

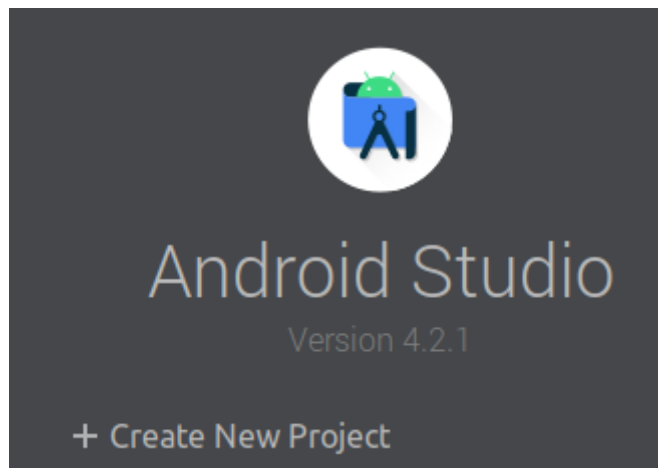
# Hello Android

Все что вам надо знать о тексте

## Содержание

1. Первое приложение на Андроид, структура проекта
2. Текстовый элемент

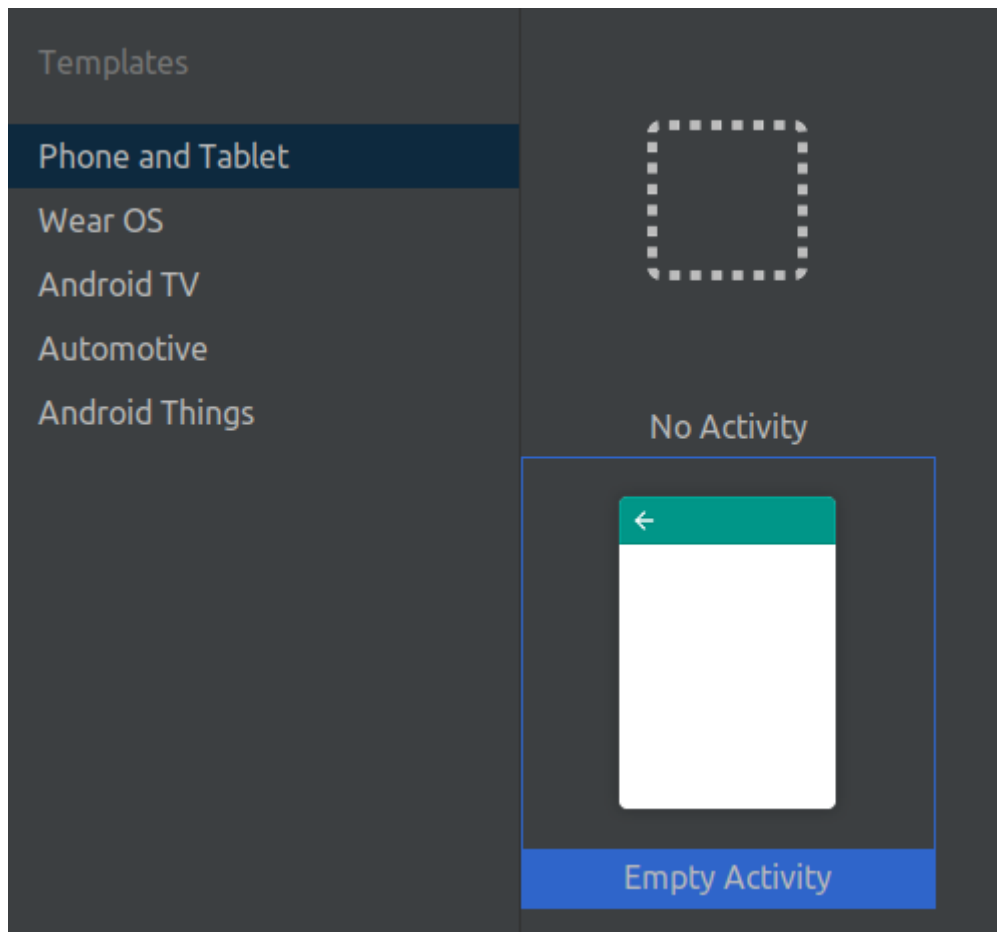
### 1. Первое приложение на Андроид, структура проекта



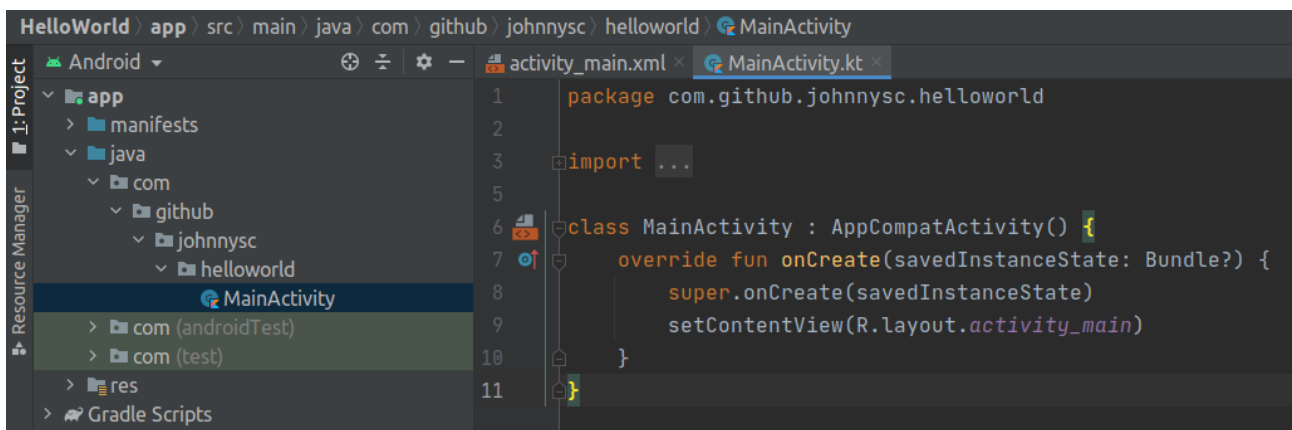
Итак, скачиваем продвинутую среду разработки под Андроид и создаем новый проект.

Для этого пользуемся шаблонами (скриншот на стр.2) – выбираем проект под телефоны и планшеты и выбираем пустую активити (обо всем подробнее далее). Далее выбираем язык – Kotlin, даем имя проекту например HelloWorld, проверяем чтобы проект создавался в папке AndroidStudioProjects и выбираем минимальное api 21. Поверьте мне, если выбрать еще меньше то страдать придется больше с поддержкой андроид ниже 5. О версиях мы еще поговорим позже. Итак, ждем Finish.

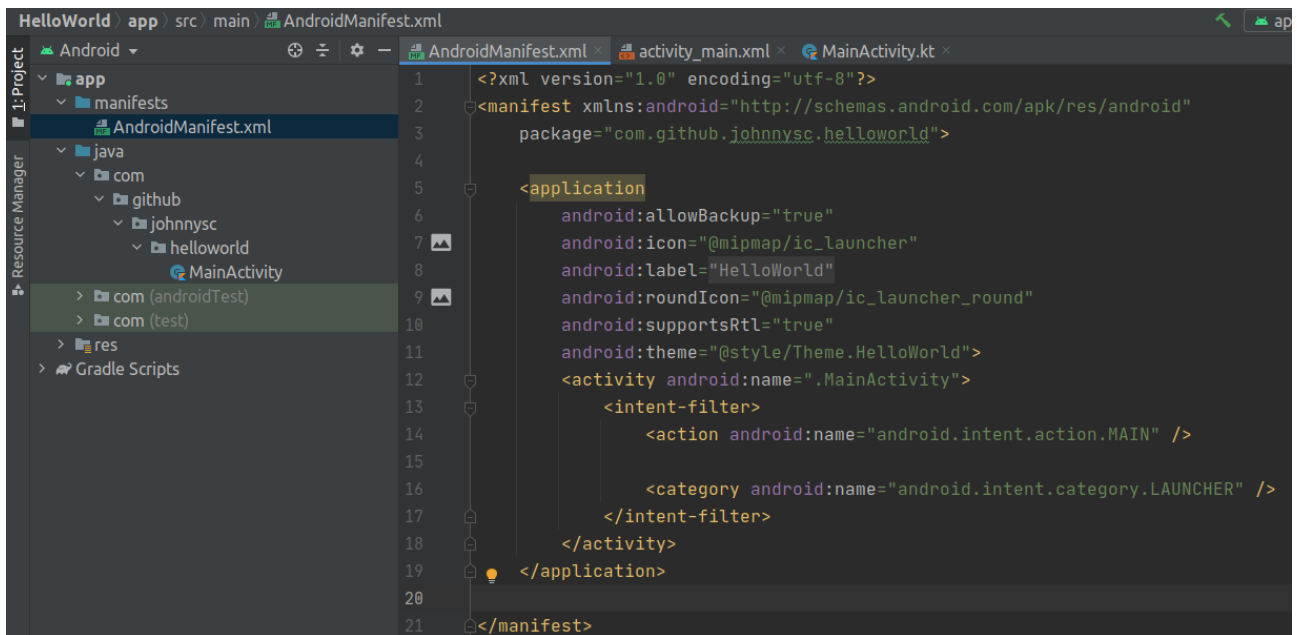
Name	HelloWorld
Package name	com.github.johnnysc.helloworld
Save location	/home/johnny/AndroidStudioProjects/HelloWorld
Language	Kotlin
Minimum SDK	API 21: Android 5.0 (Lollipop)



Я не буду говорить о том, что вам надо обновить/скачать библиотеки в SDK manager, об этом сказано сто раз уже на офф.сайте андроид студии. Теперь же, когда ваш проект собрался он должен выглядеть вот так



Слева у нас дерево проекта – то же самое что и File Explorer, ведь наши классы это те же джава и котлин классы которые лежат в файлах MainActivity.kt. Обратите внимание, у дерева поставлен вид отображения Android в верхнем левом углу и иконка. Как видите есть несколько главных папок (далее пакетов) – manifests, java, (androidTest), (test), res. О каждом скажем пару слов далее. И у нас в главном окне открыт файл MainActivity.kt – помните мы в IntelliJ Idea писали class Main и внутри был public static void main(String[] args) ? Так вот, мейн там и мейн здесь то же самое (грубо говоря). Вы можете убедиться в этом просто открыв пакет manifests и увидите там AndroidManifest



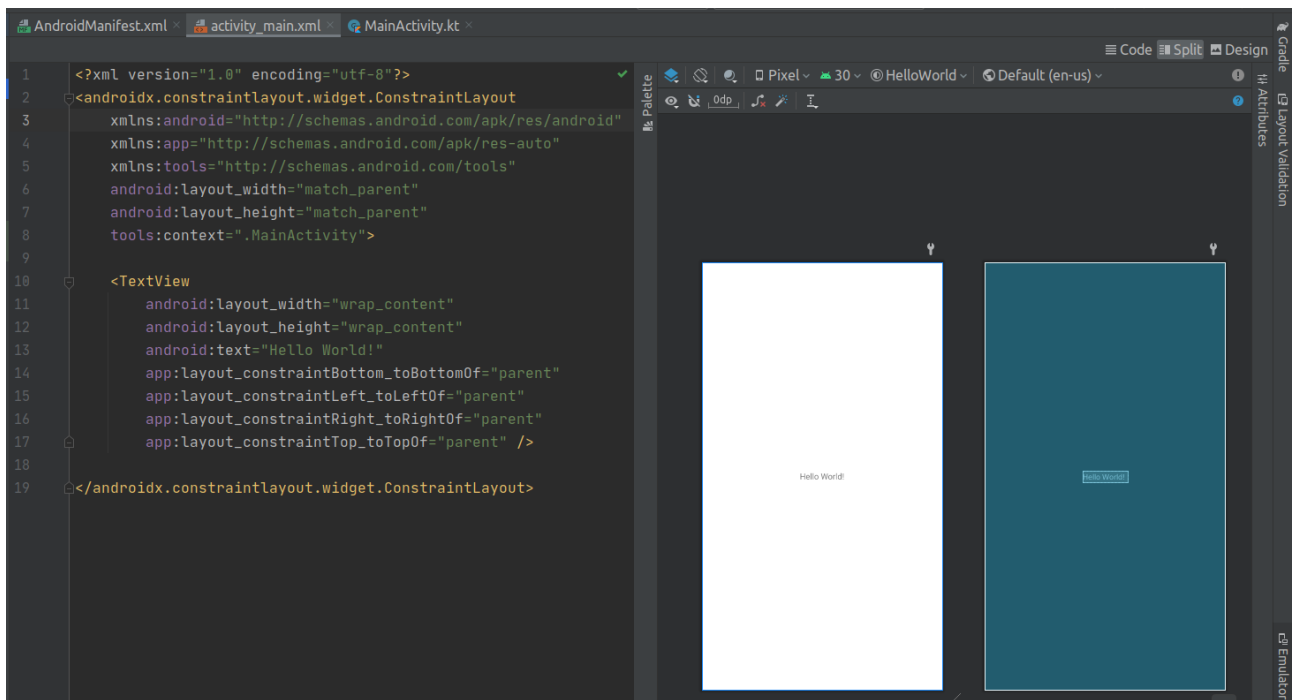
Андроид манифест это главный файл нашего проекта – в нем на данный момент указано имя проекта, и главный тег application с темой, иконкой и главной активити которая запускается на старте (LAUNCHER - когда вы тапаете на иконку на рабочем столе андроид). Обо всем этом подробнее позже. Но как минимум сейчас вы поняли, что мейн активити это именно тот класс, который запустится первым. Вернемся к нему (и да, в андроид мы пишем не только котлин или джава код, а еще у нас xml).

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

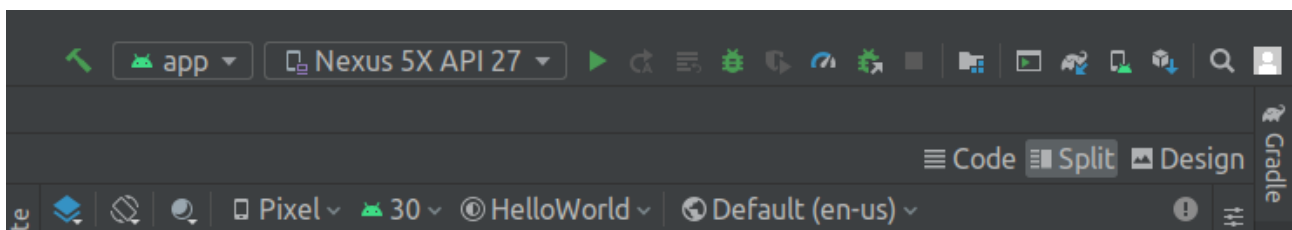
Метод onCreate будет нашим public static void main ибо он запустится первым. Значит все написанное в этом методе будет исполнено. Как видите здесь лишь 1 линия кода – setContentView(R.layout.activity\_main). Если вы заметили, то это имя xml файла который открыт в соседнем окне. Можете перейти в него с помощью клика с зажатым Ctrl. Что значит этот метод? Установить отображение из файла – мы же пишем приложение с юзер интерфейсом и нам надо где-то класть визуальные объекты. Посмотрим же на него.

В правом верхнем углу выберите Split, там где Code Split Design и вы увидите как xml так и дизайн нашего главного экрана. Да, здесь уже есть текстовка Hello World! По центру экрана.

Вы можете запустить проект на мобилке или на эмуляторе (сами погуглите как) для этого надо нажать на Run предварительно создав (если нет) эмулятор или подключив мобилку.

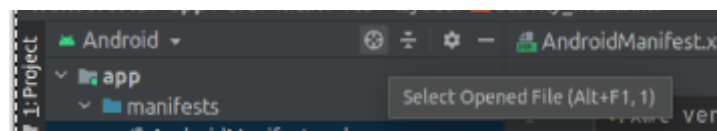


Слева направо – собрать проект, выбрать что запускаем (app это само приложение, можно выбрать запуск тестов – классы в пакетах test и androidTest), далее выпадашка с девайсами (я настроил эмулятор), RUN запуск приложения (или теста), дебаг и так далее. После иконки слоника идет иконка эмулятора, там можете настроить эмулятор и запустить его (да, можно сначала запустить эмулятор и потом уже нажать на RUN). Итак, давайте посмотрим как оно выглядит на эмуляторе (на след. странице).

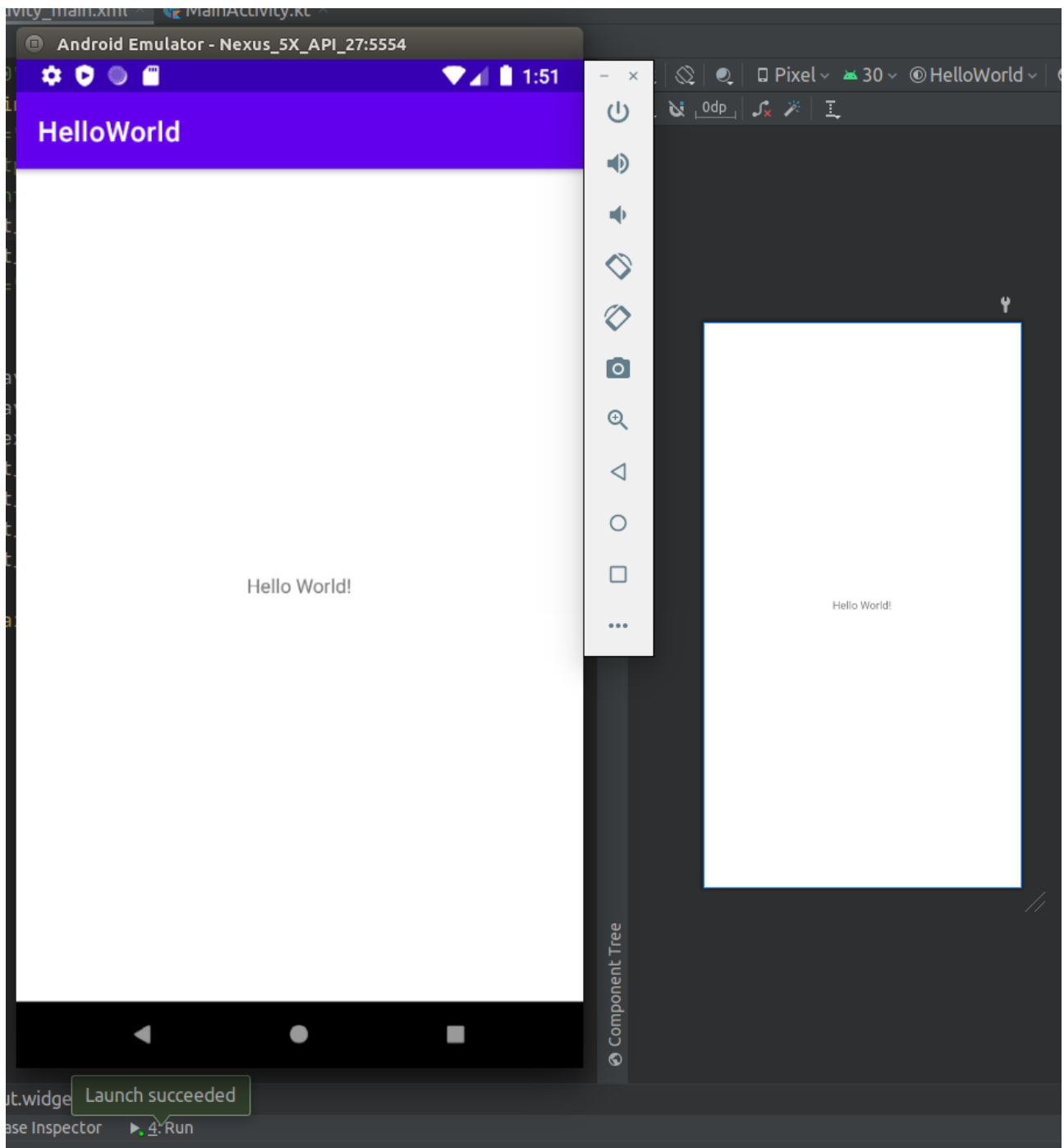


Как видите точно так как было представлено в дизайне activity\_main.xml

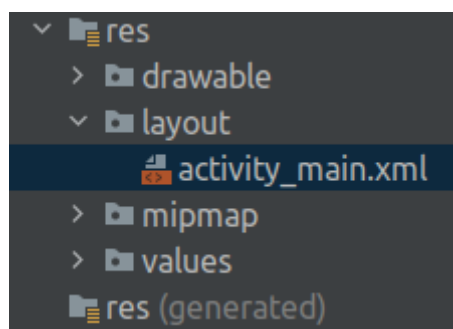
Кстати, у вас может возникнуть вопрос, а где находится сам файл activity\_main.xml Для этого в дереве объектов есть иконка метки, с ее помощью можно перейти в пакет где оно лежит.

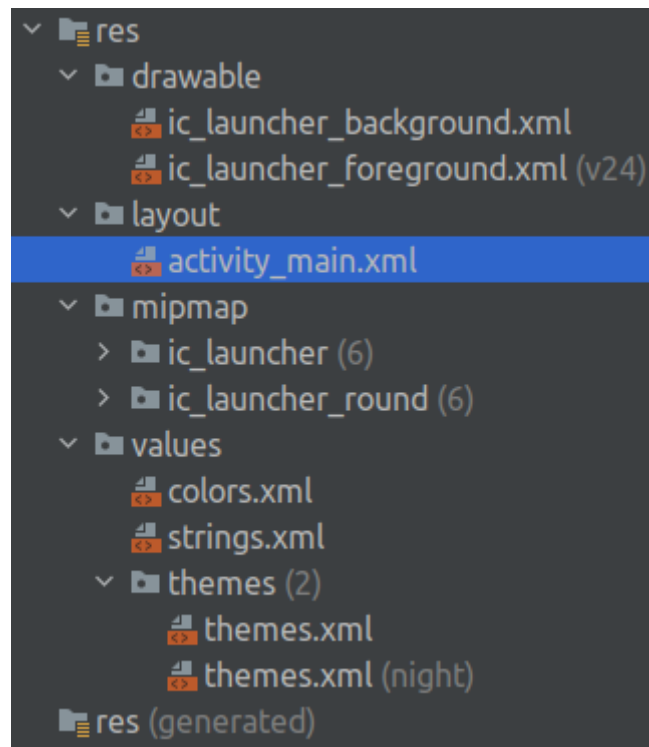


У вас должно быть активным окно xml файла и если нажать на метку, то перейдем в дереве в пакет res. Да, как вы догадались, это ресурсы нашего проекта. Если в пакете java мы кладем наш код на java или kotlin, то в пакет res мы кладем ресурсы – xml файлы, или png и другие, которые не являются кодом на java/kotlin. Давайте посмотрим на это.



Как видите в пакете res несколько пакетов – layout для тех xml файлов которые хранят визуальное отображение наших activity (и фрагментов и их составляющих). В пакете drawable будем класть картинки и все что это напоминает, в mipmap у нас будет иконка нашего приложения. Давайте раскроем все пакеты и посмотрим что внутри.





Подробно мы будем говорить по мере прохождения раздела Android. Пакет res(generated) не трогайте, там все сгенерированные ресурсы (об этом тоже позже). Если вам нужен цвет, то вы кладете его в colors.xml если же строка, то в strings.xml.

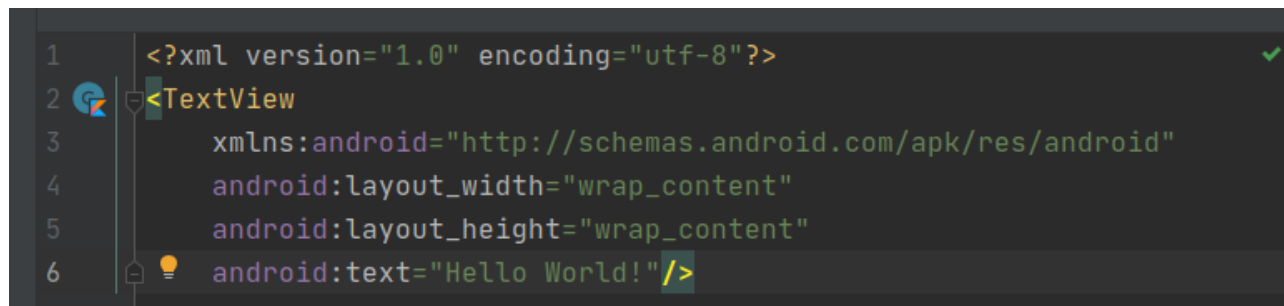
Давайте вернемся к нашему xml файлу и наконец посмотрим что там внутри.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <androidx.constraintlayout.widget.ConstraintLayout
3      xmlns:android="http://schemas.android.com/apk/res/android"
4      xmlns:app="http://schemas.android.com/apk/res-auto"
5      xmlns:tools="http://schemas.android.com/tools"
6      android:layout_width="match_parent"
7      android:layout_height="match_parent"
8      tools:context=".MainActivity">
9
10     <TextView
11         android:layout_width="wrap_content"
12         android:layout_height="wrap_content"
13         android:text="Hello World!"
14         app:layout_constraintBottom_toBottomOf="parent"
15         app:layout_constraintLeft_toLeftOf="parent"
16         app:layout_constraintRight_toRightOf="parent"
17         app:layout_constraintTop_toTopOf="parent" />
18
19 </androidx.constraintlayout.widget.ConstraintLayout>
```

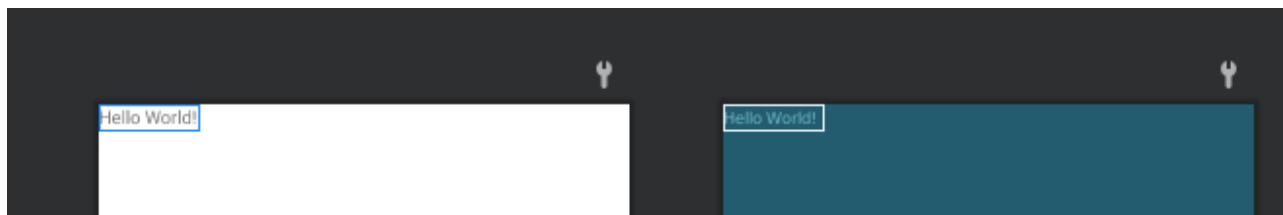
Как видите здесь 2 штуки – ConstraintLayout и TextView. Кто-то может сказать – но у нас на экране виден лишь текст HelloWorld! Зачем нам первый элемент? Действительно, оно нам не нужно (мы о нем поговорим позже). Давайте удалим его руками и оставим только текст.

## 2. Текстовый элемент

Кстати, вы заметили иконку на 2ой линии? Это быстрый доступ к классу. Нажмите на него и вас перекинет в котлин код, там же есть подобная иконка для перехода в xml файл.

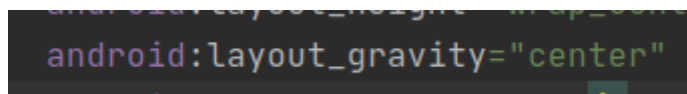


Теперь всяко лучше, правда? 6 линий вместо 19. Кстати, вы заметили зеленую галку в правом верхнем углу? Это значит что с этим файлом нет проблем. Все ОК. Но посмотрите теперь как выглядит наш текст на экране



Он сбилсся в левый угол. Кстати, у нас 2 вида – первый это как будет в реальности, а второй чисто схематический. Вы можете поменять фон приложения, но в blueprint не будет этого видно. Можете запустить проект и вы увидите что действительно наш текст ушел в левый верхний угол. Почему же так? Раньше наш текст лежал в контейнере ConstraintLayout и ему было указано положение – быть в центре (да, те 4 линии с 14 по 17). Теперь же нашему тексту не указано где быть и потому по умолчанию он стоит в верхнем левом углу. Да, наша система координат идет сверху вниз и слева направо. Т.е. верхняя левая точка это  $x = 0$ ,  $y = 0$  (для особо любопытных – нижняя правая точка это 720 1280 если у вас девайс с разрешением 360 на 640 и плотностью 2x, обо всем этом в другой лекции про экраны).

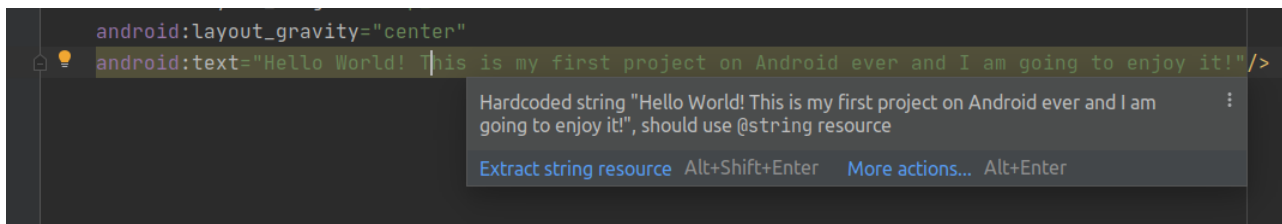
Теперь, как исправить? Надо указать ему `layout_gravity` – по сути мы говорим куда ему тяготеть - если выбрать к центру, то текстовка будет стараться находиться по центру. По центру чего? Если раньше оно находилось в контейнере ConstraintLayout, то теперь же ничего нет – значит по центру того окна, в котором оно находится – т.е. MainActivity.



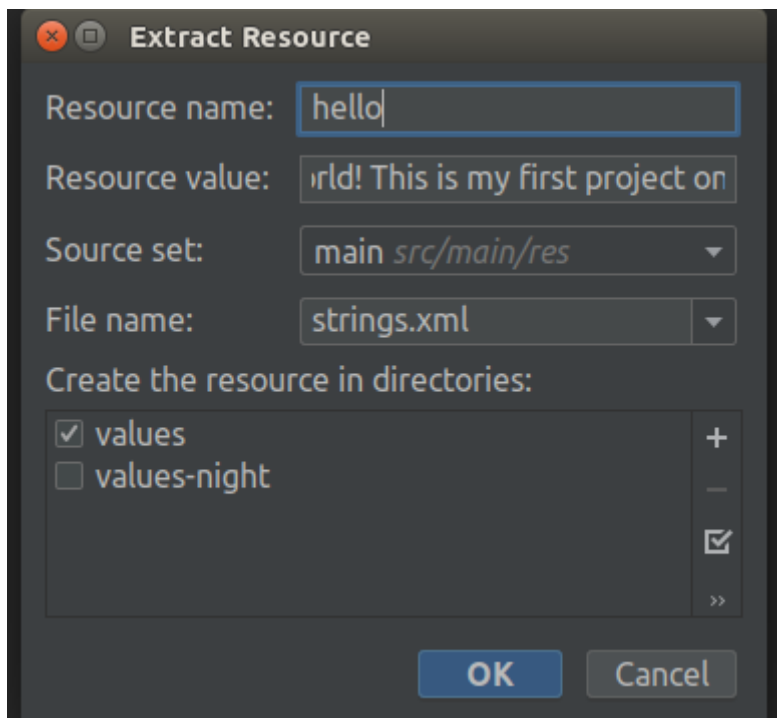
Можете запустить проект и убедиться что все как раньше.

Но давайте попробуем теперь написать нечто длиннее чем Hello World! Просто добавим текста в поле и посмотрим что будет.

Помните мы в идее всегда использовали комбинацию Alt+Enter? То же самое можно применять и здесь. Как видите Андроид Студио (далее АС) говорит нам что мы захардкодили текстовку прямо в хмл файле и это не хорошо. И оно сразу же дает нам подсказку что делать.

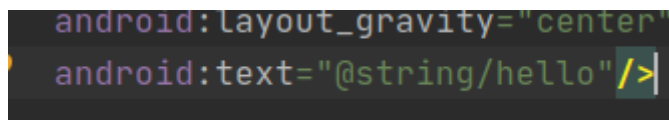


Нажмите на Extract string resource или же нажмите на Alt+Enter и выберите из меню



Мы можем выделить в константу скажем так и наш хмл файл будет выглядеть красивее. Но ведь проблема не только же в красоте, верно? Да, именно так. Как видите у нас есть еще выбор для values-night, то есть для значений для темной темы. Но и это не все, предположим вы поддерживаете несколько языков в вашем приложении и вы не хотите чтобы юзеры с русской локалью видели текст на английском. Что же тогда надо сделать? Об этом ниже.

А пока давайте нажмем ОК и посмотрим что получилось.



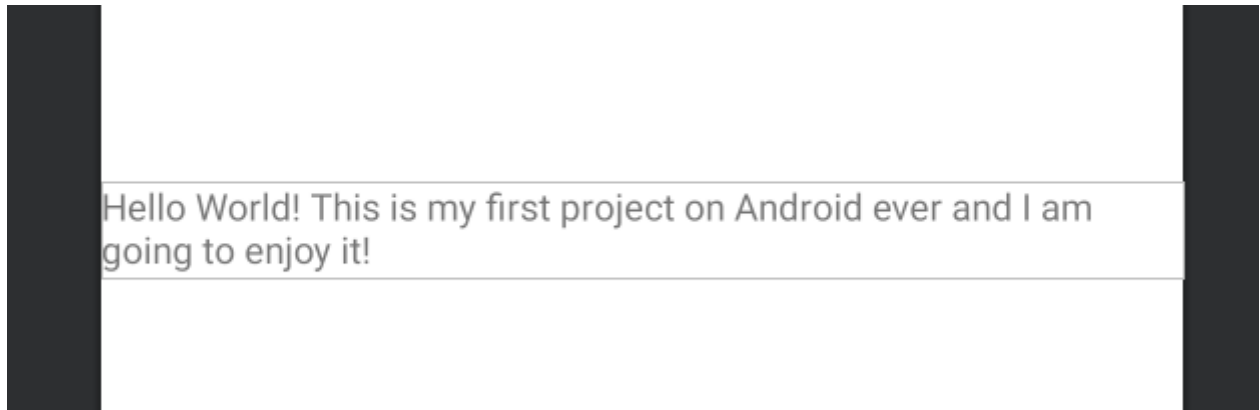
Вместо захардкоженного текста у нас теперь ссылка на константу в строковых ресурсах - `@string/hello`. Вы можете перейти в файл просто зажав Ctrl и кликнув по имени.

Кстати, `app_name` использовалось в манифесте, вы заметили? Если нет вернитесь и проверьте.



```
AndroidManifest.xml × activity_main.xml × strings.xml × MainActivity.kt ×
Edit translations for all locales in the translations editor.
1 <resources>
2   <string name="app_name">HelloWorld</string>
3   <string name="hello">Hello World! This is my first project on Android ever and I am going to enjoy it!</string>
4 </resources>
```

Теперь ваша длинная текстовка лежит в файле strings.xml. Давайте вернемся пока к дизайну.



Текстовка стала длинной и как только дошла до края перешла на следующую линию. И как видим по умолчанию оно не центрируется. Секунду, мы же написали центрирование через `layout_gravity`. Да, оно относилось именно к самой текстовке – она сама из себя представляет прямоугольник (спойлер как и все в андроид, даже круг) и по прежнему он находится по центру экрана. А вот содержимое... о нем не сказано ничего. Это легко исправить – есть атрибут `gravity`.

```
android:layout_height="wrap_content"
android:layout_gravity="center"
android:gravity="center"
android:text="@string/hello" />
```

Если вы начали писать слово `center` то заметили что АС предложило еще несколько вариантов

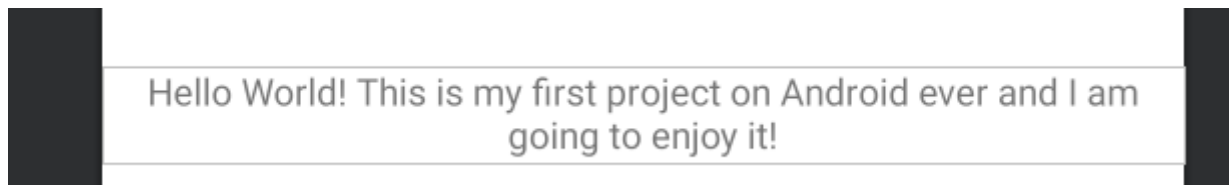
```
android:layout_gravity="center"
android:gravity="center"
android:text="@s-center
center_horizontal
center_vertical
center|bottom
center|center_horizontal
center|center_vertical
center|clip_horizontal
center|clip_vertical
center|end
center|fill
center|fill_horizontal
center|fill_vertical
center|left
center|right
center|start
center|top
Press Enter to insert, Tab to replace
```

Вы можете догадаться о сути некоторых, а если нет, то вы всегда можете найти информацию на офф.сайте [developer.android.com](https://developer.android.com) по всем атрибутам TextView и их значениям.

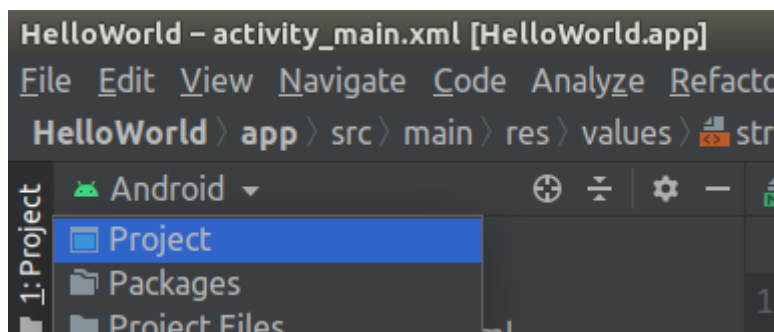
Ок, посмотрим же на наш дизайн. Теперь все ок!

И давайте вернемся к нашему вопросу с русским языком. Как же нам теперь сделать так, чтобы для тех девайсов где выбран русский язык для системы мы видели иное сообщение?

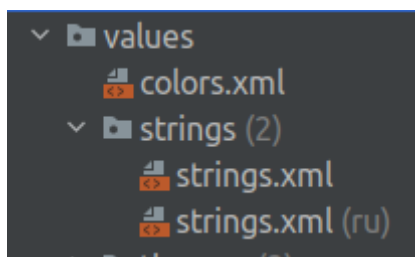
Нужно переопределить файл strings.xml, но как? Для этого создаем пакет с суффиксом -ru



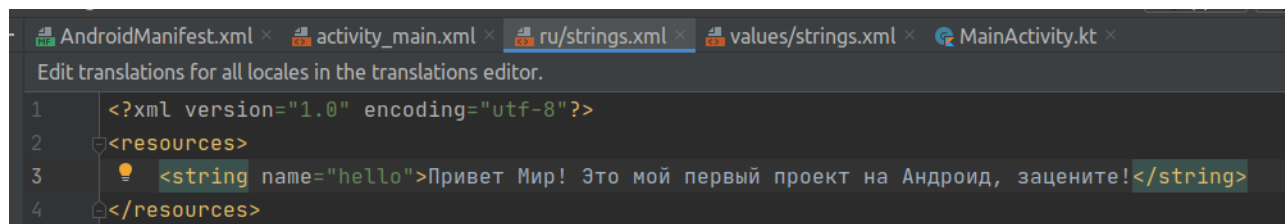
И чтобы это сделать надо переключиться на вид Project в окне дерева



Далее идем в пакет res и создаем новый файл через правый клик на пакете и выбрав New → Android Resource Directory и называем его values-ru. Далее в нем создаем New → Values Resource File и называем файл strings.xml. Возвращаемся в Android вид в дереве объектов и видим следующее

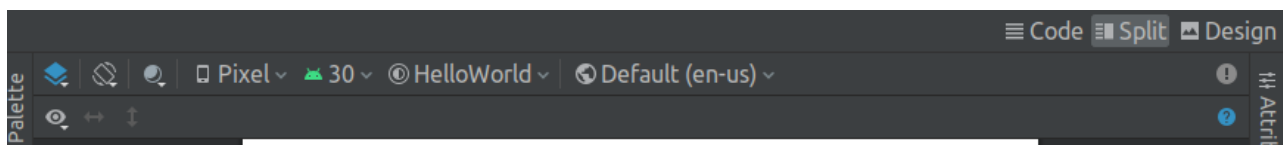


Вуаля! Все готово. Теперь надо скопировать данные из дефотльного файла и там уже написать по-русски.



Как видите в имени файла есть префикс ru/strings.xml это для того, чтобы различать с strings.xml. Вернемся теперь к нашему xml файлу для активити. Там все равно по-английски.

Почему же? А потому что по дефолту строки идут из дефолтного файла, в котором у нас английский. Чтобы посмотреть как будет выглядеть по-русски надо выбрать язык наверху.



Иконки слева направо – Вид, повернуть, темная тема, эмулятор, версия андроид, имя проекта и язык. Вот в нем надо выбрать русский



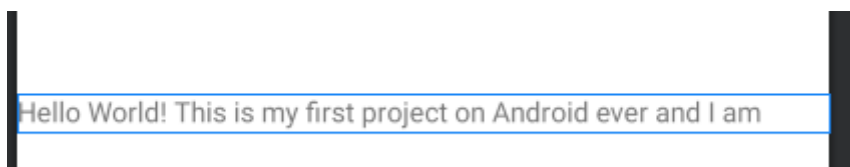
Вот и все. Так что запомните раз и навсегда – никогда не хардкодите строки в хмл файле layout. У вас может быть 1 язык и всегда одно значение для всех случаев жизни, но поверьте мне, читать длинный текст в верстке сложно. Выносите в ресурсы, для этого они и созданы. Плюс кстати вы можете получить доступ к этим константам из кода! Об этом ниже.

Попробуем запустить проект – у меня по дефолту английский в эмуляторе. Надо поменять как на телефоне и все будет ок. Что самое интересное – вам не нужно убивать приложение и перезапускать его – ваше приложение само видит когда язык системы поменялся и автоматом меняет текстовки. Попробуйте сами и увидите это.

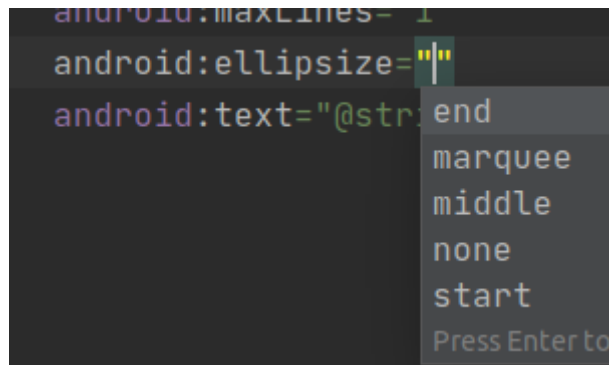
Ладно. Вернемся к английской версии. Как видите наш текст настолько длинный, что не влез в 1 линию. Зачастую мы верстаем такой экран, что нам нужно отобразить и другие вьюшки (видимые элементы) и нам приходит задание от дизайнера – если не вмещается в 1 линию, то не дать ему перейти на вторую линию и поставить просто многоточие. Как это делается в Андроид?

```
android:maxLengths="1"
```

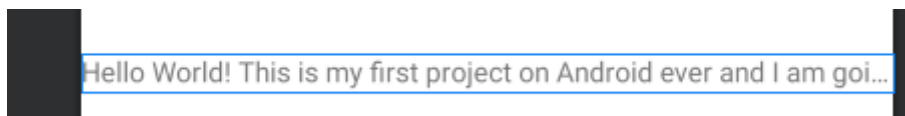
Мы можем прописать максимальное количество линий. Но нет, не так просто все. Посмотрите теперь на ваш текст – он обрывается не дойдя до конца. Давайте все же уберем gravity=center



Но три точки не появились. Что надо сделать? Есть такой атрибут как ellipsize

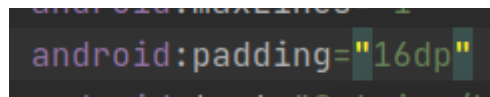


И можно указать где поставить три точки. В конце, в начале или по середине. Выберем в конце - end.

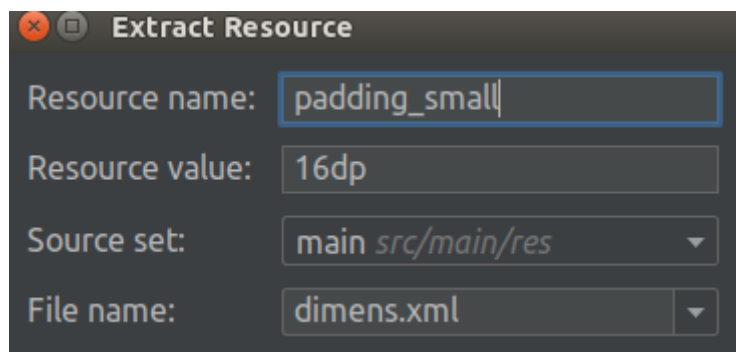


Вуаля! Сколько текста влезло столько и показывается, остальное три точки.

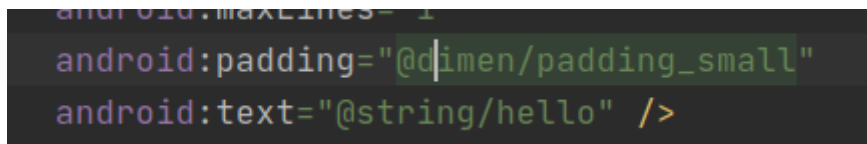
Вроде как все прекрасно, за исключением того, что теперь наш текст прям впрытык к краю экрана. Как же нам сделать отступ? Для этого есть простой атрибут padding



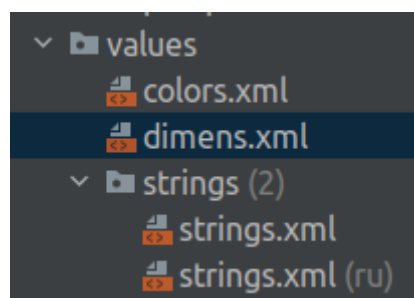
И это тоже хардкод! Не надо так делать. Если для строк есть strings.xml то для размерностей есть dimens.xml. Нажимаем на Alt+Enter и выбираем extract dimensions resource



И теперь у нас появился файл с размерами в дереве

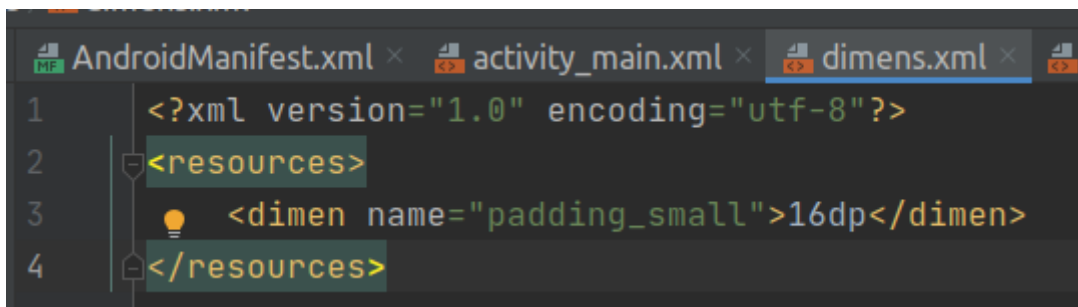


Конечно же у вас может быть разный отступ для разных конфигураций. Можете сделать отступ для больших экранов один, а для маленьких другой. Но об этом позже. А пока так.

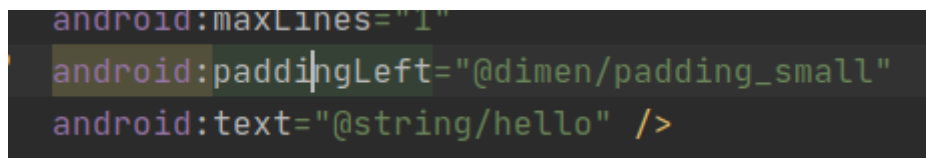


И да, вы всегда можете с легкостью изменить отступ если вам кажется что 16 слишком мало или много. Кстати говоря – запомните – правильные отступы всегда кратны 8! Или на крайний случай 4 и на самый крайний 2. Никогда не юзайте нечетные числа. У вас должна быть сетка с шагом 4, а лучше 8. Об этом можете почитать по ссылке [material.io](https://material.io).

Но вернемся к нашей текстовке. Кто-то может сказать – у нас и так мало было места, а мы еще уменьшили с помощью отступа. Как же так? Может сделать отступ только слева? Да, можно и так. Можно обозначить отступ не со всех сторон через padding, а указать сторону

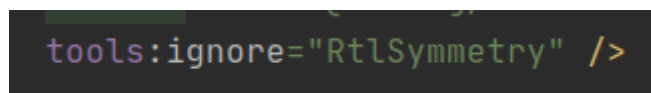


Но что-то не нравится АС! Что же не так с отступом слева? А то, что есть арабский мир...

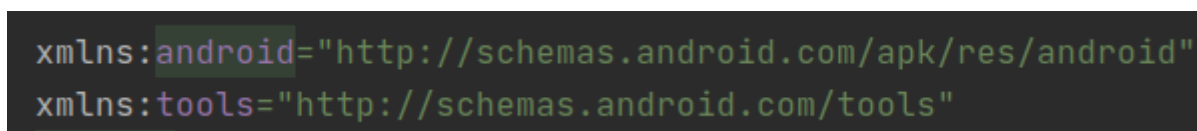


Не у всех в мире текст начинается слева. Для арабской локали начало справа и конец слева. Для этого лучше конечно послушать подсказку АС и поменять на paddingStart

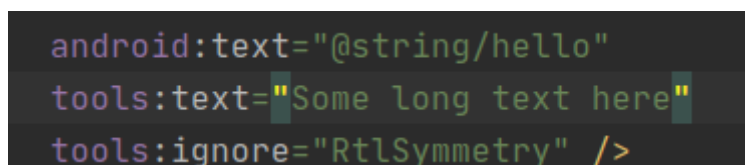
Но ему опять это не нравится! АС говорит что для симметрии надо указать не только отступ сначала, но и с конца. Вы можете забить на это с помощью ignore (но лучше конечно следовать требованиям АС).



Заметьте, что у нас появился новый xmlns (xml namespace) tools



Первое пространство имен дает доступ к атрибутам вью, а второе инструментарий. Что это такое? Мы можем показывать текст из strings.xml но для наглядности вбить свой текст, который будет отображен лишь на макете в превью в дизайне – т.е. никакого отношения к реальности не будет иметь – лишь для удобства. Смотрите



Харкодить tools:text можно! Оно ни на что не влияет.

Some long text here

Но кто-то скажет, а зачем нам не тот текст, который у нас уже есть? А все потому, что иногда у нас нет текста! Да, предположим что у вас текст не константой лежит в ресурсах, а получаем в коде! Вот и настал момент когда мы будем динамически ставить текст из кода!

Запомните – если у вас на экране всегда будет один и тот же текст и он известен, то кладите константу в ресурсы strings.xml, если же текст может меняться ввиду некоторых условий, то вы можете использовать `tools:text` для понимания как оно будет выглядеть, а в коде уже ставить текст финальный.

Но давайте еще пару вещей сделаем с текстом перед тем как ставить из кода. Мы до сих пор рассмотрели лишь разные языки, отступы и максимальное количество строк. А что насчет таких атрибутов как цвет, размер текста, шрифт? Это все есть! Но кто-то спросит – а надо все по отдельности прописывать? Нет, всегда есть готовые семплы как в Microsoft Word. Они ставятся через атрибут темы (или стиля). Для текста это `textAppearance`

```
android:textAppearance="@style/TextAppearance.AppCompat.Title"
```

Есть некоторое количество уже готовых шаблонов. Например титул. Выглядит вот так.

Some long text here

Но все равно, если вам нравится размер текста (кстати вы можете узнать сколько конкретно, для этого провалиться с зажатым `Ctrl` внутрь константы `Title` и там посмотреть у родителя (да, в стилях тоже есть наследование) вот скриншоты) то вам может не нравится цвет.

```
<style name="TextAppearance.AppCompat.Subhead.Inverse" parent="Base.TextAppearance.AppCompat.Subhead.Inverse"/>
<style name="TextAppearance.AppCompat.Title" parent="Base.TextAppearance.AppCompat.Title"/>
<style name="TextAppearance.AppCompat.Title.Inverse" parent="Base.TextAppearance.AppCompat.Title"/>
```

...

```
<style name="Base.TextAppearance.AppCompat.Title">
    <item name="android:textSize">@dimen/abc_text_size_title_material</item>
    <item name="android:textColor">?android:textColorPrimary</item>
</style>
```

Как видите сам гугл тоже использует `dimens` для размеров текста и здесь `20sp`.

```
<dimen name="abc_text_size_title_material">20sp</dimen>
```

Вы заметили что для текста не `dp`, а `sp`? Потому что текст может быть изменен в размерах из настроек для слабовидящих.

Цвет у нас `textColorPrimary` т.е. черный. Давайте изменим его. Для этого надо написать атрибут цвета типа

```

13 android:textColor="#FF0000"
14 android:textAppearance="@style/TextAppearance.AppCompat.Title"

```

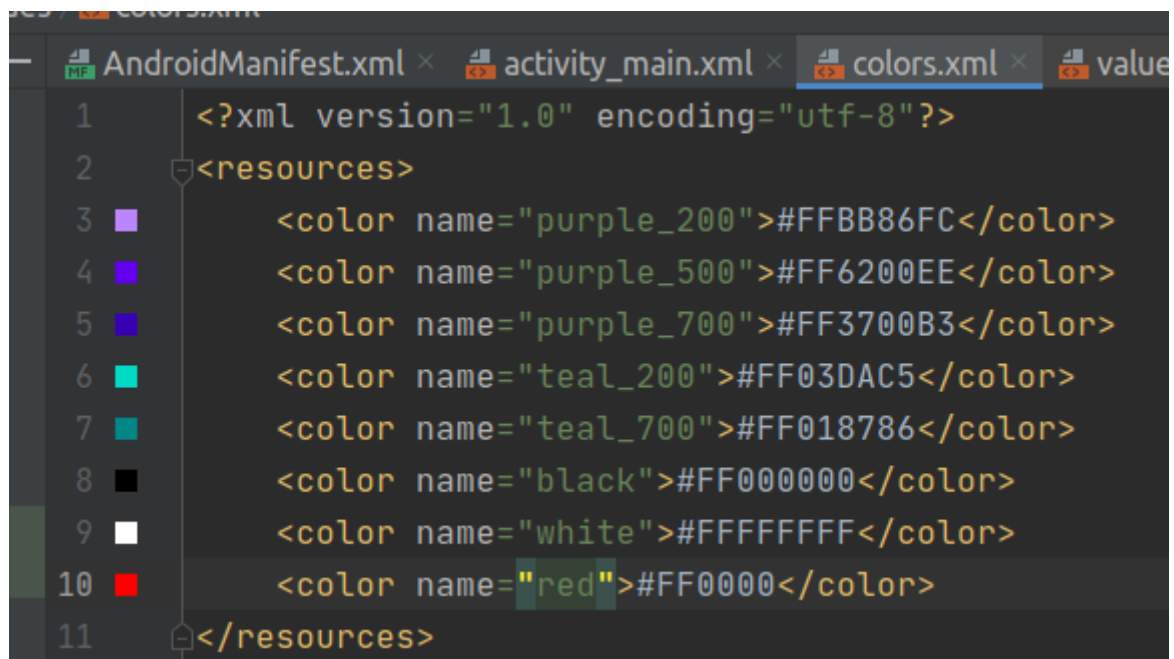
И это тоже хардкод и так нельзя делать! Надо выделить цвет в ресурсы. Alt+Enter

```

13 android:textColor="@color/red"

```

В нашем файлике colors.xml будет храниться цвет красный



```

1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3     <color name="purple_200">#FFBB86FC</color>
4     <color name="purple_500">#FF6200EE</color>
5     <color name="purple_700">#FF3700B3</color>
6     <color name="teal_200">#FF03DAC5</color>
7     <color name="teal_700">#FF018786</color>
8     <color name="black">#FF000000</color>
9     <color name="white">#FFFFFFFF</color>
10    <color name="red">#FF0000</color>
11 </resources>

```

Как видите здесь уже что-то есть. Но вы всегда можете поменять оттенок красного если ваш дизайнер или вы сами решите что вам надо светлее или полупрозрачное. И да, если вы не знаете, то это Hex формат для цвета. Есть еще RGB но в андроид если не в коде, то пишем hex.

Ладно, что же у нас получилось? Мы использовали тему тайтла, но нам не нравился цвет. Как же нам сделать так, чтобы во всем проекте все тайтлы были красного цвета? Не хотелось бы писать 1 лишнюю линию в xml файле для каждого текста. И здесь мы знакомимся с темами. Были цвета, размеры, строки и вот теперь тема (styles.xml). Для этого наследуемся от тайтла гугла и переопределим цвет на красный и все будет работать в 1 линию.

```

<style name="RedTile" parent="TextAppearance.AppCompat.Title">
    <item name="android:textColor">@color/red</item>
</style>

```

Теперь заменим 2 линии в нашем xml на это.

```

tools:text="Some long text here"
android:textAppearance="@style/RedTile"
tools:ignore="RtlSymmetry" />

```

Вуаля! Теперь у вас есть тема для заголовков. В нем можно менять что угодно и во всем проекте все заголовки автоматически будут изменены. Не надо редактировать 100500 файлов.

```
AndroidManifest.xml × activity_main.xml × colors.xml × themes.xml × values.xml × dimensions.xml × ru/strings.xml ×
1 <resources xmlns:tools="http://schemas.android.com/tools">
2     <!-- Base application theme. -->
3     <style name="Theme.HelloWorld" parent="Theme.MaterialComponents.DayNight.DarkActionBar">
4         <!-- Primary brand color. -->
5         <item name="colorPrimary">@color/purple_500</item>
6         <item name="colorPrimaryVariant">@color/purple_700</item>
7         <item name="colorOnPrimary">@color/white</item>
8         <!-- Secondary brand color. -->
9         <item name="colorSecondary">@color/teal_200</item>
10        <item name="colorSecondaryVariant">@color/teal_700</item>
11        <item name="colorOnSecondary">@color/black</item>
12        <!-- Status bar color. -->
13        <item name="android:statusBarColor" tools:targetApi="l">?attr/colorPrimaryVariant</item>
14        <!-- Customize your theme here. -->
15    </style>
16
17    <style name="RedTitle" parent="TextAppearance.AppCompat.Title">
18        <item name="android:textColor">@color/red</item>
19    </style>
20
21 </resources>
```

Можете запустить проект и проверить.

Ладно. Еще одно расскажу про текстовки. Предположим нам нужно отобразить html на экране. Типа - `<HTML><TITLE>Title</TITLE></HTML>`. И что такого спросите меня вы. А то, что в наших строковых ресурсах будет конфликт!

```
<string name="html"><HTML><TITLE>Title</TITLE></HTML></string>
resources>
```

Во-первых проблема в том, что у нас 2 языка и мы для русского не переопределили константу. Можно сделать ее непереопределимой – `Alt+Enter`

```
<string name="html" translatable="false"><HTML><TITLE>Title</TITLE></HTML></string>
```

Но будет ли оно отображено корректно на экране? Ведь сам файл строковых ресурсов является xml файлом.

Title

Как видите что-то пошло не так. Ведь мы хотели видеть символы `</>` на экране мобилки. Для этого есть такая штука как CDATA



```
<string name="html" translatable="false"><![CDATA[<HTML><TITLE>Title</TITLE></HTML>]]></string>
```

Как видите теперь у нас не будет проблем с отображением. Проверим.



Так что запомните, если в вашем тексте есть спец.символы, которые конфликтуют с xml структурой, то юзайте CDATA. Для других случаев используйте экранирование. Если вы хотите действительно загрузить html то в коде нужно сделать через `Html.fromHtml("<HTML><TITLE> и так далее")`.

Ладно, давайте все же дойдем до установки значения тексту через код.

Как это сделать? Если помните то наш мейн активити котлин класс знает про нашу разметку так как мы установили контент через `setContentView(R.layout.activity_main)` Кстати, R класс это тот сгенерированный который и лежит в пакете `res/generated` Мы обращаемся к нему как будто он есть уже, но он генерируется когда мы нажмем на build project. Подробнее ниже.

Итак, у нас есть котлин класс, а есть xml разметка. Они связаны, да. Но как нам получить доступ к текстовойвью(далее текствью)? Ладно, у нас сейчас 1 текстовка на экране, а завтра если будет 2, то нам надо как-то их различать. Для этого нужен некий уникальный идентификатор – id. Пишем айдишник в xml.

```
2 <TextView
3     xmlns:android="http://schemas.android.com/apk/res/android"
4     xmlns:tools="http://schemas.android.com/tools"
5     android:id="@+id/titleTextView"
```

Обратите внимание `@+id` означает добавить айди. Куда его добавить? Правильно, в R класс. Ведь он генерируется и прямо сейчас там нет этого айдишника. Делайте ваши айди уникальными и вы никогда не запутаетесь в них. Хорошо, сделали, что дальше? Нам надо найти объект текствью в коде. И конечно же чтобы однозначно это сделать нам и нужен был айди. Идем в код.

```
setContentView(R.layout.activity_main)

val titleTextView = findViewById<TextView>(R.id.titleTextView)
```

Так как у нас котлин и мы не написали тип переменной, то должны указать его в методе `findViewById<T>` чтобы мы в коде смогли использовать методы именно класса `TextView`. Можно иначе конечно, но это уже к вопросам языка программирования. Ладно. Что дальше?

```
val titleTextView: TextView = findViewById(R.id.titleTextView)
```

Все те атрибуты, которые мы писали в xml файле доступны и в котлин коде!

Теперь мы можем задать текст с помощью кода и тогда все что было написано в xml файле будет неважно. Потому что изначально мы смотрим на xml файл и только потом на код. Значит в коде если мы переписали значение текста, то оно и будет отображено. Проверим!

```
val titleTextView: TextView = findViewById(R.id.titleTextView)
titleTextView.text = "This is title from code!"
```

И да, АС ругается потому как мы используем литерал прямо в методе сеттера. Можно хакнуть это так – написать переменную и юзать ее

```
val titleTextView: TextView = findViewById(R.id.titleTextView)
val string = "This is title from code!"
titleTextView.text = string
```

Теперь АС молчит ибо она не знает что мы использовали константу. Это и не так, так как наша переменная может инициализироваться через if например. Ладно. Запустим же код!



Вуаля! Конечно же стиль текста остался таким каким он был в xml. Мы же переписали лишь значение текста, а не цвет или стиль. Но вы можете в коде поменять и цвет. Это делается через методы аналогичные атрибуту.

```
titleTextView.text = string
titleTextView.setTextColor(Color.parseColor("colorString: "#FFFFFF"))
```

И если вы запустите проект, то не увидите текстовку. Потому что #FFFFFF это белый цвет! И белое на белом не видно. Давайте поставим черный цвет из ресурсов. Как это делается? Сложно...

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
    titleTextView.setTextColor(resources.getColor(R.color.black, theme: null))
} else {
    titleTextView.setTextColor(resources.getColor(R.color.black))
}
```

Добро пожаловать в мир Андроид! Апи (сдк) меняется с каждой версией и некоторые методы становятся Deprecated. Как видите для этого мы пишем if else. Если текущая версия андроид выше Marshmellow т.е. 6, мы используем новый метод, если ниже 6, то старый метод, который и был Deprecated(устаревший, более не используется в новых версиях). Именно поэтому в начале создания проекта и нужно было выбирать версию минимальную выше. Если бы выбрали min api 23, а не 21, то нам бы не надо было писать такой код с ветвлением. Но тогда бы вы потеряли юзеров которые на андроид 5. Но это другой вопрос. Вы всегда можете написать экстеншн функцию для этого.

```
8 titleTextView.text = string
9 titleTextView.setColor(R.color.black)
10 }
11 }
12
13 fun TextView.setColor(@ColorRes colorResId: Int, theme: Resources.Theme? = null) {
14     if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M)
15         setTextColor(resources.getColor(colorResId, theme))
16     else
17         setTextColor(resources.getColor(colorResId))
18 }
```

Но это все лирика конечно. Ваш проект будет полон экстеншнами чтобы скрыть такие моменты. Это нормально. Но лучше конечно же если у вас цвет вбит прямо в xml.

И возвращаясь к RGB – `setTextColor(Color.rgb(255,255,255))` можно использовать этот метод.

Ладно, мы затронули тему видимости. А что если нам действительно нужно сделать текст невидимым? Для этого есть атрибут в xml и конечно же кодом тоже можно сделать.

```
android:maxLines="1"
android:visibility=""
android:paddingStart="gone"
android:text="@string/invisible"
android:textAppearance="visible"
Press Enter to
```

Три значения – gone если вам нужно не только сделать невидимым, но и убрать вообще текстовку будто ее нет вообще на экране (не занимает пространство), invisible – невидно, но занимает место на экране. Visible – видно. Можете через `tools:visibility` сделать видимость одну для проверки макета, а через `android:visibility` целевую видимость. Очень удобно.

В коде же будет просто через константы. Можете запустить проект и проверить.

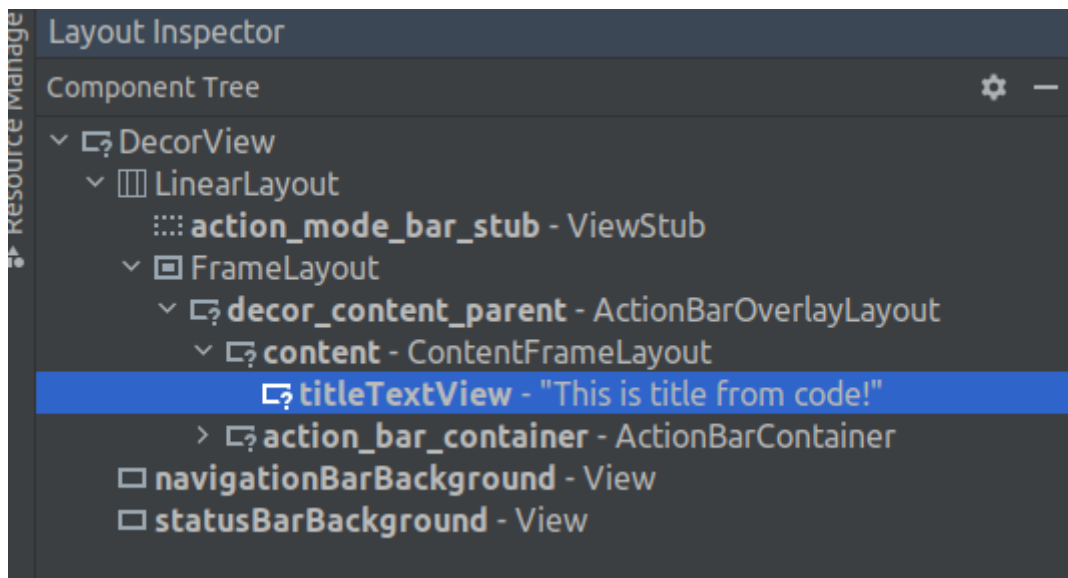
```
titleTextView.setColor(R.color.black)
titleTextView.visibility = View.GONE
```

Чтобы доказать наличие текста на экране можем воспользоваться layout inspector-ом. Для этого давайте поставим видимость `View.INVISIBLE` и чекнем инспектором что view есть.

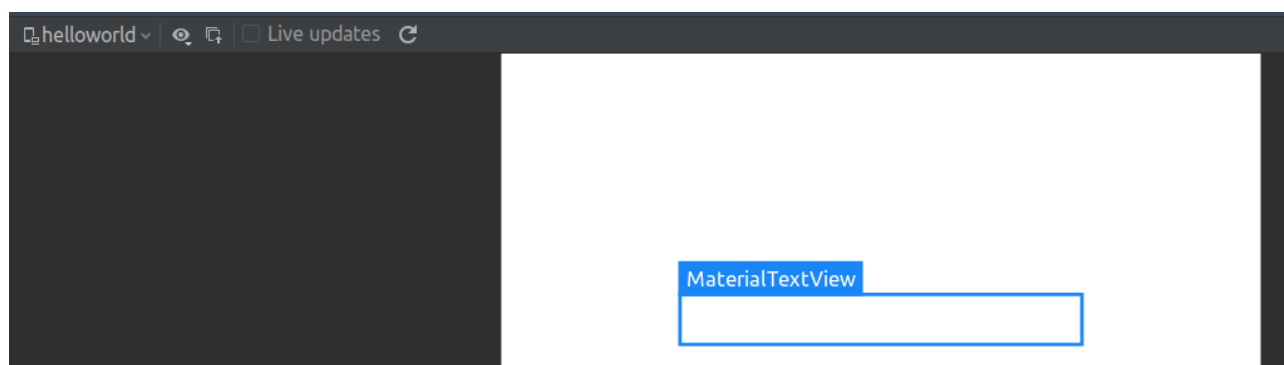
Он находится во вкладке Tools (Там где File, View, Edit...Tools, Help). Layout Inspector

Заметьте, что у нас выбран процесс HelloWorld в левом верхнем углу.

Кстати, этот инструмент очень хорош для проверки контента на экране. Мы не раз будем использовать его, так что постарайтесь уже сейчас понять как его запускать и смотреть контент экрана. Кстати, заметьте, что написано Component Tree – дерево компонентов. Все наши вью находятся в дереве. Сейчас не нужно пытаться понять что есть что, просто запомните это. На экране есть навигационный бар, статус бар, тулбар и другие штуки.



Live updates – можно поставить галку если меняется контент на экране.



Расскажу еще одну немаловажную штуку про текстовки. Иногда (очень часто) нам надо написать нечто типа – Нажимая на кнопку вы подтверждаете что ознакомились с условиями использования и политикой конфиденциальности и нужно сделать части текста кликабельными. Как это делать? В коде конечно же. В хмл никак не получится. Начнем с того, что напишем в строковых ресурсах 3 текстовки – почему 3 и что за 3 если у нас 1? на самом деле 6. У нас же еще и русский и английский. В первой текстовке полный текст, во второй первый кликабельный текст, в третьей второй кликабельный. Погнали! (Да, андроид это интересно и увлекательно!).

```
<string name="agreement_full_text">By pressing next button you confirm the agreement
    and privacy policy</string>
<string name="confidential_info">agreement</string>
<string name="privacy_policy">privacy policy</string>
```

Сначала 3 строки на английском в файле strings.xml

Потом 3 строки на русском в файле ru/strings.xml

```
<string name="agreement_full_text">Нажимая на кнопку вы подтверждаете что ознакомились с
    условиями использования и политикой конфиденциальности</string>
<string name="confidential_info">условиями использования</string>
<string name="privacy_policy">политикой конфиденциальности</string>
```

В xml можно ничего не сетить вообще (android:text убрать можно)

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <TextView xmlns:android="http://schemas.android.com/apk/res/android"
3      android:id="@+id/agreementTextView"
4      android:layout_width="wrap_content"
5      android:layout_height="wrap_content"
6      android:padding="@dimen/padding_small"/>
```

Мы только поменяем айди чтобы соответствовало логике. Итак, код!

```
val agreementTextView: TextView = findViewById(R.id.agreementTextView)

val fullText = getString(R.string.agreement_full_text)
val confidential = getString(R.string.confidential_info)
val policy = getString(R.string.privacy_policy)
val spannableString = SpannableString(fullText)
```

Сначала инициализируем переменную которая связана с xml. После чего получаем строки в коде. Так как они лежат у нас в разных файлах и в зависимости от языка они будут разными мы получаем их через getString чтобы обращаться с ними как с String. Ведь сам R.string.agreement\_full\_text является int айди строки. Но кстати это не просто число, а с аннотацией @StringRes

```
@NonNull
public final String getString( @StringRes @StringRes int resId) {
    return getResources().getString(resId);
}
```

Помните я говорил что будем читать джавакод? Ну вот. Чтобы метод не получал любое число типа 182, в андроид есть аннотация @StringRes которая сразу проверит что вы передали в метод айди строкового ресурса.

Ладно, после инициализации текста и кусков текста мы создаем SpannableString из полного текста. Что это такое?

```
/**
 * This is the class for text whose content is immutable but to which
 * markup objects can be attached and detached.
 * For mutable text, see {@link SpannableStringBuilder}.
 */
public class SpannableString
    extends SpannableStringInternal
    implements CharSequence, GetChars, Spannable
```

Читаем джавадок класса – класс для текста но с markup objects – объектами, которые например можно кликать в нашем случае. Ок. Что дальше? Нам нужен сам объект кликабельного куска текста. Как его создать? Смотрим

```
val confidentialClickable = object : ClickableSpan() {
    override fun onClick(widget: View) {
        Snackbar.make(widget, text: "Go to link1", Snackbar.LENGTH_SHORT).show()
    }

    override fun updateDrawState(ds: TextPaint) {
        super.updateDrawState(ds)
        ds.isUnderlineText = true
        ds.color = Color.parseColor(colorString: "#FF0000")
    }
}
```

Есть класс ClickableSpan, он абстрактный и потому в котлин мы пишем object который наследуется от него и тем самым создаем экземпляр анонимного класса. У него 2 метода – onClick – что делать когда на текст кликнули (да, в андроиде используем понятие клика, а не тапа, и оно отличается от тапа тем, что клик это 2 события – когда палец коснулся экрана смартфона и поднялся). Мы также переопределили второй метод где делаем область текста подчеркнутой через объект TextPaint. Можете сами посмотреть какие у него еще методы есть.

Также меняем цвет участка текста. Snackbar это класс для уведомлений внутри экрана (не те уведомления о которых вы подумали, которые падают сверху). То же самое сделаем и для второго участка кода. Да, здесь грубое нарушение того, что мы захардкодили цвет красный.

```
val policyClickable = object : ClickableSpan() {
    override fun onClick(widget: View) {
        Snackbar.make(widget, text: "Go to link2", Snackbar.LENGTH_SHORT).show()
    }

    override fun updateDrawState(ds: TextPaint) {
        super.updateDrawState(ds)
        ds.isUnderlineText = true
        ds.color = Color.parseColor(colorString: "#FF0000")
    }
}
```

На самом деле можно было бы сделать иначе, раз у нас 2 одинаковых участка и отличаются лишь тем, куда ведут их ссылки. В конце я перепису этот код. Итак, создали классы для нажатия на текст. А как определить сами участки?

```
spannableString.setSpan(  
    confidentialClickable,  
    fullText.indexOf(confidential),  
    end: fullText.indexOf(confidential) + confidential.length,  
    Spanned.SPAN_EXCLUSIVE_EXCLUSIVE  
)
```

Сетим спан – кликабельный объект, начало, конец и метод – исключительно исключительный. Можете сами провалиться в константу и прочитать джавадок – Ctrl клик.

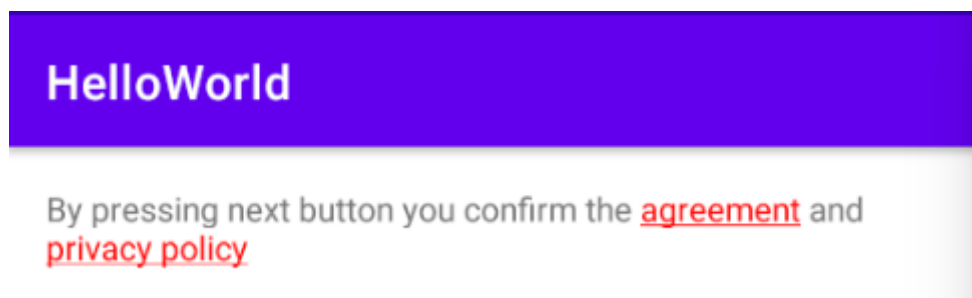
Кстати – indexOf вернет индекс участка текста в полном тексте и для определения конца просто плюсанем длину участка. Вот и все. Аналогично для второго

```
spannableString.setSpan(  
    policyClickable,  
    fullText.indexOf(policy),  
    end: fullText.indexOf(policy) + policy.length,  
    Spanned.SPAN_EXCLUSIVE_EXCLUSIVE  
)
```

И последнее действие – применяем к текствью. Также делаем прозрачным момент когда нажали (можете выбрать другой цвет и увидите в момент нажатия какая подсветка)

```
agreementTextView.run { this: TextView  
    text = spannableString  
    movementMethod = LinkMovementMethod.getInstance()  
    highlightColor = Color.TRANSPARENT  
}
```

Давайте запустим и проверим!



Участки подчеркнуты и они красные. Нажмем и увидим снейкбар





Как и обещал немного порефакторим код

```
class MyClickableSpan(private val lambda: (view: View) -> Unit) : ClickableSpan() {
    override fun onClick(widget: View) = lambda.invoke(widget)

    override fun updateDrawState(ds: TextPaint) {
        super.updateDrawState(ds)
        ds.isUnderlineText = true
        ds.color = Color.parseColor("colorString: "#FF0000")
    }
}
```

И сразу все стало прекраснее – заметьте как изящно работают лямбды.

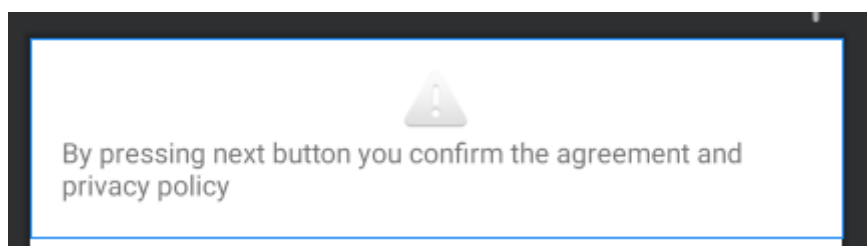
```
val confidentialClickable = MyClickableSpan { it: View
    Snackbar.make(it, text: "Go to link1", Snackbar.LENGTH_SHORT).show()
}
val policyClickable = MyClickableSpan { it: View
    Snackbar.make(it, text: "Go to link2", Snackbar.LENGTH_SHORT).show()
}
```

В следующих лекциях мы рассмотрим открытие браузера с ссылкой и заменим снейкбар.

Ну и напоследок добавим к этому тексту изображение сверху. Для этого в хмл добавим 1 линию

```
8      tools:text="@string/agreement_full_text"
9      android:padding="@dimen/padding_small"
10     app:drawableTopCompat="@android:drawable/ic_dialog_alert" />
```

Да, в андроид есть свои изображения и можно использовать их (хотя не рекомендуется конечно). Как видите у нас белый фон у экрана и на белом фоне белая иконка не сильно видна будет

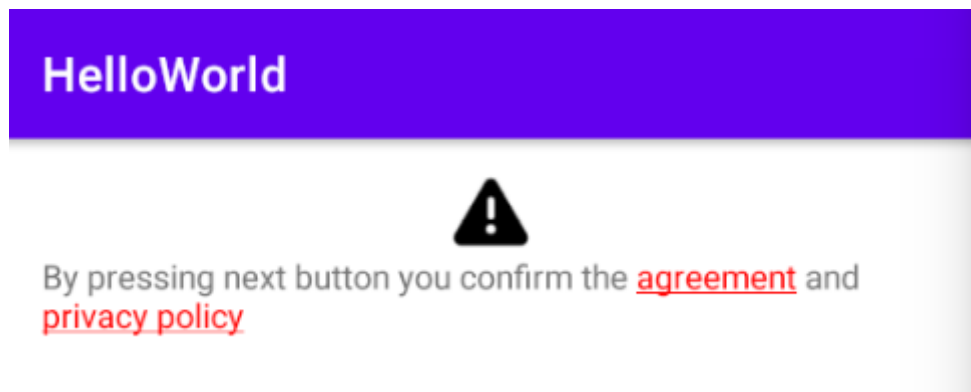




Что же делать? Менять фон всего экрана? Или фон текста? Можно конечно. Но есть решение лучше – поменять цвет иконки. Но ведь иконка в формате png как мы меняем ее цвет? А вот так

```
10 app:drawableTint="@color/black"
11 app:drawableTopCompat="@android:drawable/ic_dialog_alert" />
```

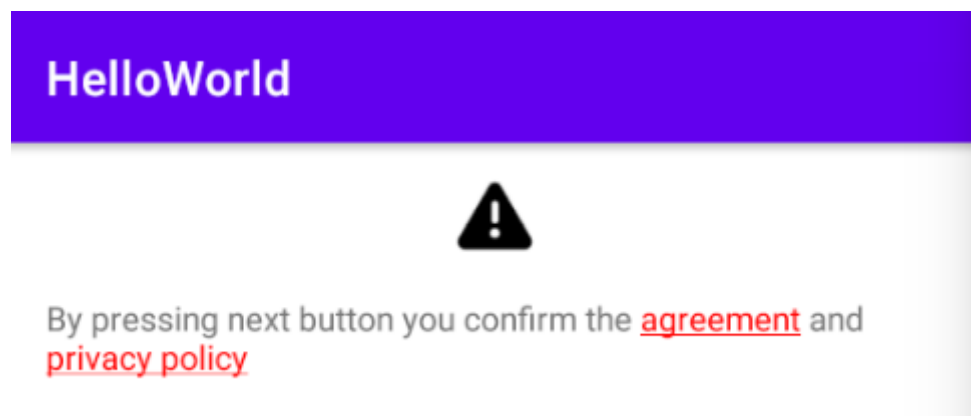
Есть такая штука как tint и оно может поменять цвет вашей иконки. Да, вам не надо в проекте иметь 100500 иконок всех цветов на все случаи жизни. Можно иметь 1 иконку белую или черную и заменять цвет таким образом. Подробно про иконки и картинки мы поговорим в следующей лекции. А пока в рамках лекции про текстовки вот вам столько информации.



Но вам не кажется, что изображение слишком близко к тексту? Да, но мы уже сделали отступ у текста, можно ли сделать отступ именно у картинки? Да. Смотрите как

```
10 app:drawableTint="@color/black"
11 android:drawablePadding="@dimen/padding_small"
12 app:drawableTopCompat="@android:drawable/ic_dialog_alert" />
```

И можно все же поменять цвет фона на последок у текстовки, например на серый (дизайнеры любят оттенки серого).



Ставим серый фон картинке

```

0  ■      <color name="red">#FF0000</color>
1  ■      <color name="gray">#A9A9A9</color>
2  ■      </resources>

```

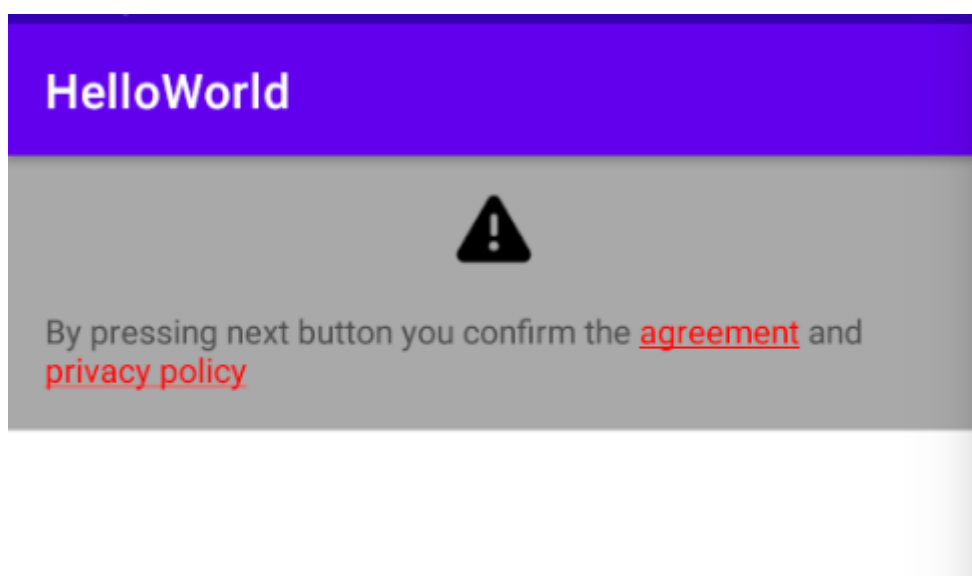
Возможно не самый удачный цвет, но в идеале вы будете получать значения от дизайнера

```

10  ■      app:drawableTint="@color/black"
11  ■      android:drawablePadding="@dimen/padding_small"
12  ■      android:background="@color/gray"
13  ■      app:drawableTopCompat="@android:drawable/ic_dialog_alert" />

```

Посмотрим как выглядит



Как видите серый фон стал только у картинки вместе с иконкой, дальше внизу там белый фон. Почему же так? А потому то наша текстовка заполняет собой не весь экран, а исходя из содержимого. Это прописано в атрибутах высоты и ширины. Смотрите сами

```

android:layout_width="wrap_content"
android:layout_height="wrap_content"

```

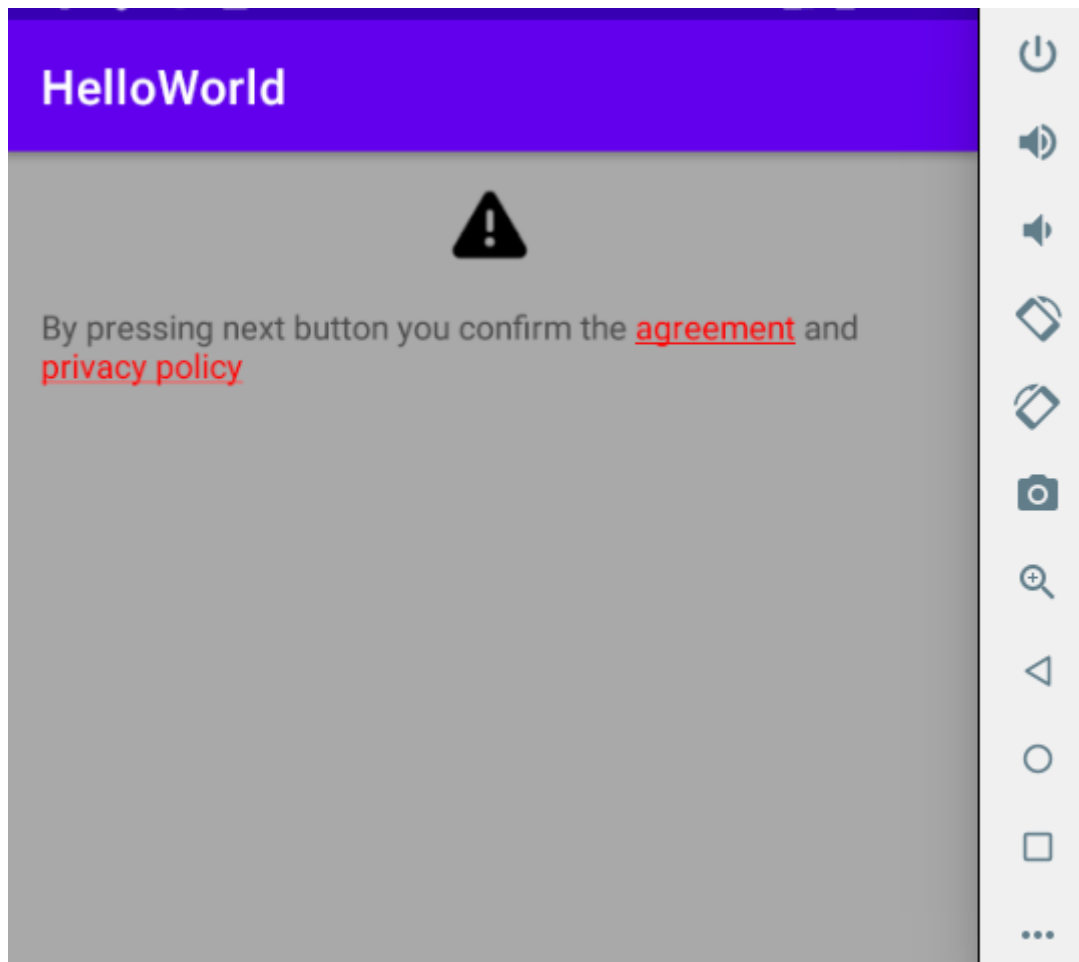
Если мы хотим заполнить текстом все пространство надо поменять высоту с wrap\_content (по содержимому) на match\_parent – совпадает с родителем. А в нашем случае родитель это весь экран.

```

android:layout_width="match_parent"
android:layout_height="match_parent"

```

Мы бы могли не менять ширину, так как текст у нас довольно длинный. Но это так только для мобилок, а если у нас планшет в ориентации альбомной? Поэтому перестраховуемся и вот:

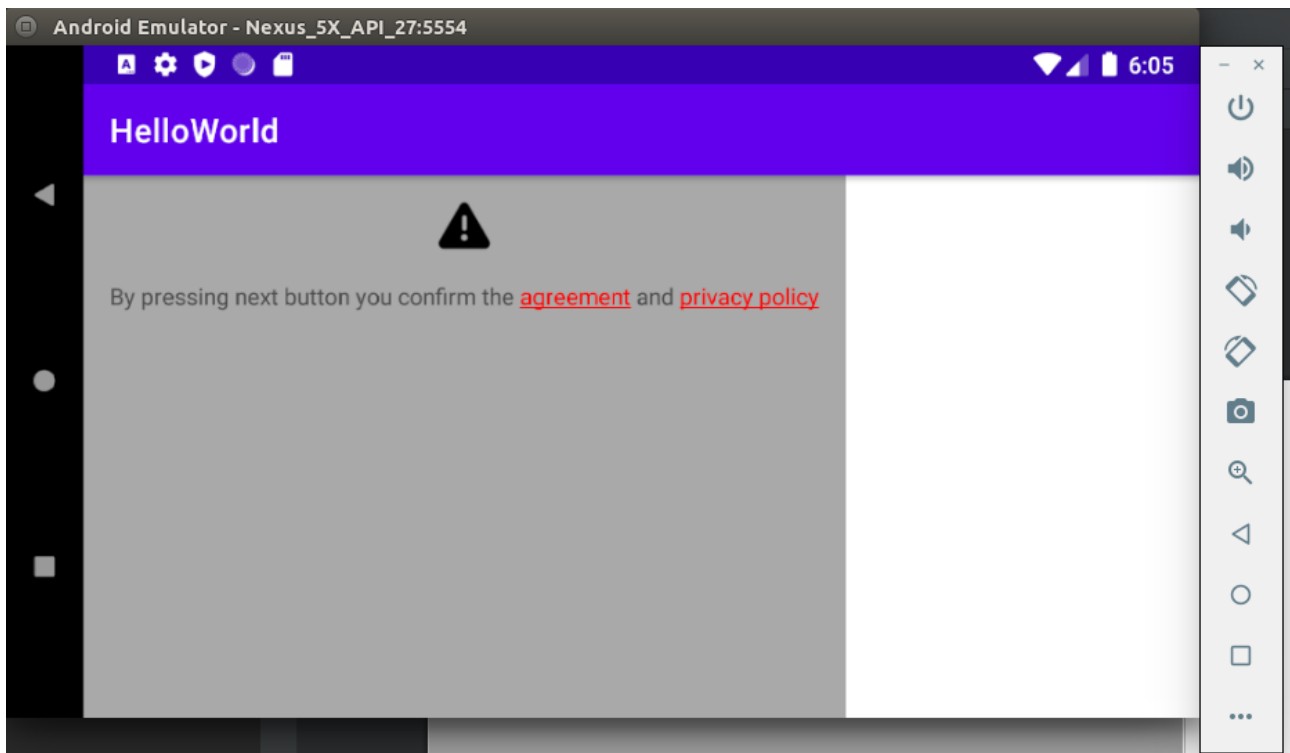


Кстати вы можете проверить ширину повернув экран эмулятора. Поэтому всегда проверяйте свою разметку на больших девайсах во всех ориентациях. Или проще просто представить как будет все выглядеть исходя из значений атрибутов. Ну и можно запретить поворот экрана, но это не тема текущей лекции.

```
android:layout_width="wrap_content"  
android:layout_height="match_parent"
```

В данной лекции я постарался вам рассказать про все самые частые случаи использования текстовок. В следующей лекции рассмотрим изображения и иконки отдельно от текстовок.

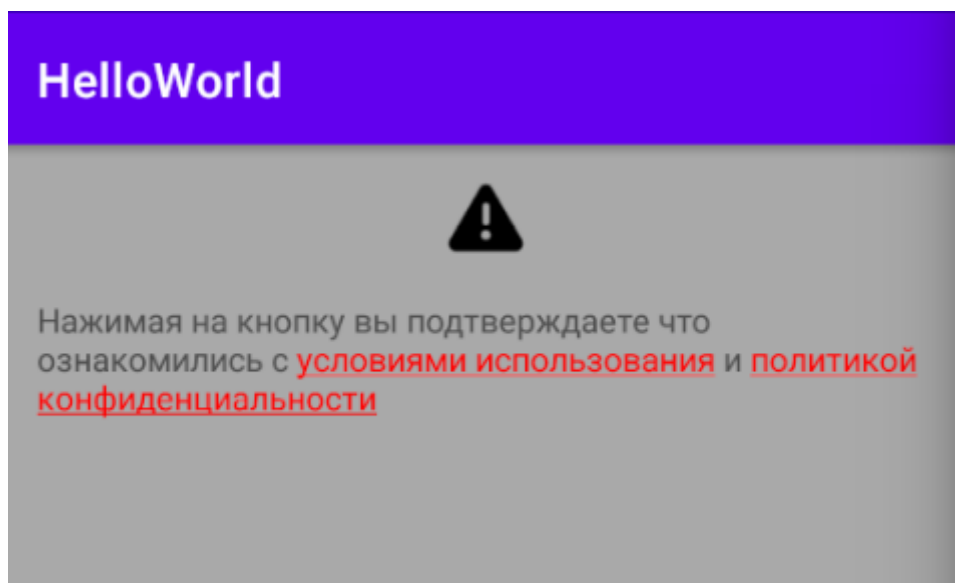
Для того чтобы закрепить материал напишите такой проект (описание на следующей странице) :



Напишите игру текстовый квест. Для этого используйте ClickableSpan и меняйте программно текст на экране исходя из выбранного. Также можете менять изображение шага через код(загуглите сами как это сделать). Для разнообразия можете менять фон и цвет текста.

Если вам сложно придумать свой текстовый квест можете использовать готовые из сети. Или же если вы решили подобную задачу в джава части, то просто переиспользуйте тот код.

п.с. вот скрин как выглядит на русской локали



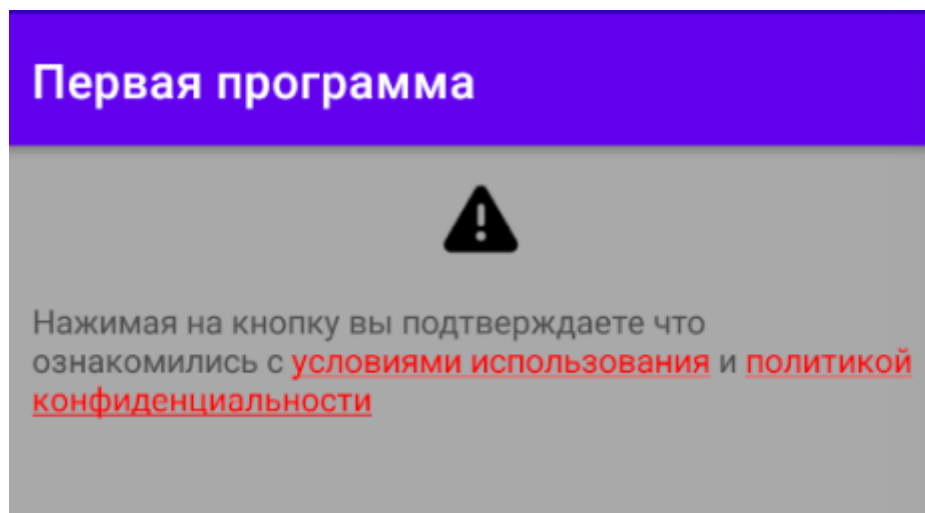
Как видите в шапке имя проекта, мы не переопределили для русской локали в файле ru/strings.xml и потому видим HelloWorld. Можете переопределить и увидите что поменялось.

Key	Resource Folder	Untranslatable	Default Value	Russian (ru)
app_name	app/src/main/res	<input type="checkbox"/>	HelloWorld	Первая программа
hello	app/src/main/res	<input type="checkbox"/>	Hello World! This is my f	Привет Мир! Это мой п
html	app/src/main/res	<input checked="" type="checkbox"/>	<![CDATA[<HTML><TIT	
agreement_full_text	app/src/main/res	<input type="checkbox"/>	By pressing next button	Нажимая на кнопку вы
confidential_info	app/src/main/res	<input type="checkbox"/>	agreement	условиями использо
privacy_policy	app/src/main/res	<input type="checkbox"/>	privacy policy	политикой конфиденц

Alt+Enter на линии

```
<string name="app_name">HelloWorld</string>
```

и выбрать open editor и можно вбивать все значения сразу в одном окошке



Вот и все.