

# Освой программирование игр

## Сайт Александра Климова



*/\* Моя кошка замечательно разбирается в программировании. Стоит мне объяснить проблему ей - и все становится ясно. \*/*

*John Robbins, Debugging Applications, Microsoft Press, 2000*

Главная
Теория
Palette
ListView
Котошоп
Анимация
SQLite
OpenGL ES
Библиотеки
Игры
Wear
Эмулятор
Android Studio
RxJava
Советы
Статьи

Java
Kotlin
Дизайн
Отладка
Open Source
Полезные ресурсы

# Hello Kitty - Создаём первое приложение для Android

После установки Android Studio (далее Студия) можно приступать к созданию своей первой программы.

Учтите, что студия постоянно обновляется, поэтому внешний вид окон и другие детали могут отличаться от данного примера. Большинство уроков на сайте сейчас используют версию 2.3. 25 октября 2017 года вышла версия 3.0, в которой многое поменялось. В этой статье я постарался заменить все картинки под новую версию.

В качестве языка программирования для Android используется Java. Для создания пользовательского интерфейса используется XML.

Здесь следует сделать небольшое отступление. В Android Studio 3.0 добавлена полноценная поддержка нового языка Kotlin, разработанная котанами. Google объявила о своих планах сделать новый "кошачий" язык основным. Но вы должны понимать, что за предыдущие годы было написано огромное количество примеров на Java. Если вы новичок в программировании, то лучше в первый период обучения полностью сосредоточиться на Java, вам будет проще находить ответы на вопросы. Kotlin от вас никуда не денется, перейти потом на него будет проще, а вот обратный процесс будет проходить тяжелее. Когда немного освоитесь в Java, то можете параллельно изучать примеры на Kotlin. Google сейчас активно переписывает документацию под Kotlin, но до полного перехода ещё далеко, даже меньше 50%. Чуть позже я также буду делать уроки под Kotlin, но это будет не скоро.

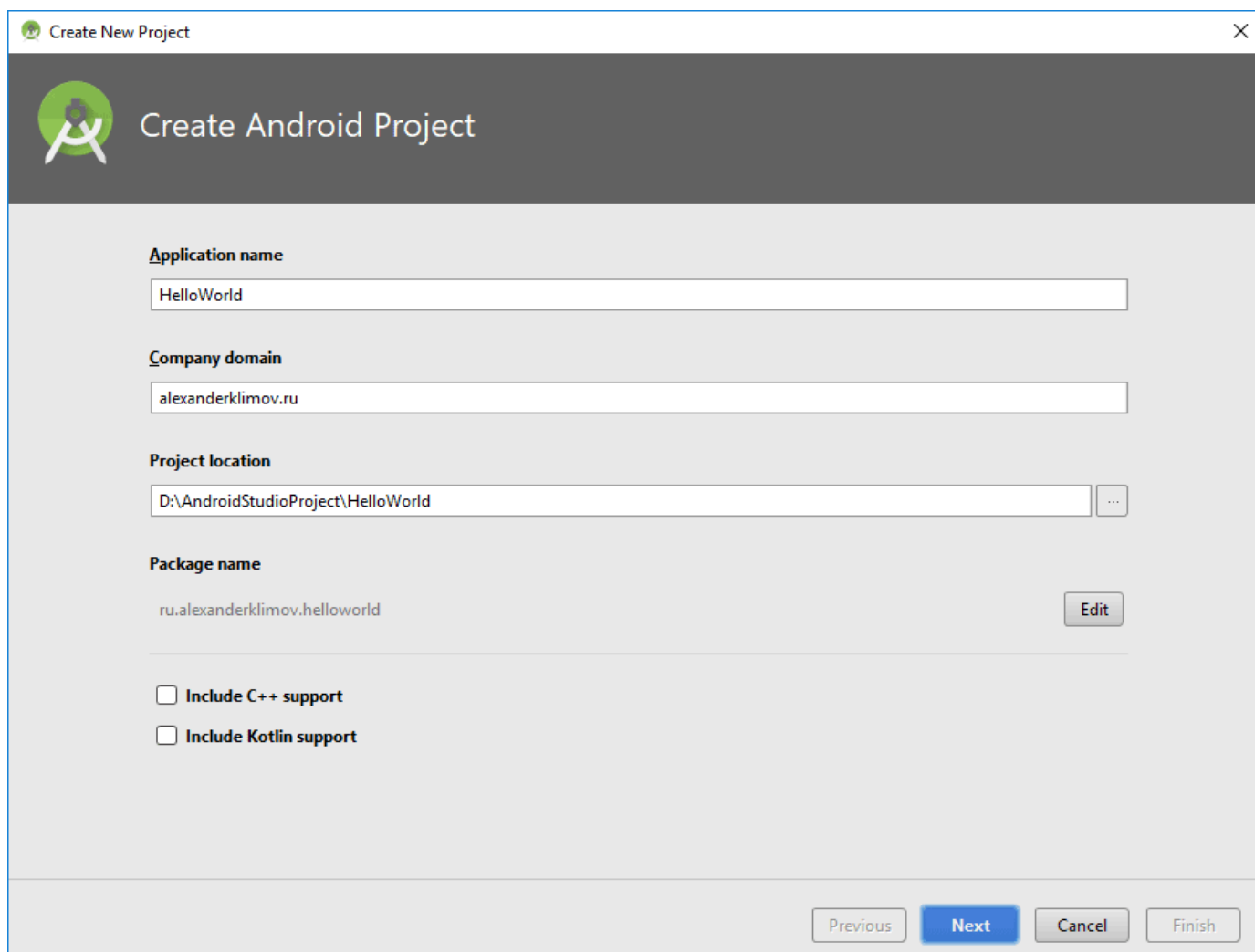
По традиции, заложенной в прошлом веке, каждый программист должен был написать «Hello World!» (Здравствуй, Мир!) в качестве первой программы. Времена меняются, и программа «Hello World!» уже встроена в среду разработки под Android в целях

чем с каким-то миром.

Поэтому разобьём задачу на две части. Сначала запустим готовую программу **Hello World!** без написания кода, чтобы убедиться, что весь инструментарий корректно установлен, и мы можем создавать и отлаживать программы. А потом уже напишем свою первую программу.

## Создание нового проекта

Запускаем Студию и выбираем **File | New | New Project...** Появится диалоговое окно мастера.



Create New Project

Create Android Project

**Application name**  
HelloWorld

**Company domain**  
alexanderklimov.ru

**Project location**  
D:\AndroidStudioProject\HelloWorld

**Package name**  
ru.alexanderklimov.helloworld

☐ Include C++ support

☐ Include Kotlin support

Previous Next Cancel Finish

Поле **Application name** - понятное имя для приложения, которое будет отображаться в заголовке приложения. По умолчанию у вас уже может быть **My Application**. Заменим на **Hello World**. В принципе вы могли написать здесь и **Здравствуй, мир!**, но у Android есть замечательная возможность выводить нужные строки на телефонах с разными языками. Скажем, у американца на телефоне появится надпись на английском, а у русского - на русском. Поэтому в первоначальных настройках всегда используются английские варианты, а локализованные строки подготовите позже. Необходимо сразу вырабатывать привычку к правильному коду.

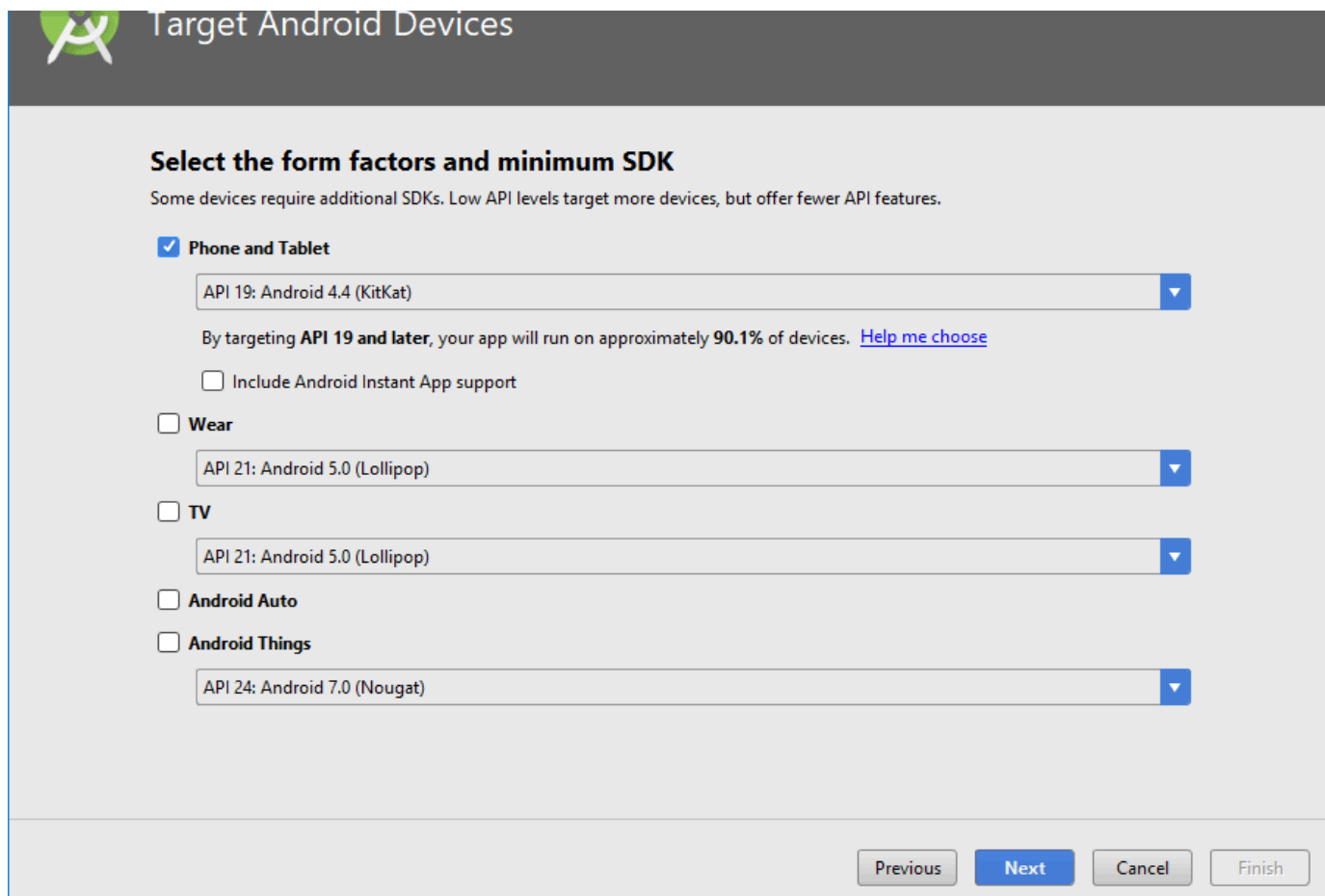
его адрес, либо придумайте какое-нибудь название. Введенное имя запоминается и будет автоматически подставляться в следующих новых проектах. Экономия, однако.

Третье поле **Project location** позволяет выбрать место на диске для создаваемого проекта. Вы можете создать на своём диске отдельную папку для своих проектов и хранить свои программы в ней. Студия запоминает последнюю папку и будет автоматически предлагать сохранение в ней. В случае необходимости вы можете задать другое местоположение для отдельного проекта через кнопку с тремя точками.

Поле **Package name** формирует специальный Java-пакет на основе вашего имени из предыдущего поля. В Java используется перевёрнутый вариант для наименования пакетов, поэтому сначала идёт **ru**, а потом уже название сайта. Пакет служит для уникальной идентификации вашего приложения, когда вы будете его распространять. Если сто человек напишет сто приложений с названием "Cat", то будет непонятно, где приложение, написанное разработчиком Василием Котовым. А приложение с именем пакета **ru.vaskakotov.cat** проще найти. Обратите внимание, что Гугл в своей документации использует пакет **com.example** в демонстрационных целях. Если вы будете просто копировать примеры из документации и в таком виде попытаетесь выложить в Google Play, то у вас ничего не выйдет - это название зарезервировано и запрещено к использованию в магазине приложений. Кнопка **Edit** позволяет отредактировать подготовленный вариант. Например, вы пишете приложение на заказ и вам нужно использовать имя пакета, утверждённое заказчиком, а не ваш вариант по умолчанию.

Ниже представлены два варианта для написания программ на C++ и Kotlin. Эти варианты мы пока не рассматриваем. Когда вы будете писать на Kotlin, то ставьте соответствующий флажок. Впрочем, вы можете конвертировать проект с Java на Kotlin и позже средствами студии.

Нажимаем на кнопку **Next** и переходим к следующему окну. Здесь мы выбираем типы устройств, под которые будем разрабатывать своё приложение. В большинстве случаев мы будем писать для смартфонов и планшетов, поэтому оставляем флажок у первого пункта. Также вы можете писать приложения для Android TV, Android Wear, Android Auto и Android Things.



**Target Android Devices**

**Select the form factors and minimum SDK**

Some devices require additional SDKs. Low API levels target more devices, but offer fewer API features.

☒ **Phone and Tablet**

API 19: Android 4.4 (KitKat)

By targeting **API 19 and later**, your app will run on approximately **90.1%** of devices. [Help me choose](#)

☐ Include Android Instant App support

☐ **Wear**

API 21: Android 5.0 (Lollipop)

☐ **TV**

API 21: Android 5.0 (Lollipop)

☐ **Android Auto**

☐ **Android Things**

API 24: Android 7.0 (Nougat)

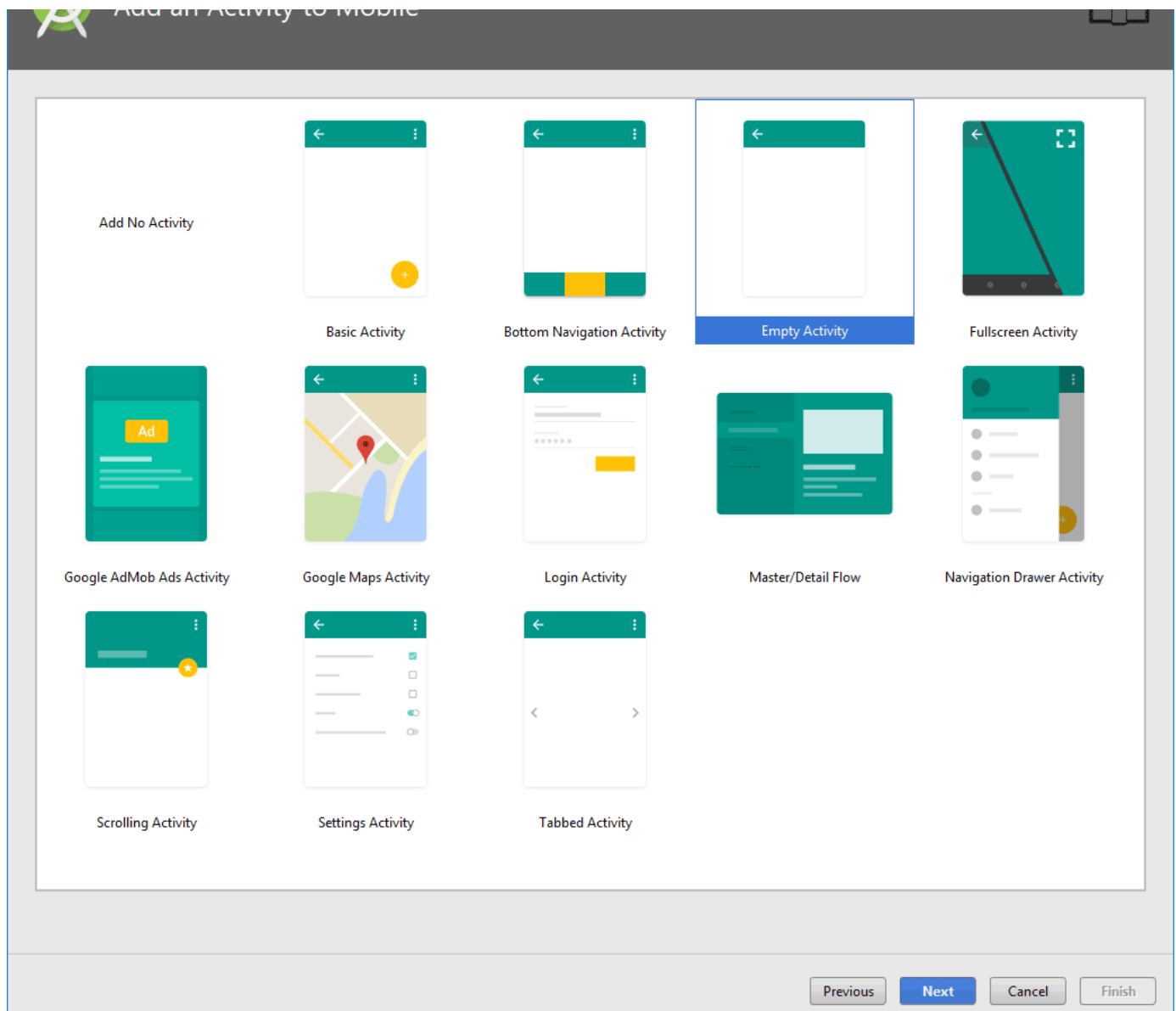
Previous Next Cancel Finish

Кроме выбора типа устройств, надо выбрать минимальную версию системы, под которую будет работать приложение. Выберите свой вариант. На данный момент Гугл поддерживает версии, начиная с API 7, выпуская специальные библиотеки совместимости для старых устройств. Но вы можете выбрать более современный вариант. У меня в наличии телефон с минимальной версией Android 4.4, поэтому я выставляю этот вариант.

Если щёлкнуть по ссылке [Help me choose](#), то откроется окно с графиком. Если вам интересно, можете посмотреть, но котиков там нет.

Идём дальше и снова нажимаем кнопку **Next**.

Здесь следует выбрать внешний вид экрана приложения.



Предложенные шаблоны позволяют сэкономить время на написание стандартного кода для типичных ситуаций. Опытный разработчик может вручную написать любой из предложенных вариантов, используя вариант **Add No Activity**, где никаких заготовок не будет.

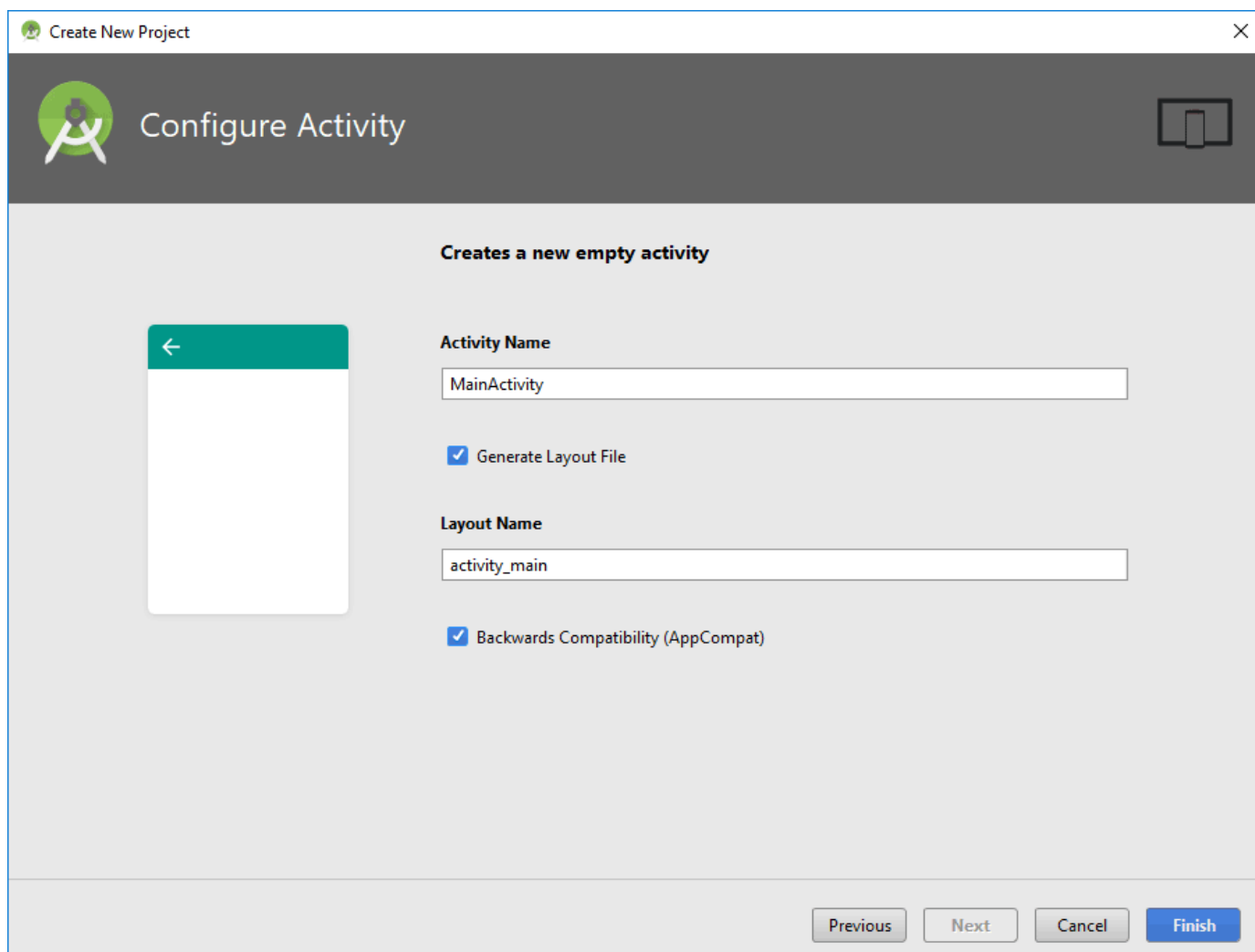
Несколько лет назад был только один шаблон. Список шаблонов постоянно меняется (добавляют и удаляют) и теперь их уже 13. Перечислю часть из них.

- **Basic Activity**
- **Bottom Navigation Activity**
- **Empty Activity**
- **Fullscreen Activity**
- **Google Maps Activity**
- **Google AdMod Ads Activity**
- **Login Activity**
- **Master/Detail Flow**

Шаблон **Empty Activity** предназначен для обычных телефонов. На картинке над названием шаблона вы видите приблизительный вид приложения с использованием данной заготовки. Для учебных программ в 99% подойдёт этот вариант. Практически все примеры на сайте написаны с помощью данного шаблона.

Шаблон **Master/Detail Flow** предназначен для планшетов с реализацией двухпанельного режима. Шаблон **Fullscreen Activity** можно использовать для игр, когда требуется дополнительное пространство без лишних деталей. Другие шаблоны нужны для создания приложений с гуглокартами или сервисами Google Play.

Итак, мы выбрали вариант **Empty Activity** и переходим к следующему окну.



The screenshot shows the 'Configure Activity' dialog box in Android Studio. The title bar says 'Create New Project'. The dialog has a dark header with the Android Studio logo and the text 'Configure Activity'. Below the header, the main area is light gray. On the left, there is a preview of a mobile app screen with a green header bar containing a white back arrow. To the right of the preview, the text 'Creates a new empty activity' is displayed. Below this, there are four fields: 'Activity Name' with the value 'MainActivity', 'Layout Name' with the value 'activity\_main', and two checked checkboxes: 'Generate Layout File' and 'Backwards Compatibility (AppCompat)'. At the bottom right, there are four buttons: 'Previous' (disabled), 'Next' (disabled), 'Cancel' (disabled), and 'Finish' (active, highlighted in blue).

Ничего не трогаем. Мы закончили с первоначальной настройкой. Нажимаем кнопку **Finish** и смотрим, что дальше будет.

А дальше студия формирует проект и создаёт необходимую структуру из различных файлов и папок. Поначалу глаза разбегаются. Давайте разбираться.

реже.

В левой части среды разработки на вкладке **Android** появится иерархический список из папок, которые относятся к проекту. В некоторых случаях желательно переключиться на режим **Project**, который показывает истинное расположение файлов. Но на первых порах удобнее использовать именно вид **Android**, который прячет служебные файлы, чтобы не путать новичков.

## Содержание проекта

Вкладка **Android** содержит две основные папки: **app** и **Gradle Scripts**. Первая папка **app** является отдельным модулем для приложения и содержит все необходимые файлы приложения - код, ресурсы картинок и т.п. Вторая папка служит для различных настроек, управления проектом и многих других вещей.

Сейчас нас должна интересовать папка **app**. Раскройте её. В ней находятся три папки: **manifest**, **java**, **res**.

### manifest

Папка **manifest** содержит единственный файл манифеста **AndroidManifest.xml**. В этом файле должны быть объявлены все активности, службы, приёмники и контент-провайдеры приложения. Также он должен содержать требуемые приложению разрешения. Например, если приложению требуется доступ к сети, это должно быть определено здесь. «AndroidManifest.xml» можно рассматривать, как описание для развертывания Android-приложения.

Более подробно о структуре манифеста читайте в дополнительной статье [Файл AndroidManifest.xml](#)

### java

Папка **java** содержит три подпапки - рабочую и для тестов. Рабочая папка имеет название вашего пакета и содержит файлы классов. Сейчас там один класс **MainActivity**. Папки для тестов можете не трогать. Если вы знаете, как работают пакеты в Java, то можете создавать новые папки и подпапки.

### res

Папка **res** содержит файлы ресурсов, разбитых на отдельные подпапки.

- **drawable** — в этих папках хранят графические ресурсы - картинки и xml-файлы, описывающие цвет и фигуры.
- **layout** — в данной папке содержатся xml-файлы, описывающие внешний вид форм и различных элементов форм. После создания проекта там уже имеется файл



- **values** — тут размещаются строковые ресурсы, ресурсы цветов, тем, стилей и измерений, которые мы можем использовать в нашем проекте. Здесь вы можете видеть файлы **colors.xml**, **strings.xml**, **styles.xml**. В старых проектах был ещё файл **dimens.xml**, сейчас от него отказались

Со временем вы будете свободно ориентироваться в этих папках, пока не забивайте себе голову.

## Работа с проектом - Здравствуй, Мир!

Как уже говорилось, программа **Hello, World!** уже встроена в любой новый проект, поэтому вам даже не нужно ничего писать. Просто нужно запустить проект и получить готовую программу!

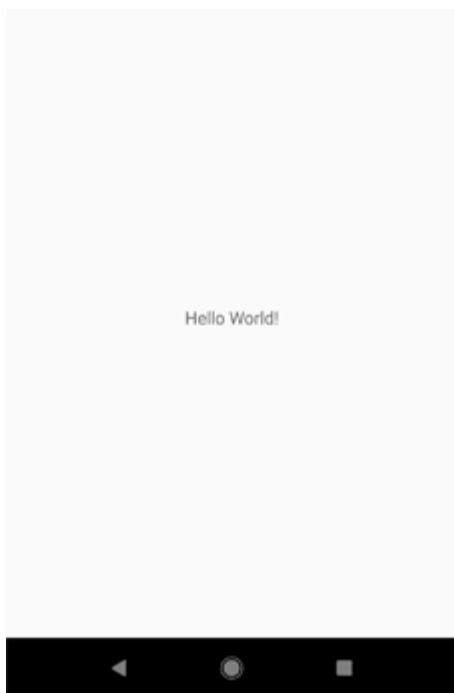
Для изучения вам нужно открыть два файла - **MainActivity** (скорее всего он уже открыт) и **activity\_main.xml (res/layout)** в центральной части Студии. Если файлы не открыты, то откройте их самостоятельно двойным щелчком для редактирования (или просмотра). Таким способом вы можете открыть любой нужный вам файл.

Не будем пока изучать код, а просто нажмём на зелёный треугольник **Run** (Shift+F10) на панели инструментов в верхней части студии для запуска приложения.

Если вы не настроили эмулятор, значит вы не читали предыдущий урок. Настройте сначала эмулятор и запускайте проект снова. Либо подключайте реальное устройство.

Если всё сделали правильно, то в эмуляторе или на устройстве загрузится ваша программа. Поздравляю!

Итак, если программа запустилась, то увидите окно приложения с надписью **Hello World**. Заголовок у программы будет также **Hello World**. Все эти строки можно найти в файле **res/values/strings.xml** и отредактировать при желании.



Теперь посмотрим на код. Сначала изучим **activity\_main.xml**.

Смотреть его можно в двух режимах - **Design** и **Text**.

Откройте в режиме **Text**.

```
<?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="ru.alexanderklimov.helloworld.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>
```

Это новый код шаблона, который появился в Android Studio 2.3 в марте 2017 года. Раньше использовался другой код с **RelativeLayout** (а ещё раньше и другой код с **LinearLayout**). Если вам будут попадаться старые примеры, то в студии есть контекстное меню, которое

теперь в этом коде вместо специального компонента **TextView**, в котором размещён компонент **TextView**, предназначенный для вывода текста.

Теперь посмотрим на Java-код (**MainActivity.java**)

```
package ru.alexanderklimov.helloworld;

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);
    }
}
```

Перед вами открыт файл класса, где имя класса **MainActivity** совпадает с именем файла с расширением **java** (это правило, установленное языком Java). В первой строке идет название пакета - его мы задавали при создании проекта (**Package Name**). Далее идут строки импорта необходимых классов для проекта. Для экономии места они свёрнуты в одну группу. Разверните её. Если однажды вы увидите, что имена классов выводятся серым цветом, значит они не используются в проекте (подсказка **Unused import statement**) и можете спокойно удалить лишние строки. Также они могут удаляться автоматически (настраивается).

Далее идёт объявление самого класса, который наследуется (**extends**) от абстрактного класса **Activity**. Это базовый класс для всех экранов приложения. Не исключено, что у вас будет **AppCompatActivity**, если при создании проекта вы оставили поддержку старых устройств (флажок **Backwards Compatibility (App Compat)**). В старых версиях не было плюшек, которые появились после Android 4, поэтому для них была создана специальная библиотека совместимости, которая позволяет использовать новинки от новых версий Android в старых программах. Класс **AppCompatActivity** как раз и относится к библиотеке совместимости. Считайте её бедным родственником базовой **Activity**. У неё есть все нужные методы и вспомогательные классы, но названия могут немного различаться. И смешивать названия нельзя. Если уж используете класс из библиотеки совместимости, то методы берите соответствующие.

На разных этапах использовались разные названия класса активности, которые могут вам встретиться в старых проектах. Например, сначала использовался **FragmenActivity**, затем **ActionBarActivity**, а 22 апреля 2015 года вышла новая версия

В самом классе мы видим метод **onCreate()** – он вызывается, когда приложение создаёт и отображает разметку активности. Метод помечен как **protected** и сопровождается аннотацией **@Override** (переопределён из базового класса). Аннотация может пригодиться вам. Если вы сделаете опечатку в имени метода, то компилятор сможет предупредить вас, сообщив об отсутствии такого метода у родительского класса **Activity**.

Разберём код метода.

Строка **super.onCreate(savedInstanceState);** – это конструктор родительского класса, выполняющий необходимые операции для работы активности. Эту строчку вам не придётся трогать, оставляйте без изменений.

Вторая строчка **setContentView(R.layout.activity\_main);** представляет больший интерес. Метод **setContentView(int)** подключает содержимое из файла разметки. В качестве аргумента мы указываем имя файла без расширения из папки **res/layout**. По умолчанию проект создаёт в нём файл **activity\_main.xml**. Вы можете переименовать файл или создать свой файл с именем **cat.xml** и подключить его к своей активности. Тогда код будет выглядеть так:

```
setContentView(R.layout.cat);
```

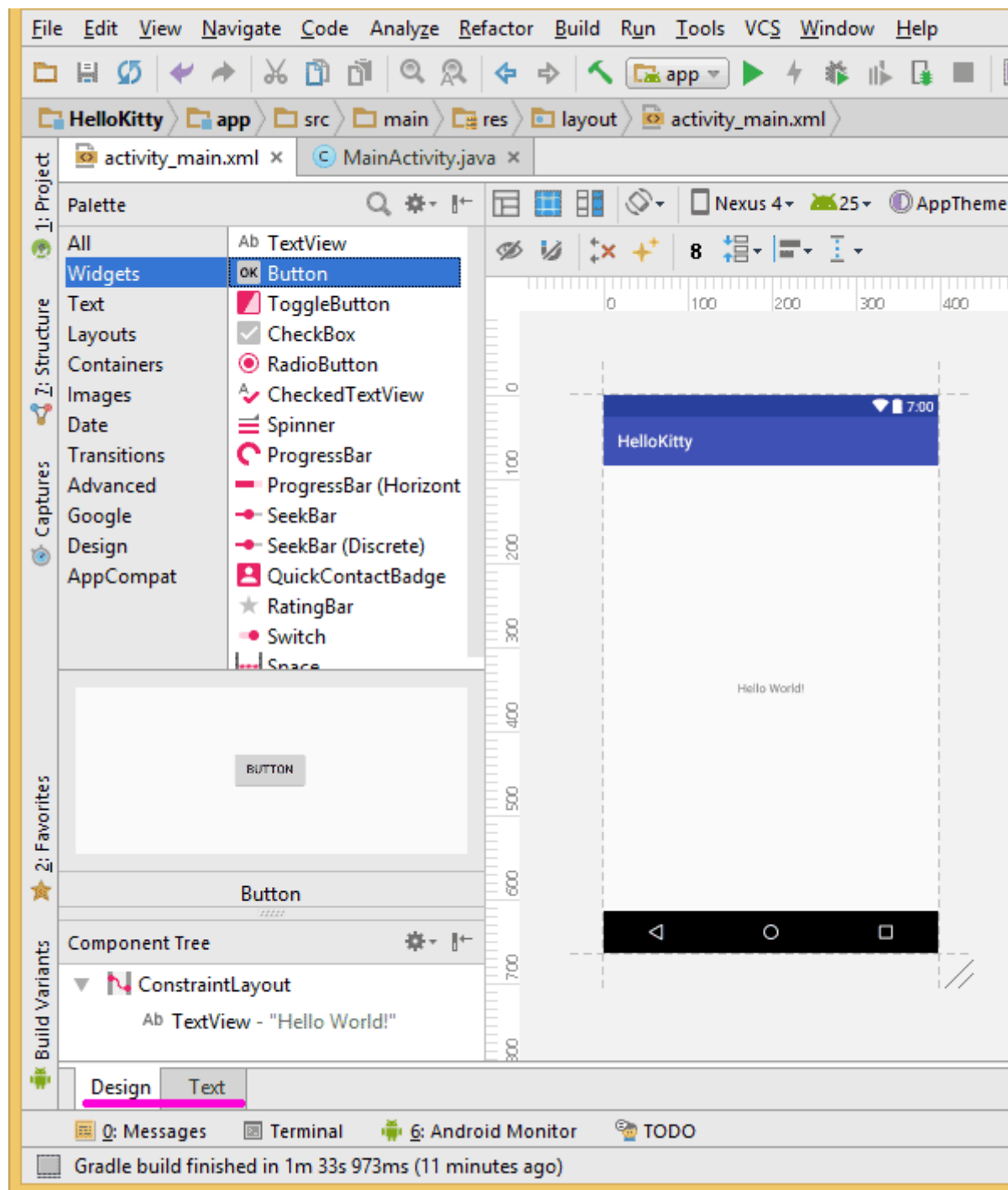
Чтобы ваш код был аккуратным, старайтесь придерживаться стандартов. Если вы создаёте разметку для активности, то используйте префикс **activity\_** для имени файла. Например, разметка для второй активности может иметь имя **activity\_second.xml**.

## Hello Kitty!

Вы создали новую программу, но это ещё не повод считать себя программистом, так как вы не написали не единой строчки кода. Настало время набраться смелости и создать программу "Hello Kitty!". На данный момент наша программа слишком проста. Представьте себе, что у вас на экране должны располагаться несколько кнопок, текстовых полей, картинок. Каждому объекту нужно задать размеры, координаты, цвет, текст и так далее. Android поддерживает способ, основанный на XML-разметке, который будет напоминать разметку веб-страницы. Начинающие программисты могут использовать визуальный способ перетаскивания объектов с помощью мыши. Более продвинутые могут писать код вручную. Чаще используется комбинированный подход.

Файлы XML-разметки находятся в папке **res/layout** вашего проекта. Слово "res" является сокращением от слова "resources" (ресурсы). Папка содержит ресурсы, не связанные с кодом. Кроме разметки, там же содержатся изображения, звуки, строки для локализации и т.д.

просматривать в двух режимах: текстовом и визуальном. Для этого предназначены две вкладки в нижней части окна редактора: **Design** и **Text**.



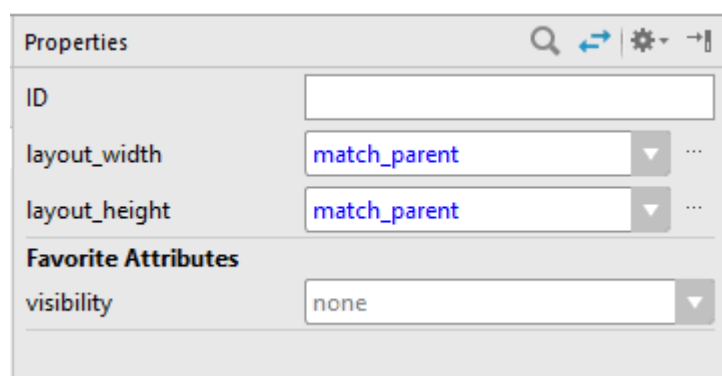
Переключитесь в режим **Text**.

Структура XML-файла достаточно проста - стандартное дерево XML-элементов, где каждый узел является именем класса **View** (**TextView** - один из элементов **View**). Вы можете создать интерфейс программы, используя структуру и синтаксис XML. Подобный подход позволяет разделить код программы и визуальное представление.

подглядывая в описание. Назовите проект **Hello Kitty** и повторите все предыдущие шаги.

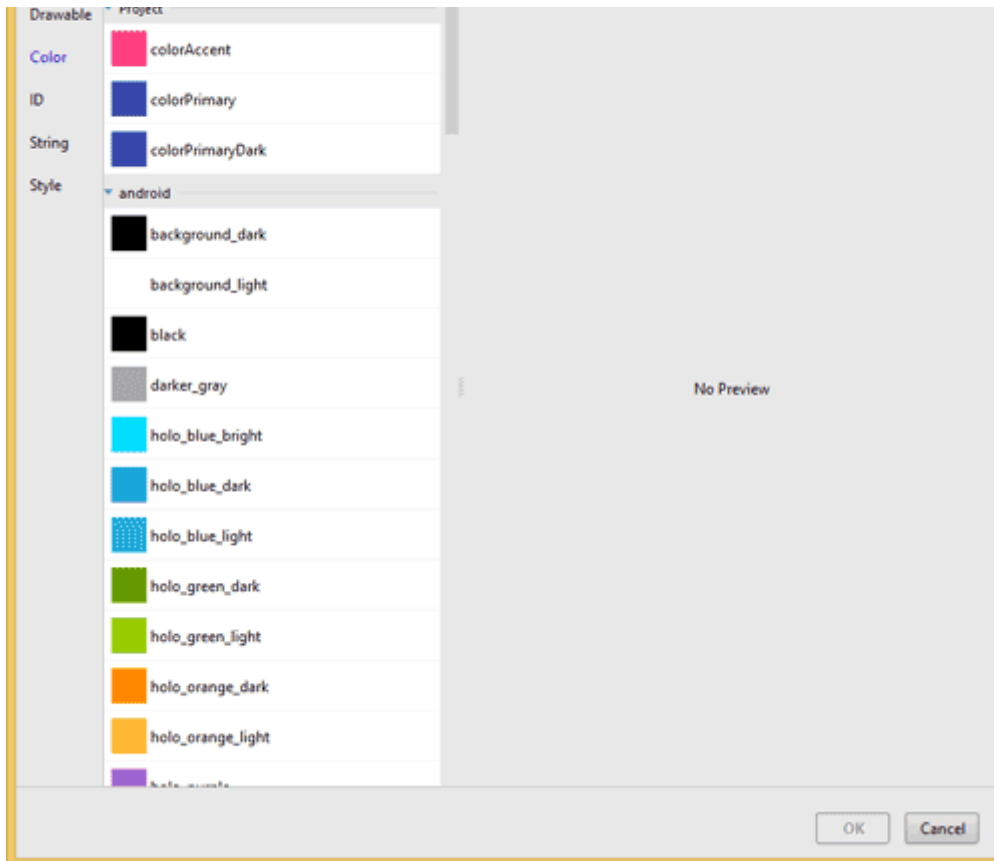
Когда разметка открыта в графическом представлении, то слева от основной части редактора кода можно увидеть панель инструментов, в которой сгруппированы различные элементы по группам **Widgets**, **Texts**, **Layouts** и так далее. В группе **Images** найдите элемент **ImageButton**, перетащите её на форму и отпустите. Точное расположение нас не интересует, поэтому не заморачивайтесь по этому поводу, постарайтесь разместить компонент в центре экрана активности. У вас появится диалоговое окно с просьбой выбрать изображение для кнопки. Пока выбирайте любую, например, первую. Потом заменим.

Теперь научимся менять фон для экрана приложения. Сейчас у нас экран белого цвета. Возвращаемся в файл разметки **activity\_main.xml**. Справа найдите вкладку **Properties**, в которой отображаются свойства для выбранного элемента. А слева есть вкладка **Component Tree**, который отображает структуру компонентов на экране. Вам нужно выделить какой-нибудь компонент, что на вкладке свойств увидеть все доступные свойства компонента. Новички часто путаются на первых порах и начинают менять свойства не у тех элементов, которые им были нужны. Сейчас у вас есть окно активности, графическая кнопка **ImageButton** и текстовая метка **TextView** с надписью **Hello World!**. Пощёлкайте по этим элементами, чтобы увидеть, как меняется содержание свойств в панели свойств. Так как мы собираемся работать с фоном экрана приложения, то щёлкните на **ConstraintLayout**. В панели свойств отобразятся самые употребительные свойства выбранного компонента. К ним относятся идентификатор, ширина и высота.



Щёлкаем по кнопке **View all properties** (две стрелочки), чтобы открыть все свойства компонента.

Найдите свойство **background**. Щелкните рядом с этим словом во второй колонке, где нужно прописывать значения. Появится текстовое поле, в которое можно ввести значение вручную, и кнопка с тремя точками, которая запустит диалоговое окно для создания ресурса.



Переходим на вкладку **Color** и выбираем цвет. Сверху в разделе **Project** показаны цвета, определённые в проекте в стиле Material Design. В разделе **Android** показаны цвета, существующие в ресурсах системы. При желании можно установить свой цвет. Давайте выберем цвет **colorAccent** и нажмём кнопку **OK**.

Экран окрасится в розовый цвет. Получилось глаМYPненько.

Если переключиться в текстовый режим, то увидим, что у элемента **ConstraintLayout** добавилась строчка:

```
android:background="@color/colorAccent"
```

Мы связали фон экран с именем цветового ресурса. Существует неправильный подход, когда можно сразу напрямую указать значение цвета.

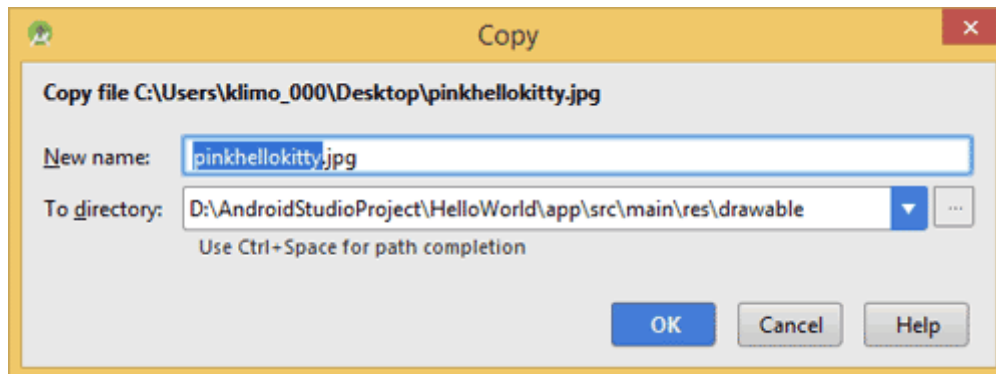
```
android:background="#ffffc0cb"
```

Старайтесь так не делать, всегда используйте ресурсы.

Далее поменяем картинку для графической кнопки. Находим подходящее изображение и копируем его в папку **res/drawable**. Картинку можете взять у меня.

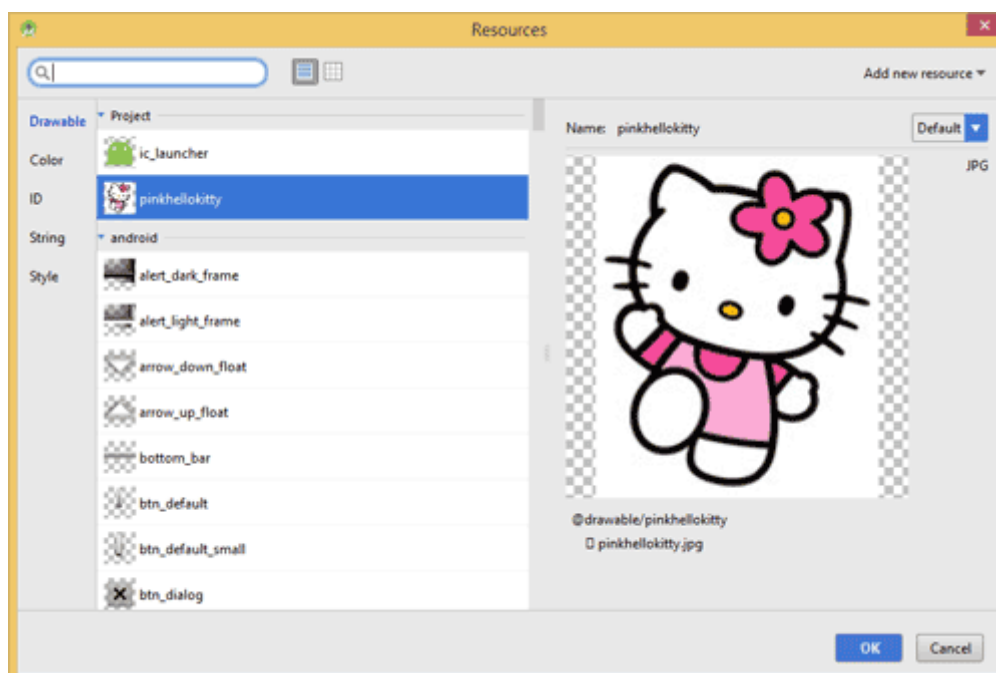


Простое перетаскивание из проводника в папку студии не сработает. Поэтому лучше скопировать картинку в буфер, затем щёлкнуть правой кнопкой мыши на папке **drawable** в студии, выбрать команду "Вставить".



Когда вы поместите графический файл в указанную папку, то студия автоматически создаёт ресурс типа **Drawable** с именем файла без расширения, к которому можно обращаться программно.

Выделяем элемент **ImageButton** на форме и в панели свойств откроем только важные свойства, выбираем свойство **srcCompat**. Снова щёлкаем на кнопке с тремя точками и выбираем ресурс в категории **Drawable** - вы там должны увидеть ресурс *pinkhelloworld* (имя добавленного ранее файла).





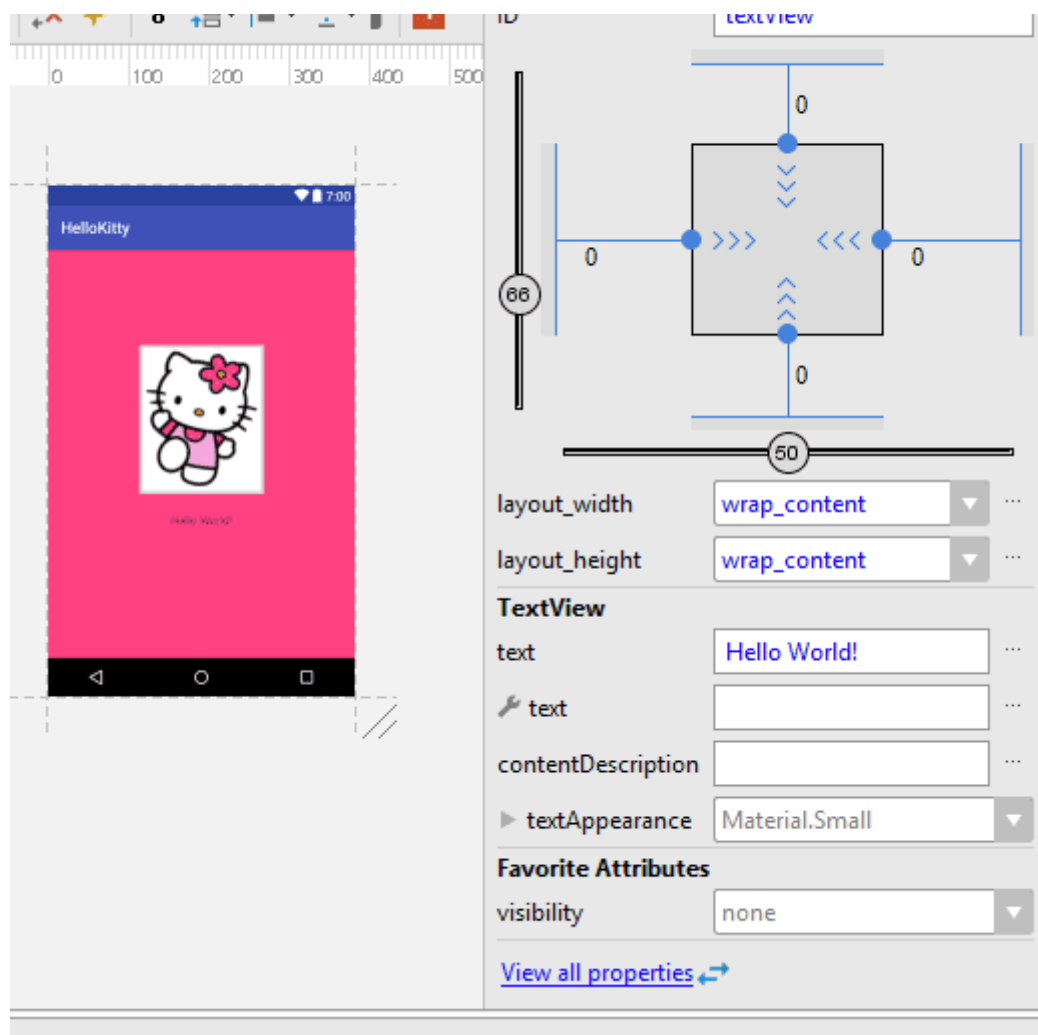
т.д. использовать нельзя.

Там же в окне свойств находим свойство **onClick** и вручную прописываем **onClick** - это будет именем метода для обработки нажатия на кнопку. Вы можете придумать и другое имя, например, **onButtonPressed**, но я привык к такому подходу.

Мы закончили работу с графическим интерфейсом приложения. Напоследок, выделите элемент **TextView** с надписью **Hello, World** и в окне свойств посмотрите на его идентификатор (ID). Если там пусто, то удалите его, он не оправдал наших надежд. В разделе **Widgets** найдите компонент **TextView** и перетащите его на форму приложения. Постарайтесь разместить его под графической кнопкой с котёнком.

У этого компонента точно будет что-то написано в свойстве **id**. Скорее всего, это будет **textView**. Запомните его. Впрочем, мы могли не удалять первый компонент, а прописать идентификатор вручную. Но мне пришлось бы объяснять лишние детали, а так сразу получили результат. Вот я не удалял его и у меня экран выглядит так. А у вас будет текст **TextView**. Ничего страшного.

Если текст вам кажется мелковатым, то у свойства **textAppearance** установите значение **AppCompat.Display2**.



У меня получилось следующее:

```
<android.support.constraint.constraint.layout 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"
android:background="@color/colorAccent"
tools:context="ru.alexanderklimov.hellokitty.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:id="@+id/textView"
    android:layout_marginTop="0dp"
    app:layout_constraintVertical_bias="0.668"/>
```

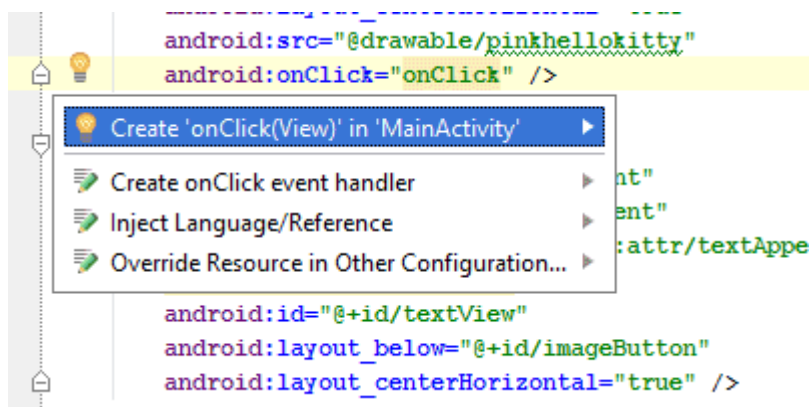
```
<ImageButton
    android:id="@+id/imageButton"
    android:layout_width="164dp"
    android:layout_height="199dp"
    android:layout_marginBottom="8dp"
    android:layout_marginLeft="8dp"
    android:layout_marginRight="8dp"
    android:layout_marginTop="8dp"
    android:onClick="onClick"
    android:scaleType="centerCrop"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintHorizontal_bias="0.502"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.353"
    app:srcCompat="@drawable/pinkhellokitty"
    tools:layout_editor_absoluteX="104dp"
    tools:layout_editor_absoluteY="101dp"/>
```

```
</android.support.constraint.ConstraintLayout>
```

Здесь очень много кода, который генерирует **ConstraintLayout** при размещении и выравнивании компонентов. Это новый контейнер, к которому нужно привыкать. Сейчас нет смысла объяснять принцип его работы, на сайте есть отдельная статья по нему.



В всплывающем окне выберите вариант **Create 'onClick(View)' in 'MainActivity'**.



В коде класса **MainActivity** появится заготовка для обработки щелчка кнопки.

Раз уж у нас теперь открыт файл **MainActivity.java**, то продолжим теперь работу в нём. Так как мы собираемся менять текст в текстовой метке, необходимо прописать данный элемент в коде. До метода **onCreate()** наберите строчку:

```
private TextView mHelloTextView;
```

Мы объявили переменную типа **TextView** под именем **mHelloTextView**.

Если вы набирали вручную и при подсказках использовали клавишу Enter, то класс **TextView** автоматически импортируется и запись появится в секции **import**. Если вы просто копируете текст с сайта, то студия подчеркнёт название класса **TextView** и предложит импортировать его вручную.

Далее внутри метода **onCreate()** после вызова метода **setContentView()** добавьте строку:

```
mHelloTextView = (TextView)findViewById(R.id.textView); // помните, я просил запомнить идентификатор?
```

активно помогает подсказками. Теперь система знает о существовании элемента **TextView**, и мы можем к нему обращаться для изменения различных свойств, например, поменять текст.

Если вы используете компилятор **compileSdkVersion 26**, то скобки с названием компонента можно опустить, так как изменилась сигнатура метода.

```
// было
public View findViewById(int id);

// стало
public <T extends View> T findViewById(int id);
```

Пишите теперь так.

```
mHelloTextView = findViewById(R.id.textView);
```

Эта возможность появилась совсем недавно, поэтому во всех примерах используется старый код. Можете сразу привыкать к новому способу, студия будет подсказывать.

Переходим к заготовке для щелчка кнопки.

```
public void onClick(View view) {
}
```

В следующих занятиях мы подробнее разберём работу с данным методом, пока просто пишем код между фигурными скобками:

```
mHelloTextView.setText("Hello Kitty!");
```

Мы обращаемся к элементу **mHelloTextView** и через его метод **setText()** программно меняем текст на нужные слова.

Запускаем программу и нажимаем на кнопку с изображением котёнка. Если всё сделали правильно, то отобразится замечательная фраза. С этого момента можете считать себя настоящим программистом - вы научились создавать цветовые и графические ресурсы, менять фон у приложения через XML-разметку, обрабатывать нажатия кнопки и выводить текстовые сообщения.



Hello Kitty!

В папке **app\build\outputs\apk** проекта можно найти готовый APK-файл, который вы можете выложить у себя на сайте и дать скачать знакомым девушкам и парням (в телефоне должно быть разрешение на установку неподписанных приложений), вы станете невероятно круты в их глазах.

Забегая вперёд, скажу одну вещь. Каждому приложению выделяется определённый объем памяти под картинки. В новых устройствах чуть больше, в старых поменьше. Но в любом случае не нужно пытаться загрузить в этом примере фотографию своего любимого кота объёмом в несколько десятков мегабайт, иначе можете получить ошибку в приложении. Позже вы узнаете, как лучше использовать большие картинки.

## Исходный код для ленивых

► Ткните лапкой, чтобы развернуть текст

## Здороваемся с вашим котом

Программа получилась замечательная, но у неё есть недостаток. Она показывает одну и ту же фразу "Hello Kitty!". Вряд ли ваш кот знает английский, да и здороваться лучше по имени. Не пытайтесь с котом мяукать, иначе разговор выглядит следующим образом.



Поздороваемся с котом по человечески. Найдите в разделе **Text** компонент **Plain Text** и перетащите его на экран активности, разместив где-то над картинкой. Оставляем все свойства без изменений, разве только в свойстве **hint** можно добавить строчку-подсказку, которая будет исчезать при вводе текста.

```
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_marginBottom="8dp"
android:layout_marginLeft="8dp"
android:layout_marginRight="8dp"
android:layout_marginTop="8dp"
android:ems="10"
android:hint="Введите имя кота"
android:inputType="textPersonName"
android:text="Name"
.../>
```

Переходим в класс **MainActivity** и добавляем новую переменную рядом с переменной **mHelloTextView**:

```
private EditText mNameEditText;
```

Свяжем созданную переменную с компонентом в методе **onCreate()**:

```
mNameEditText = (EditText) findViewById(R.id.editText);
```

Поменяем код для щелчка кнопки.

```
public void onClick(View view) {
    if (mNameEditText.getText().length() == 0) {
        mHelloTextView.setText("Hello Kitty!");
    } else {
        mHelloTextView.setText("Привет, " + mNameEditText.getText());
    }
}
```

Мы внесли небольшую проверку. Если в текстовом поле пустой текст, то длина текста составляет ноль символов, и мы по-прежнему выводим надпись "Hello Kitty!". Если пользователь введёт имя своего кота, то приложение поздоровается с ним. Какая умная и вежливая программа у нас получилась.



Мурзик



Привет, Мурзик

Для сравнения напишем пример [Hello Kitty на Kotlin](#)

## Дополнительное чтение

[Обсуждение статьи](#) на форуме.

[Исходник на Гитхабе](#)

Реклама

Реклама



ozon.ru

LED  
Телевизор  
Thomson

10 989 ₽

**9 990 ₽**

81 см  
32"

ОАО «И...  
г. Москва  
ул. 41, Пав...  
10277392