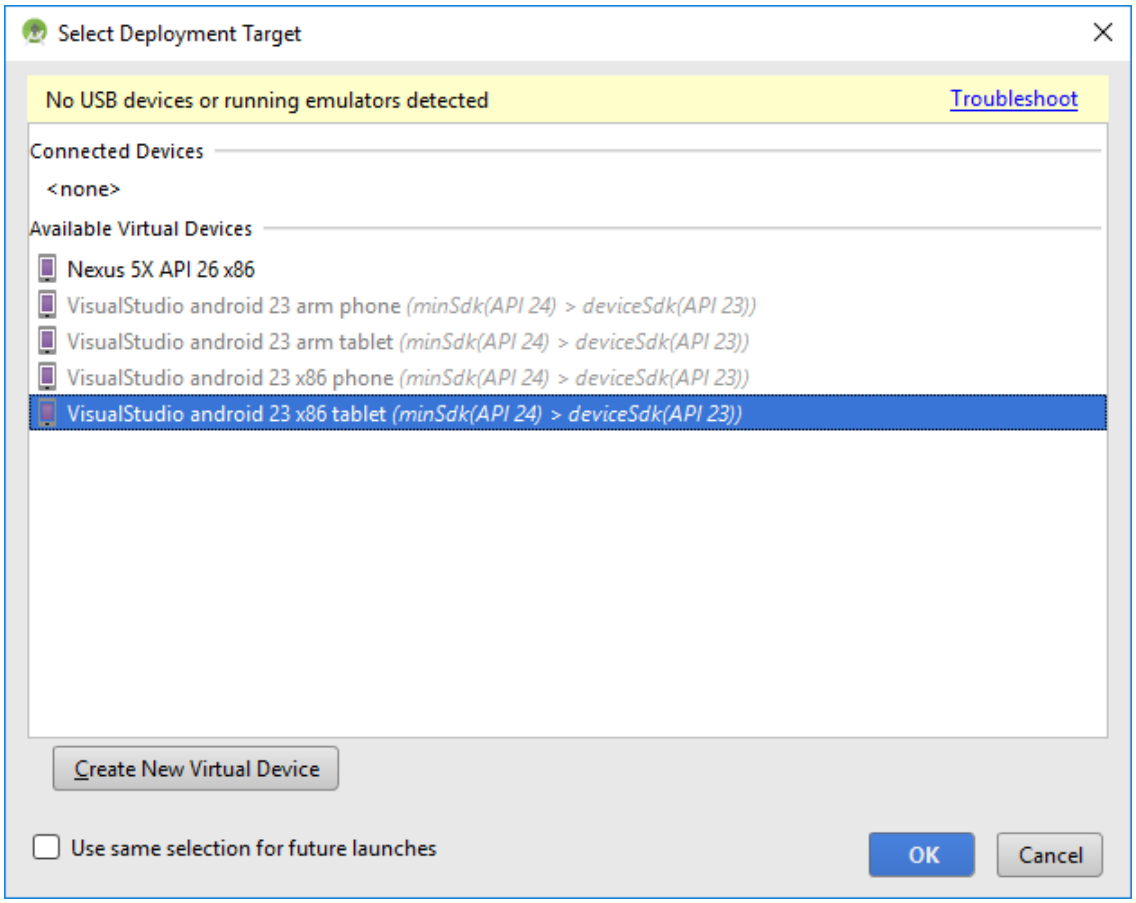
Рис. Структура класса ресурсов *R.java*

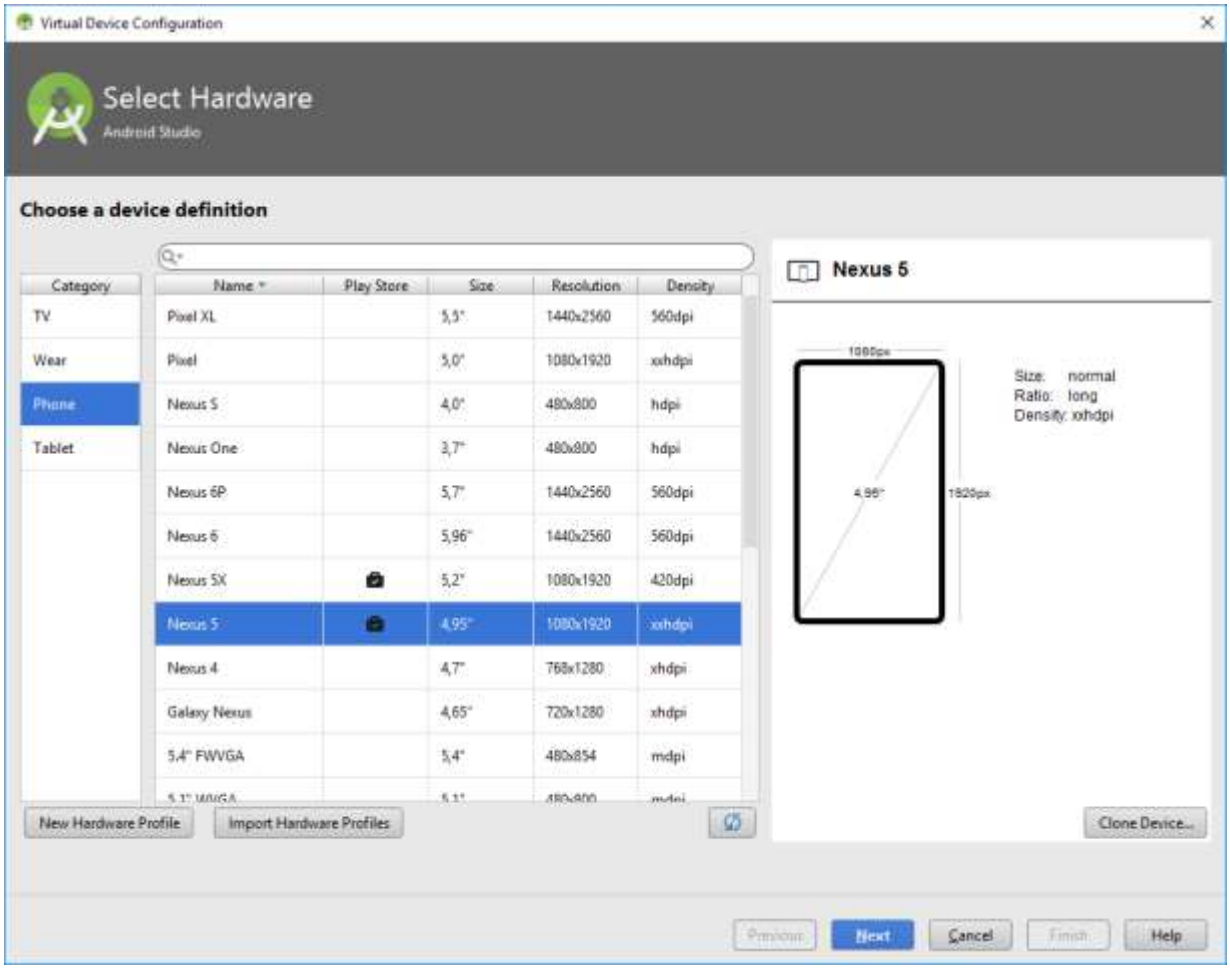
По этому идентификатору в последствии можно извлечь ресурс в файле кода. При обновлении ресурсов во время компиляции этот файл также обновляется.

### *Запуск приложения*

Запускать можно на реальном Android-устройстве или эмуляторе.

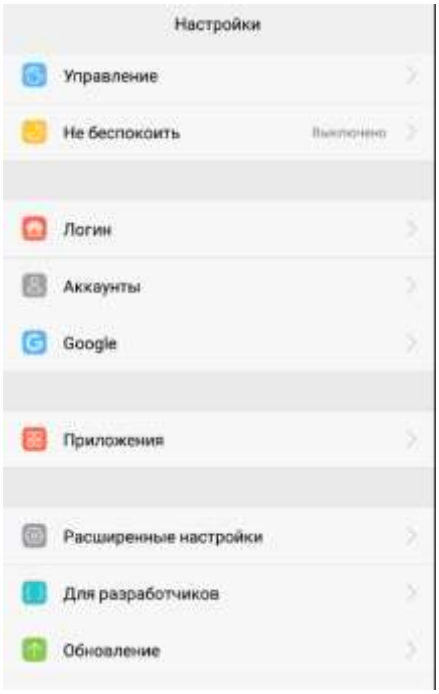
1) Создание эмулятора






2) Использования мобильного устройства для тестирования

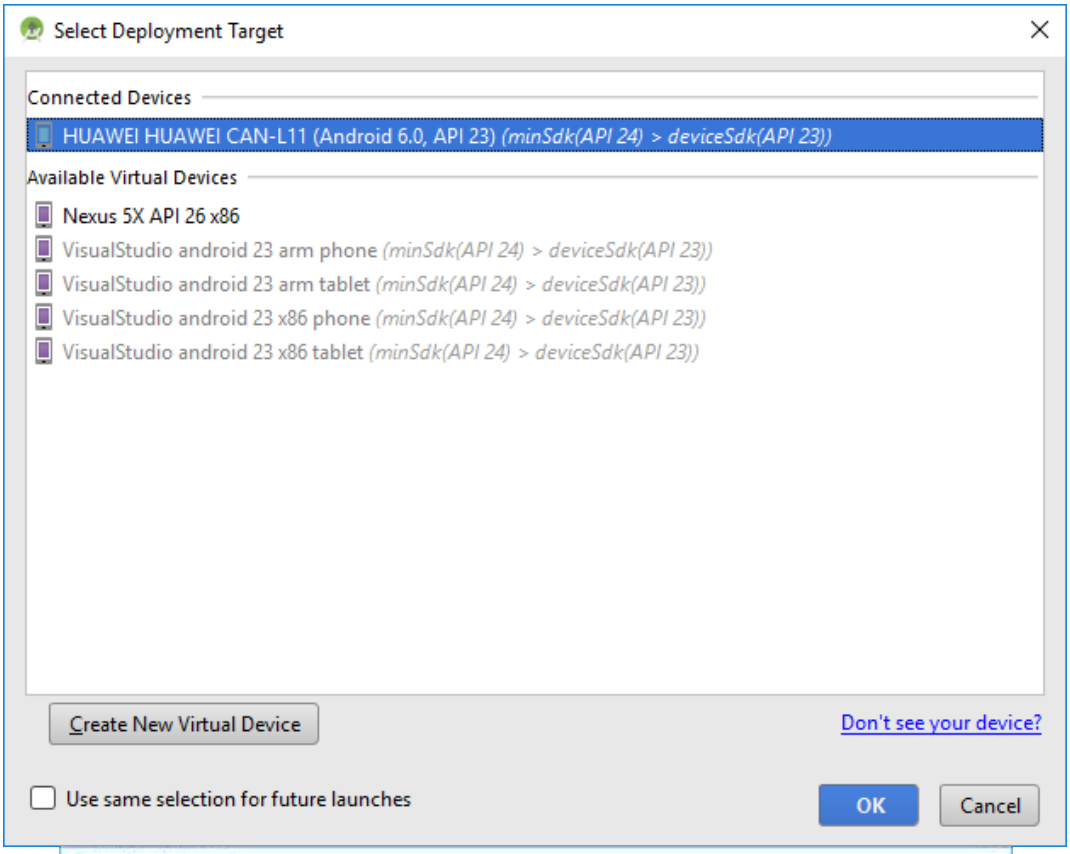
- Установить опции разработчика на МУ
  - **Settings > About phone (Настройки > О телефоне)** и семь раз нажмите **Build Number (Номер сборки)**.
  - Вернитесь к предыдущему экрану и там вы увидите доступный пункт **Developer options (Для разработчика)**.
  - Перейдем к пункту **Для разработчиков** и включим

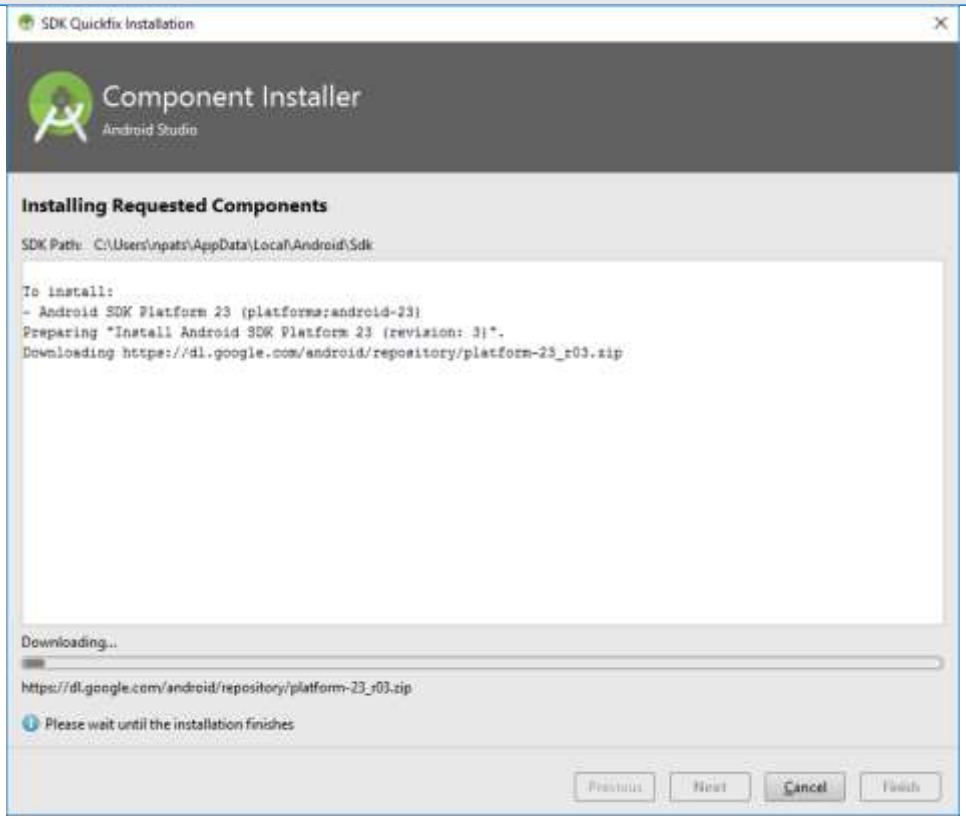



возможность отладки по USB

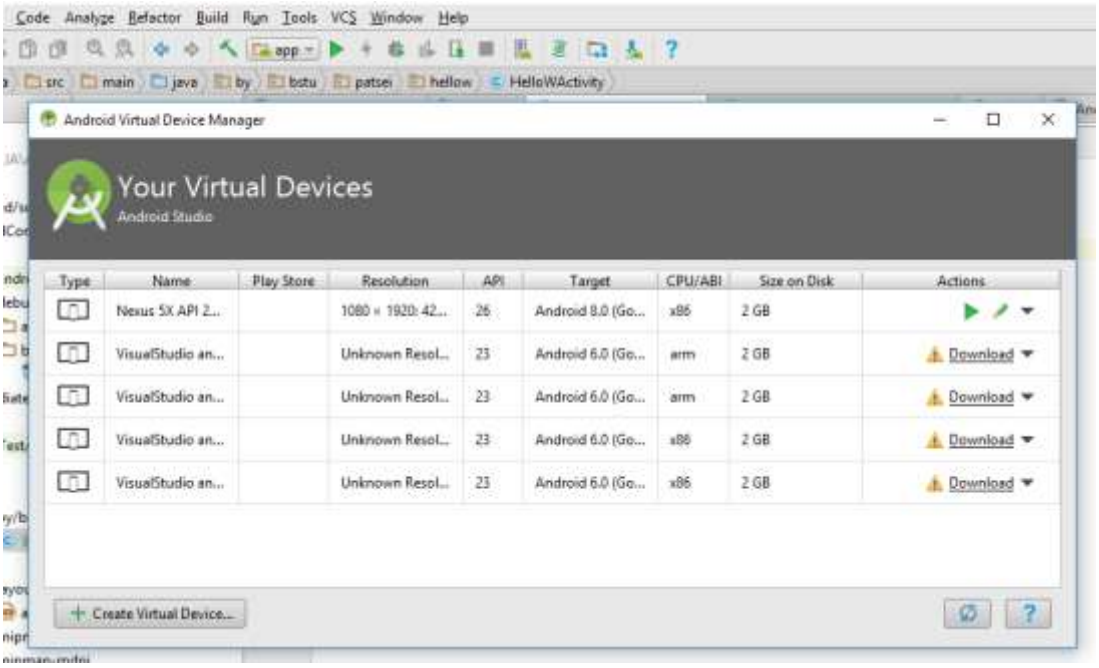
- чрез SDK Manager установить пакет **Google Usb Driver или**

**другой драйвер**


Select Deployment Target
Connected Devices
HUAWEI HUAWEI CAN-L11 (Android 6.0, API 23) (minSdk(API 24) > deviceSdk(API 23))
Available Virtual Devices
Nexus 5X API 26 x86
VisualStudio android 23 arm phone (minSdk(API 24) > deviceSdk(API 23))
VisualStudio android 23 arm tablet (minSdk(API 24) > deviceSdk(API 23))
VisualStudio android 23 x86 phone (minSdk(API 24) > deviceSdk(API 23))
VisualStudio android 23 x86 tablet (minSdk(API 24) > deviceSdk(API 23))
Create New Virtual Device
Don't see your device?
Use same selection for future launches
OK
Cancel


SDK Quickfix Installation
Component Installer
Android Studio
Installing Requested Components
SDK Path: C:\Users\npats\AppData\Local\Android\Sdk
To install:
- Android SDK Platform 23 (platforms;android-23)
Preparing "Install Android SDK Platform 23 (revision: 3)".
Downloading https://dl.google.com/android/repository/platform-23_r03.zip
Downloading...
https://dl.google.com/android/repository/platform-23_r03.zip
Please wait until the installation finishes
Previous
Next
Cancel
Finish

***AVD Manager***




Результат запуска


***Сторонние эмуляторы***
Genymotion
Droid4x
BlueStacks
….