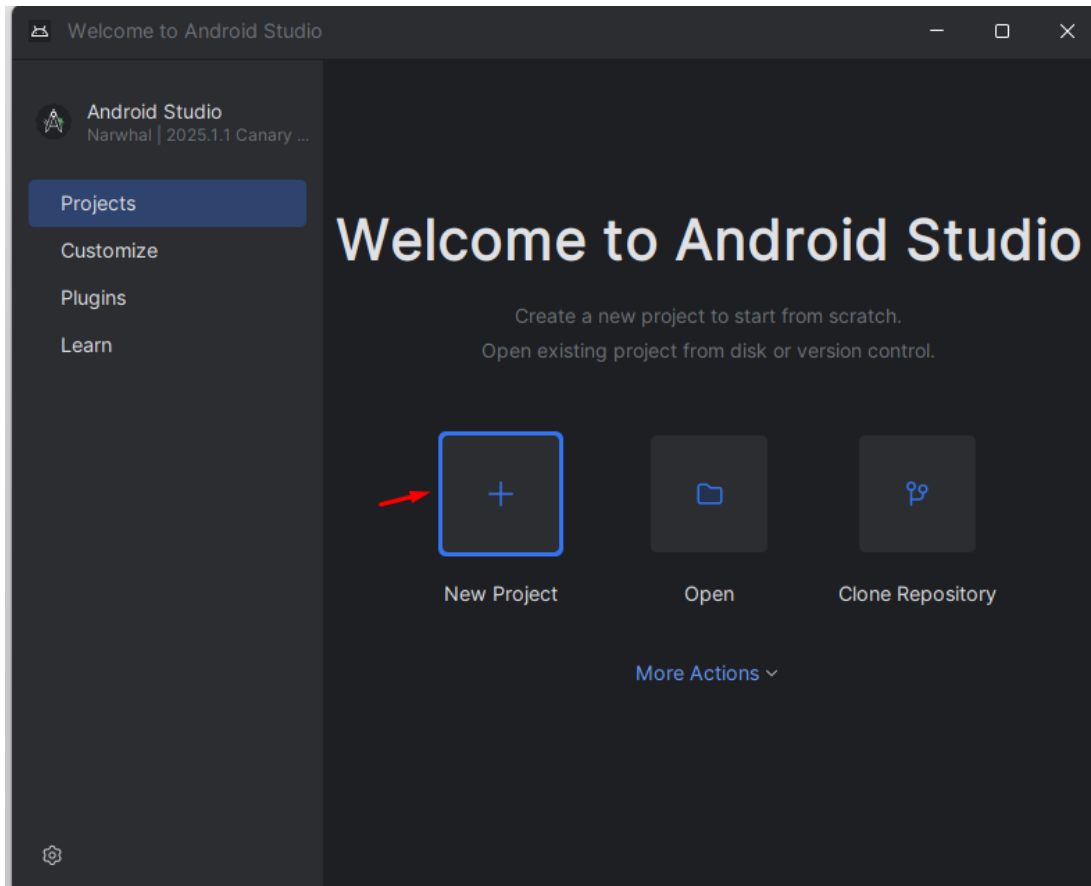


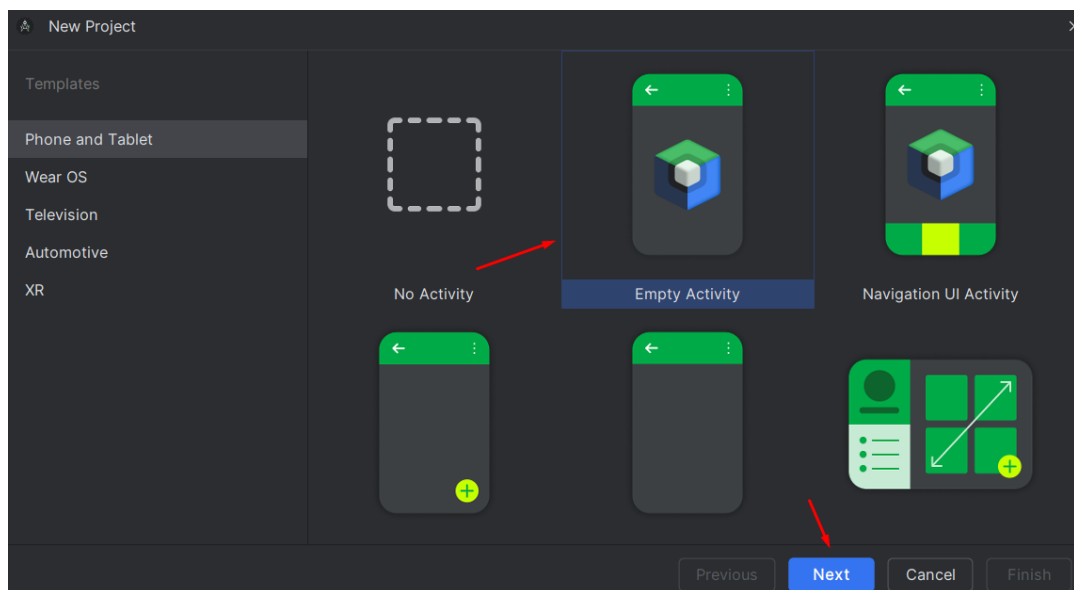
Лабораторная работа №1. Создание первого приложения в Android Studio

Шаг 1. Запуск и настройка проекта в Android Studio

Запустите **Android Studio**. В диалоговом окне **Welcome to Android Studio** нажмите **New Project**:



Выберите шаблон **Empty Activity** и нажмите **Next**:



Empty Activity — это базовый шаблон для создания проекта, в котором используется один экран. Он идеально подходит для старта и поддерживает **Jetpack Compose** — современный способ создания пользовательского интерфейса в Android.

Откроется диалоговое окно **New Project**. В нем есть несколько полей для настройки вашего проекта.

Настройте свой проект следующим образом:

- Поле «**Name**» используется для ввода названия вашего проекта, введите «**LearnAndroid**».

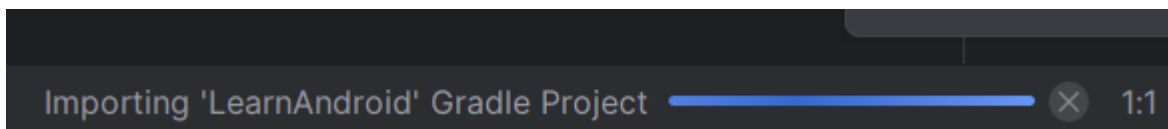
- В поле **Package name** поменяйте **второе имя** на **вашу фамилию**. Так будут организованы ваши файлы в файловой структуре. В этом случае имя пакета будет **com.leontev.learnandroid**.

- Оставьте поле «**Save location**» как есть. Оно содержит место, где сохраняются все файлы, связанные с вашим проектом. Запомните, где это находится на вашем компьютере, чтобы вы могли найти свои файлы.

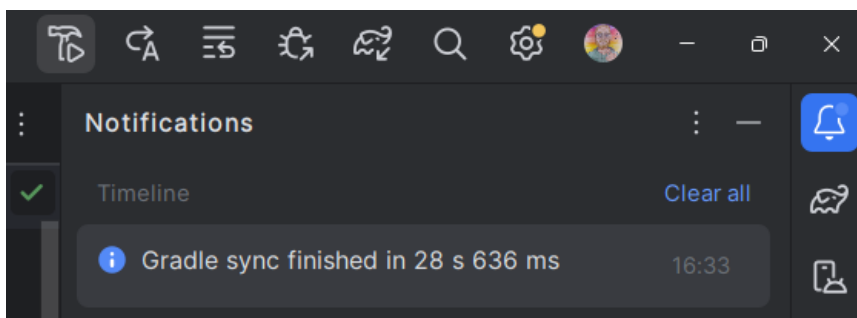
- Выберите **API 25: Android 7.1.1 (Nougat)** из меню в поле **Minimum SDK**. Он указывает минимальную версию Android, на которой может работать ваше приложение. **Важно!** Не выбирайте слишком высокую версию SDK, иначе приложение не будет запускаться на более старых устройствах.

- Нажмите **Finish**.

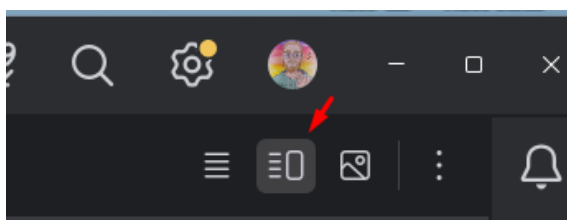
Это может занять некоторое время пока Android Studio настраивается, индикатор выполнения и сообщение показывают, настраивает ли Android Studio ваш проект:



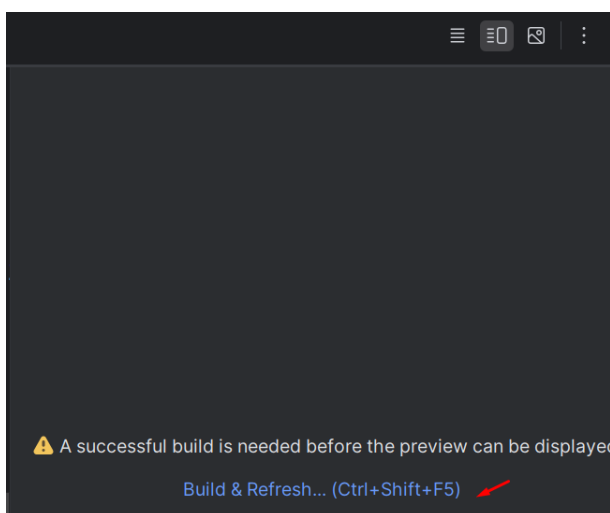
Подождите, пока индикатор сборки завершится (появится сообщение **BUILD SUCCESSFUL**):



Нажмите вкладку **Split** в правом верхнем углу, чтобы просматривать одновременно код и превью интерфейса. Также доступны режимы **Code** и **Design** отдельно:



Если превью не отображается, нажмите **Build & Refresh** в окне предпросмотра:

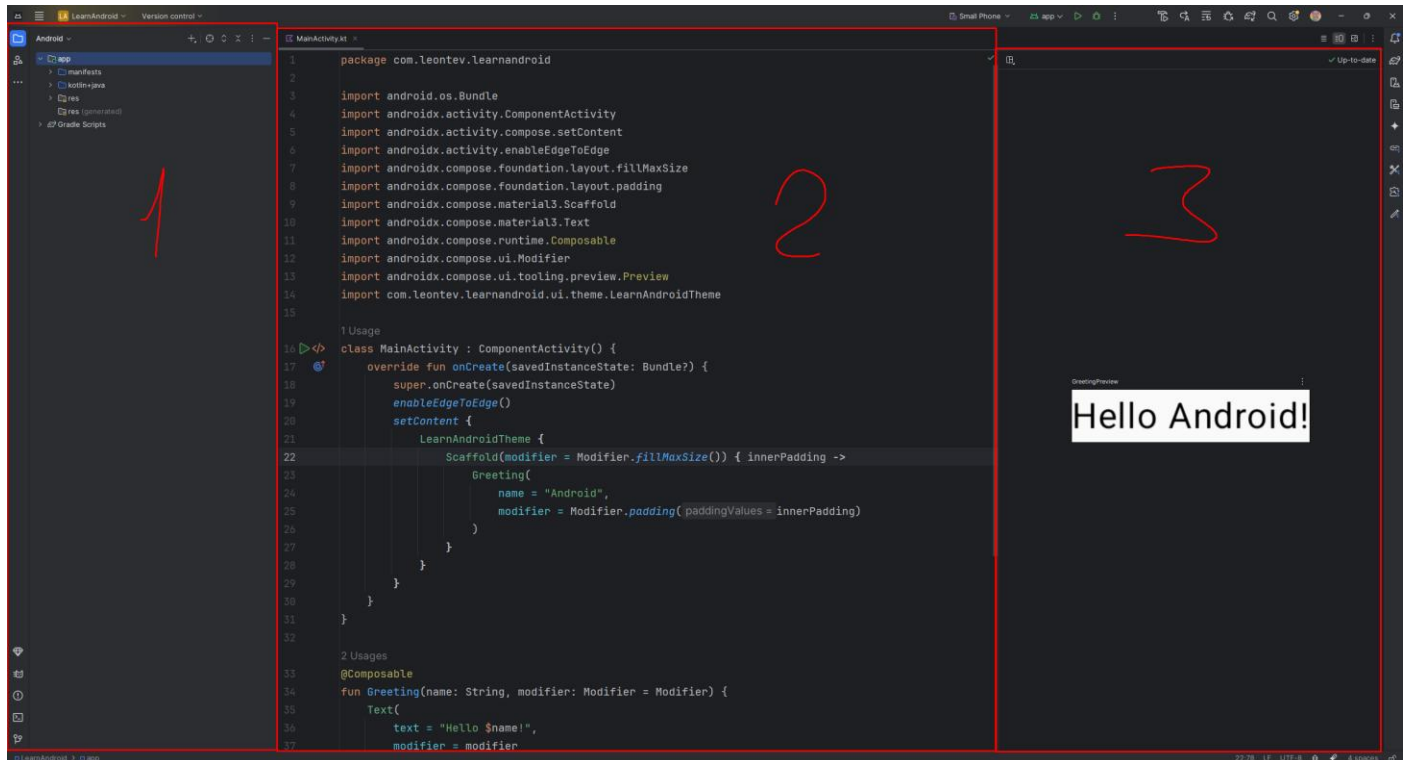


В предварительном просмотре отобразится текстовое поле с надписью « **Hello Android!** ». Активность **Empty Compose** содержит весь код, необходимый для создания этого приложения.



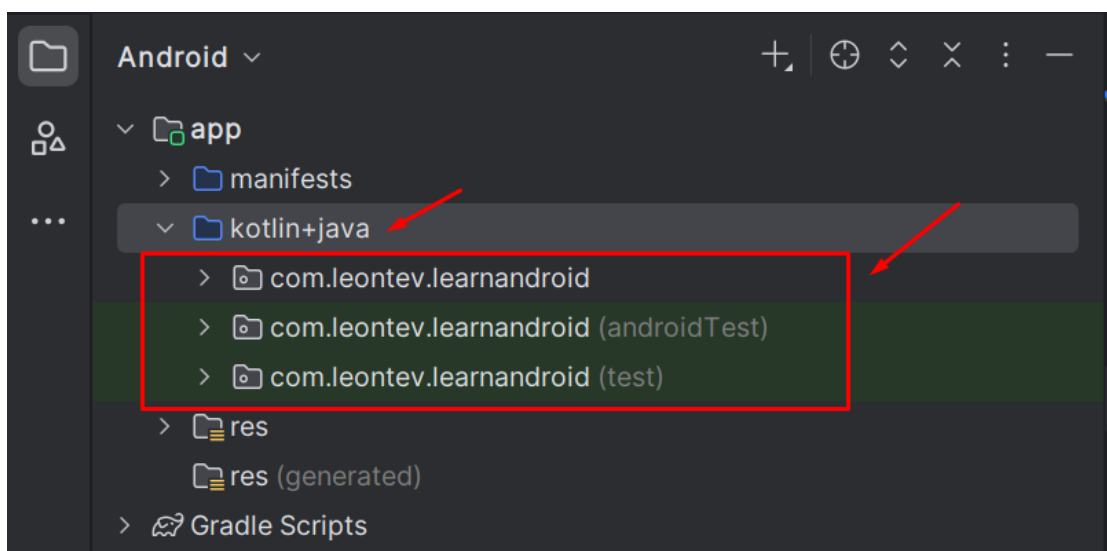
Шаг 2. Обзор интерфейса Android Studio

После нажатия кнопки «Split» вы увидите три области:



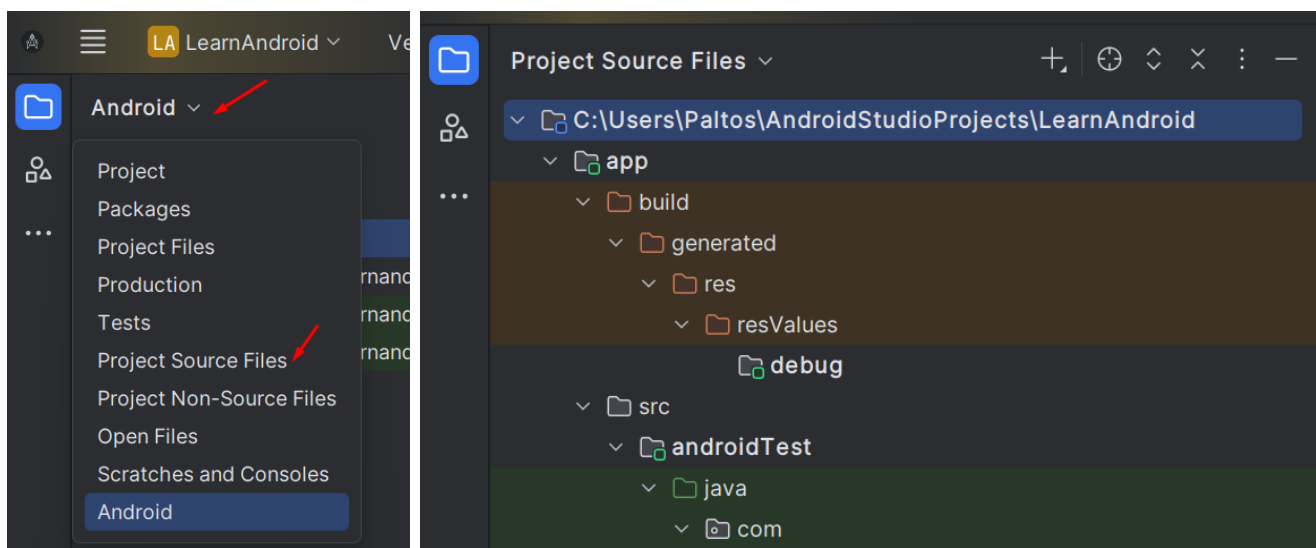
- В окне **Project** (1) отображаются файлы и папки вашего проекта.
- В окне «**Code**» (2) вы редактируете код.
- В окне «**Design**» (3) вы можете предварительно просмотреть, как будет выглядеть ваше приложение.

В Android Studio взгляните на вкладку **Project**. На ней отображаются файлы и папки вашего проекта. Когда вы настраивали свой проект, имя пакета было **com.leontev.learnandroid**. Вы можете увидеть этот пакет прямо здесь, на вкладке **Project**. Пакет — это, по сути, папка, в которой находится код. Android Studio организует проект в структуре каталогов, состоящей из набора пакетов.

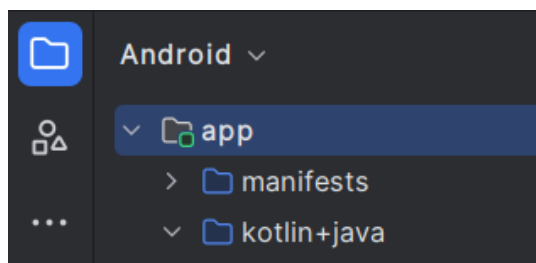


Это стандартный вид и организация файлов, которые вы используете. Это полезно, когда вы пишете код для своего проекта, потому что вы можете легко получить доступ к файлам, с которыми будете работать в своем приложении. Однако, если вы посмотрите на файлы в файловом браузере, иерархия файлов организована совсем иначе.

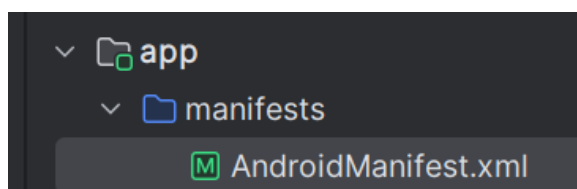
Выберите **Project Source Files** из выпадающего меню. Теперь вы можете просматривать файлы так же, как в любом файловом браузере:



Выберите **Android** еще раз, чтобы вернуться к предыдущему виду.



В файле **AndroidManifest.xml** хранятся настройки вашего приложения. Откройте его и посмотрите, в будущем мы будем к нему обращаться, все изменения тут проводятся на языке разметке **xml**.



Папка **res** (сокращение от **resource**) содержит некодированные ресурсы, такие как макеты, изображения, строки и другие файлы, которые определяют внешний вид и поведение приложения.

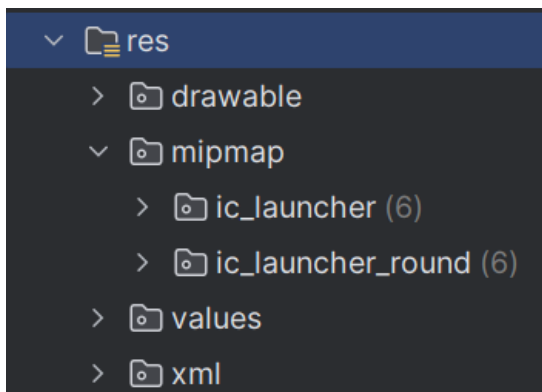
Некоторые подпапки Res-папки:

Drawable (res/drawable). Содержит изображения, XML-файлы, определяющие формы, градиенты или другие графические ресурсы.

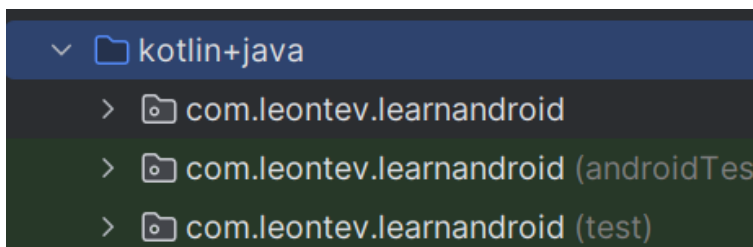
Values (res/values). XML-файлы, содержащие простые значения, такие как строки, целые числа и цвета.

Mipmap (res/mipmap). Используется для иконок запуска приложения разных размеров для различных плотностей экрана.

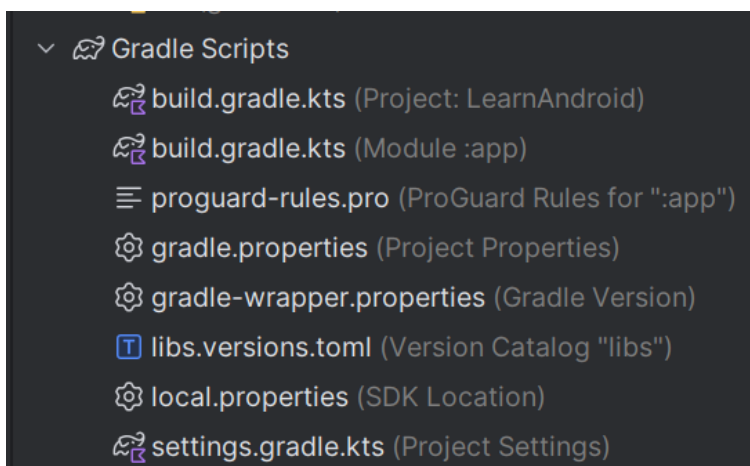
Xml (res/xml). Произвольные XML-файлы, которые можно прочитать во время выполнения, вызвав `Resources.getXML()`.



И последняя папка, в которой мы будем писать код — **kotlin+java**. Здесь находятся все файлы на языке **Kotlin** или **Java**. Обратите внимание, что используется иерархия **пакетов**, которая соответствует имени пакета, введённому вами при создании проекта (например, **com.leontev.learnandroid**).



Папка **Gradle Scripts** содержит конфигурационные файлы, которые управляют сборкой проекта, зависимостями, версиями SDK и плагинами.

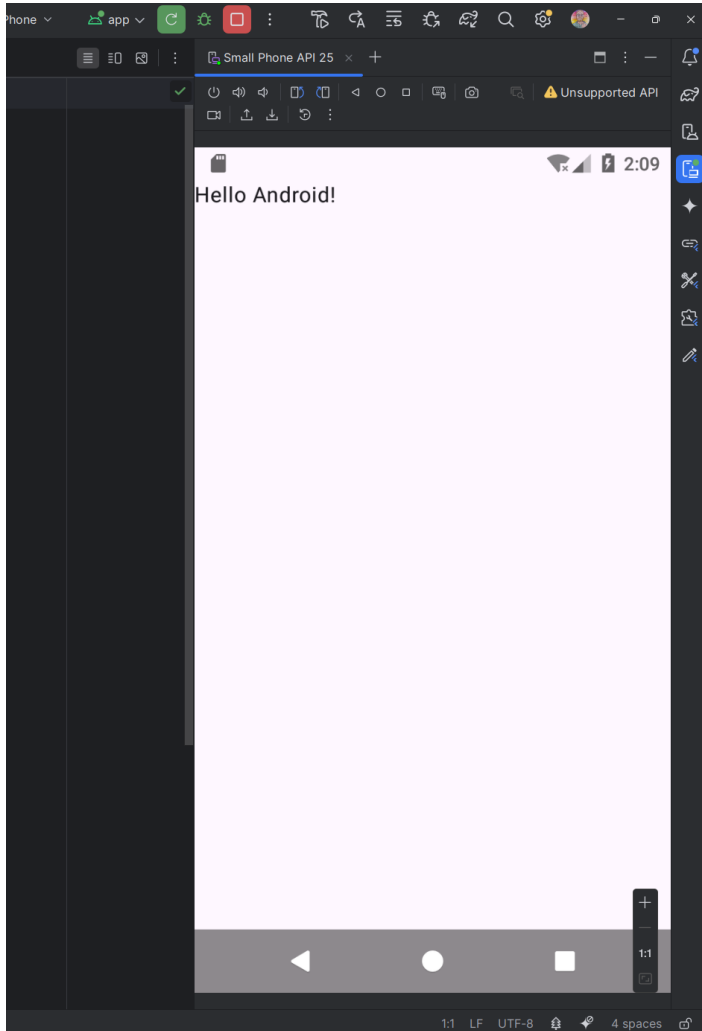


Шаг 3. Запуск эмулятора

Давайте запустим эмулятор, чтобы увидеть, как работает наше приложение.

Обратите внимание:

- Первый запуск может занять несколько минут.
- Если у вас **не установлен эмулятор**, обратитесь к начальной инструкции в курсе.

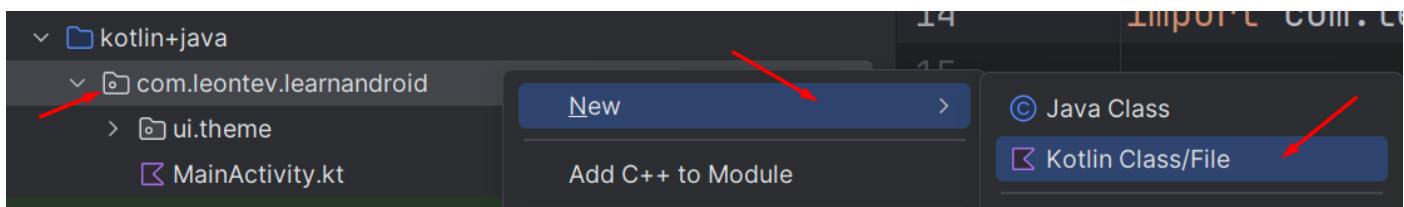


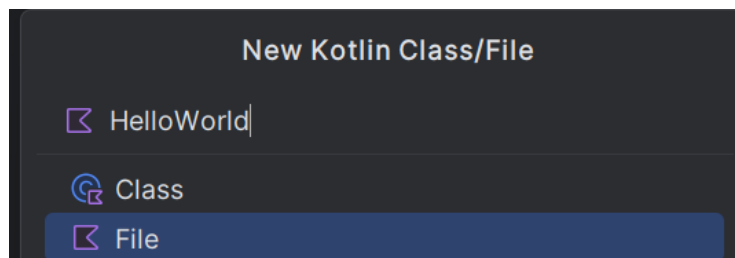
Шаг 4. Создание простой программы Hello World на Kotlin

Так как сейчас в проекте много кода, который мы пока не понимаем, давайте начнём с простого. Напишем первую программу на языке Kotlin.

Создайте новый Kotlin-файл в основном пакете:

- Щёлкните правой кнопкой мыши по папке с именем пакета (**com.leontev.learnandroid**) → **New > Kotlin Class/File** → выберите **File**, и назовите его **HelloWorld.kt**.

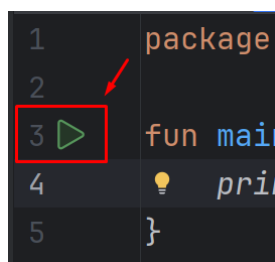




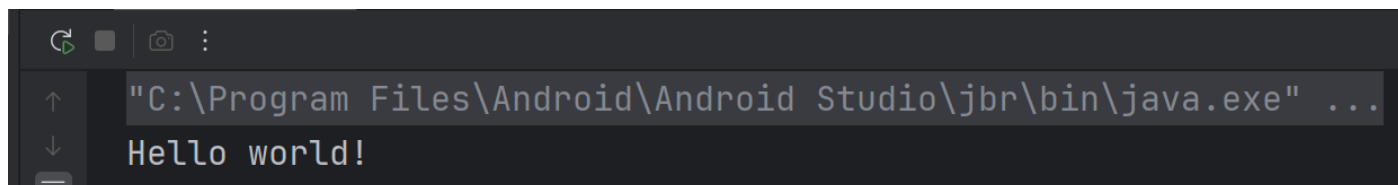
Добавляем точку входа **main** и выводим стандартный **Hello World**:



Нажмите зелёную стрелку рядом с **main()** или запустите через контекстное меню:



Если всё сделано правильно — в **консоли** появится надпись:



Шаг 5. Понимание структуры Android-приложения

Теперь перейдём к тому, как работает **настоящее Android-приложение**:

1. Сначала приложение устанавливается на устройство (эмулятор).
2. Затем оно запускается пользователем.
3. После запуска открывается главный экран (**Activity**).
4. На экране отображается пользовательский интерфейс — текст, кнопки и т.д.
5. Пользователь взаимодействует с этим интерфейсом.

Чтобы упростить разработку, **Android Studio** уже создала для нас шаблон **MainActivity**, который реализует основной экран приложения.

Откройте файл **MainActivity.kt**. В нём содержится код, который запускается при старте приложения.

Всё, что мы напишем внутри метода **onCreate** будет выполнено при запуске приложения.

```
1 Usage
16 <> class MainActivity : ComponentActivity() {
17     override fun onCreate(savedInstanceState: Bundle?) {
18         super.onCreate(savedInstanceState)
19         enableEdgeToEdge()
20         setContent {
21             LearnAndroidTheme {
22                 Scaffold(modifier = Modifier.fillMaxSize()) { innerPadding ->
23                     Greeting(
24                         name = "Android",
25                         modifier = Modifier.padding(paddingValues = innerPadding)
26                     )
27                 }
28             }
29         }
30     }
31 }
32 }
```

Давайте удалим весь код внутри **MainActivity**, кроме определения самого класса:

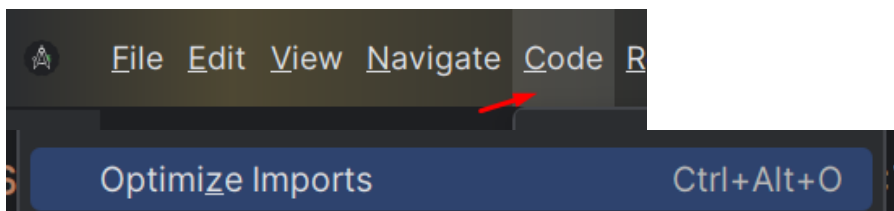
```
1 package com.leontev.learnandroid
2
3 > import ...
15
16 <> class MainActivity : ComponentActivity() {
17     |
18 }
```

После удаления кода многие **импорты** станут серыми, потому что они больше не используются:

```
3 import android.os.Bundle
4 import androidx.activity.ComponentActivity
5 import androidx.activity.compose.setContent
6 import androidx.activity.enableEdgeToEdge
7 import androidx.compose.foundation.layout.fillMaxSize
8 import androidx.compose.foundation.layout.padding
9 import androidx.compose.material3.Scaffold
10 import androidx.compose.material3.Text
11 import androidx.compose.runtime.Composable
12 import androidx.compose.ui.Modifier
13 import androidx.compose.ui.tooling.preview.Preview
14 import com.leontev.learnandroid.ui.theme.LearnAndroidTheme
```

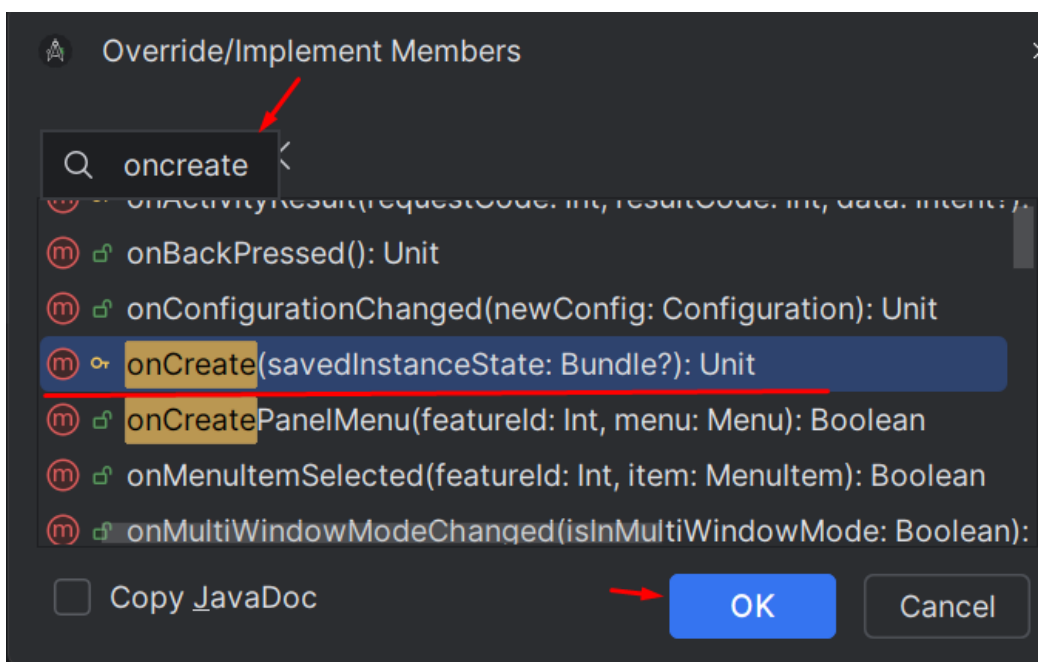
Чтобы удалить неиспользуемые импорты:

- Перейдите в верхнее меню **Code** → **Optimize Imports**, или
- Нажмите сочетание клавиш **Ctrl + Alt + O** (на Mac: **Cmd + Option + O**).



Теперь добавим метод **onCreate()** — он вызывается, когда экран приложения создаётся:

- Внутри класса нажмите **Ctrl + O** и начните вводить **onCreate**.
- Выберите метод с параметром **savedInstanceState: Bundle?**.



- Вставьте его.

В итоге у вас получится:

```
class MainActivity : ComponentActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
    }  
}
```

super.onCreate(...) вызывает поведение родительского класса, а ваш код будет добавлен ниже.

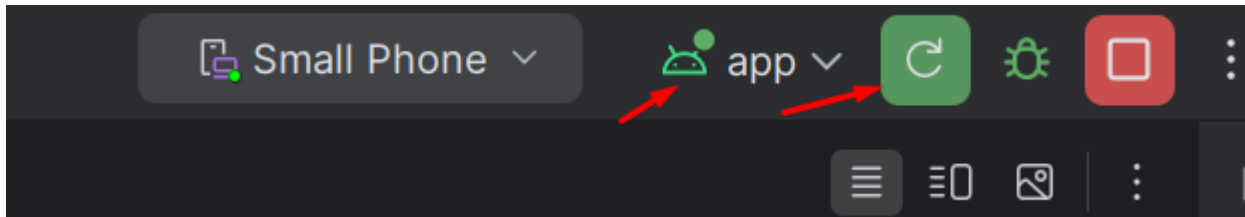
Обратите внимание, что у нас подключился импорт:

```
import androidx.activity.ComponentActivity
```

Теперь добавим простой вывод текста в консоль:

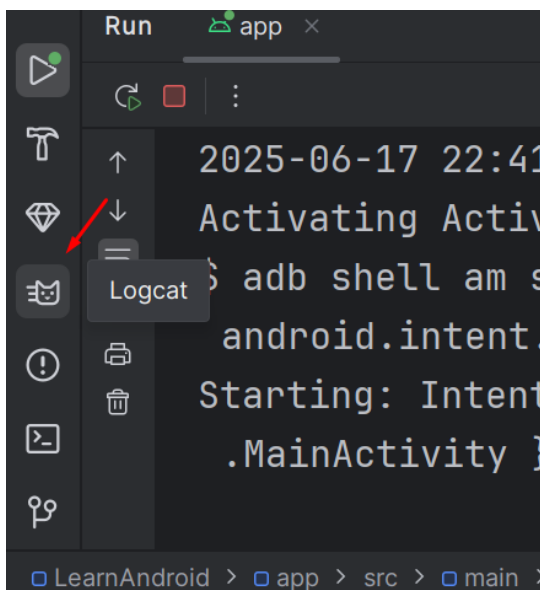
```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        println("Hello world!")
    }
}
```

Нажмите зелёную кнопку **Run 'app'** или используйте **Shift + F10**. Убедитесь, что в выпадающем списке выбрано **app**:

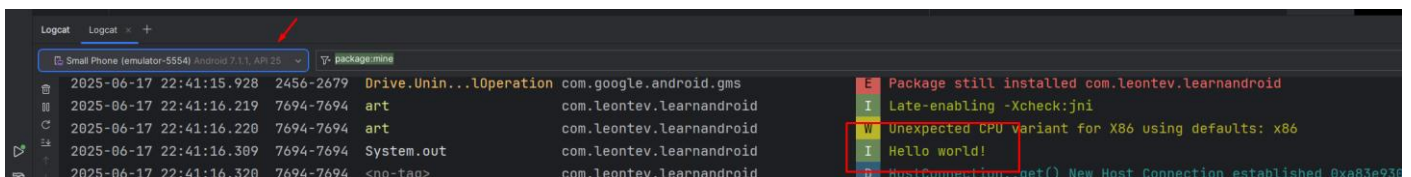


Поскольку мы вывели текст не на экран, а в консоль, он не отобразится в приложении. Зато его можно увидеть в **Logcat**:

- Нажмите на вкладку **Logcat** (внизу **Android Studio**):



Выберите ваше устройство в списке и сможете увидеть вывод надписи на консоль:



Чтобы отобразить текст не в консоли, а на экране, мы используем функцию **setContent()** из **Jetpack Compose**. Добавьте его:

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent { }
        println("Hello world!")
    }
}
```

Обратите внимание, если вы руками просто перепишите код, то у вас не добавится в начале импорт (используйте автозавершение с клавишей TAB):

```
import androidx.activity.compose.setContent
```

Шаг 6. Использование setContent() и Composable-функций

Функция `setContent` принимает лямбда-выражение, поэтому круглые скобки () можно опустить, если не передаются аргументы. Внутри этой функции мы определяем пользовательский интерфейс (UI) с помощью **Composable-функций**.

Composable-функции — это специальные функции, обозначенные аннотацией `@Composable`, которые описывают элементы интерфейса в **Jetpack Compose**. Они не возвращают значения, а используются для **визуализации** компонентов.

Создадим простой текст на экране с помощью **Composable-функции Text**:

```
setContent {
    Text()
}
```

В отличие от обычных функций, **Composable-функции** в **Compose** начинаются с заглавной буквы (например, **Text**, **Button** и т.д.).

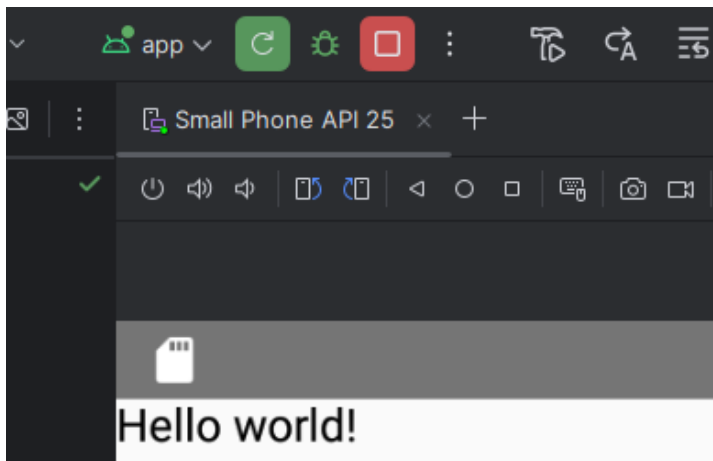
Android Studio автоматически подскажет нужный импорт. Если он не добавился:

```
import androidx.compose.material3.Text
```

Указываем именованный параметр `text` и что хотим вывести на экран:

```
setContent {
    Text(text = "Hello world!")
}
```

Запустите приложение (сочетание клавиш **Shift + F10**). Если всё сделано верно, вы увидите надпись на экране.



Что такое **Jetpack Compose**? (Кратко)

Особенность	Описание
Декларативный подход	UI описывается в виде Kotlin-функций, а не XML
UI = функции	Элементы интерфейса — это обычные функции с <code>@Composable</code>
Реактивность	UI обновляется при изменении состояния автоматически
Нет <code>findViewById</code>	Не нужно вручную искать элементы по ID
Меньше кода	Упрощённая реализация интерфейсов
Live Preview	Возможность просматривать результат без запуска эмулятора

Если вы зажмёте **Ctrl** и щёлкните **левой кнопкой мыши** по `Text`, вы откроете описание этой функции:

```
91  @Composable
92  fun Text(
93      text: String,
94      modifier: Modifier = Modifier,
95      color: Color = Color.Unspecified,
96      fontSize: TextUnit = TextUnit.Unspecified,
97      fontStyle: FontStyle? = null,
98      fontWeight: FontWeight? = null,
99      fontFamily: FontFamily? = null,
100     letterSpacing: TextUnit = TextUnit.Unspecified,
101     textDecorations: TextDecorations? = null,
102     textAlign: TextAlign? = null,
103     lineHeight: TextUnit = TextUnit.Unspecified,
104     overflow: TextOverflow = TextOverflow.Clip,
105     softWrap: Boolean = true,
106     maxLines: Int = Int.MAX_VALUE,
107     minLines: Int = 1,
108     onTextLayout: ((TextLayoutResult) -> Unit)? = null,
109     style: TextStyle = LocalTextStyle.current
110 ) {
```

Некоторые параметры по названию думаю вам будут очевидны. Позже мы будем к ним обращаться также через именованные параметры.

Давайте, например посмотрим параметр, который отвечает за жирность текста **fontWeight**. Чтобы узнать какие значения ему можно передавать мы можем обратиться к документации или же можем перейти в описание класса и посмотреть. Зажмите левый **Ctrl** и щёлкните ЛКМ по классу **FontWeight**:

```
fontWeight: FontWeight? = null,  
fontFamily: FontFamily? = null,
```

Мы можем догадаться что **Bold** будет отвечать за **полужирность** текста:

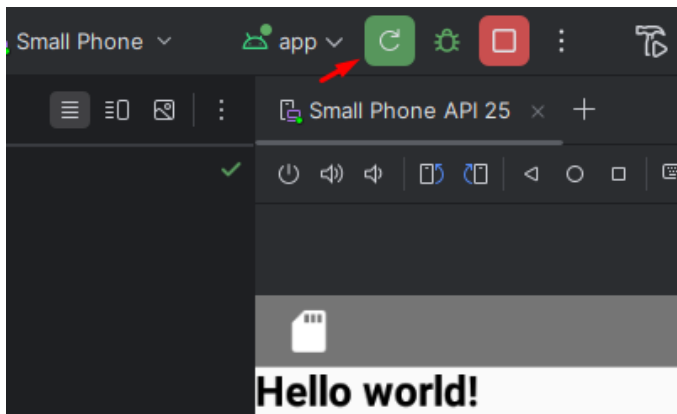
```
@Stable  
val Bold = W700
```

Давайте это проверим. Вернёмся в наш **MainActivity**.

Передадим в качестве второго параметра, через запятую – **fontWeight**. Передаём значение используя название класса, и через точечную нотацию выбираем **Bold**:

```
Text(text = "Hello world!",  
      fontWeight = FontWeight.Bold)
```

Перезапустите приложение и убедитесь, что тест стал полужирным:



Теперь самостоятельно попробуйте поменять **цвет текста** на **зеленый**:



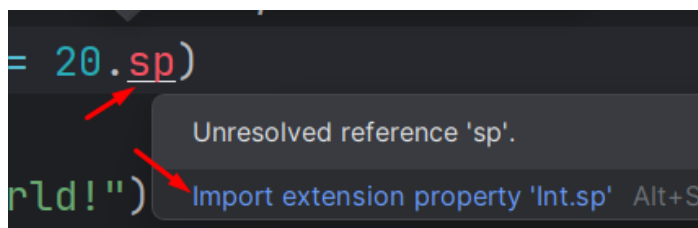
Теперь изменим размер текста. Для этого нам понадобится параметр **fontSize**. И для указания размера нам нужно с вами будет воспользоваться не величиной пикселей (**px**), а **sp**. Для работы с пикселями существуют - **dp** (независимые от плотности пиксели) и **sp** (масштабируемые пиксели), которые одинаково соответствуют широкому

диапазону плотностей экранов, классов размеров, форм-факторов и соотношений сторон, используемых в устройствах Android. Подробнее прочитать про них вы можете в [официальной документации](#).

```
Text(text = "Hello world!",  
      fontWeight = FontWeight.Bold,  
      color = Color.Green,  
      fontSize = 20.sp)
```

Вкратце – для размеров любых элементов (кроме шрифтов) используйте единицу измерения **dp**. Для размера текста **sp**. Обе единицы не зависят от плотности экрана, поэтому одинаково выглядят на разных устройствах с разным разрешением.

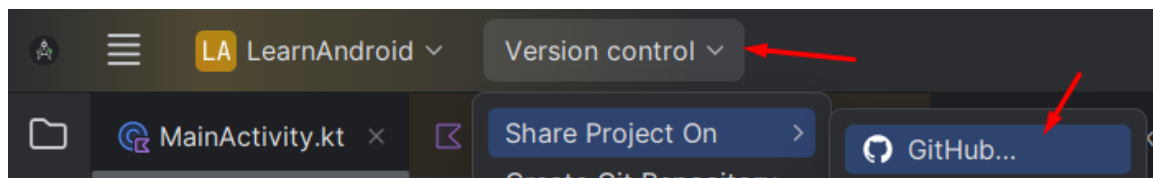
После нам также надо импортировать пакет:



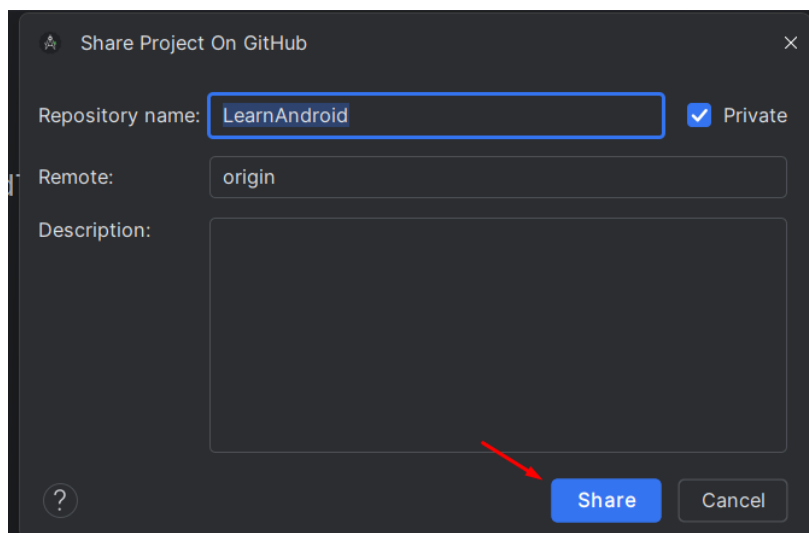
Попробуйте самостоятельно сделать текст курсивом.

Шаг 7. Создание Git репозитория

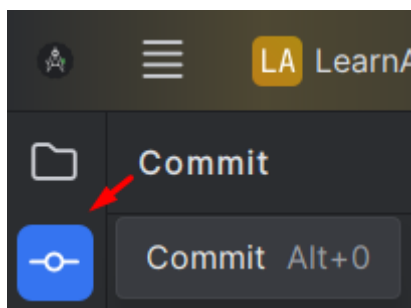
Нажмите на **Version control** и выберите пункт **Share Project On -> GitHub**:



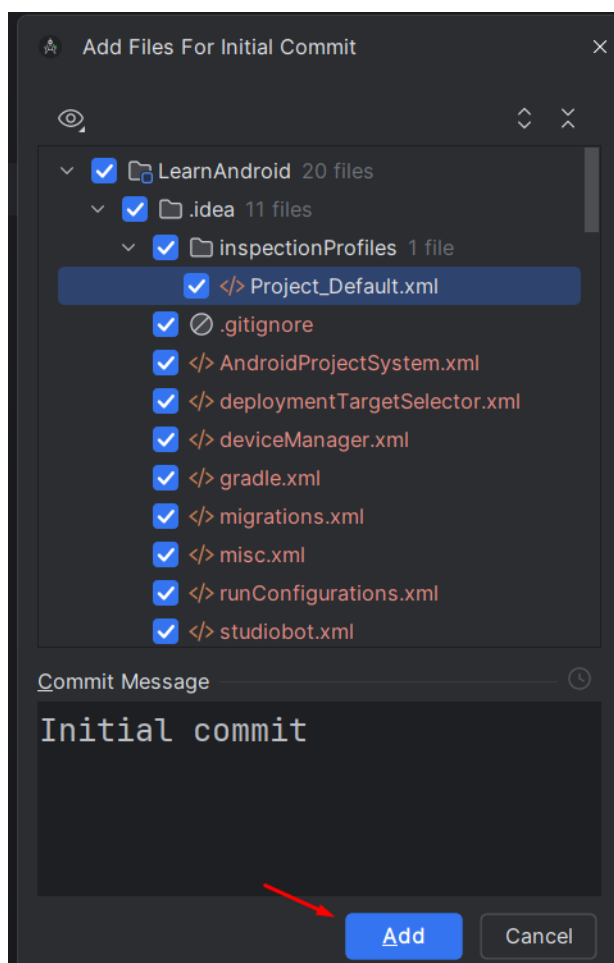
Если вы не вошли в свою учетную запись, то войдите на данном шаге, затем нажмите **Share**:



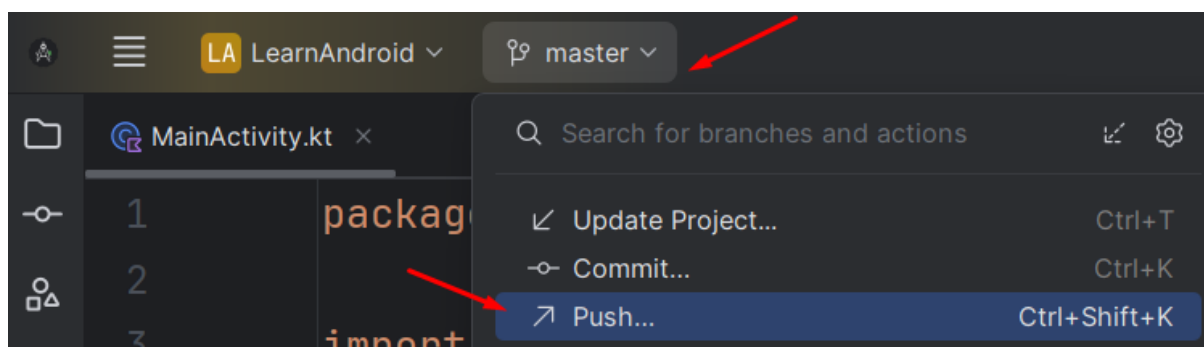
После нажмите на значок слева **Commit** (сочетание клавиш **Alt+0**):



Выберите все файлы, введите название коммита (например **Initial commit**) и **Add**:



После можете из главной ветки запустить проект:



Итак, у нас было задание сделать текст курсивом. Надеюсь, вы догадались что за это отвечает параметр – **fontFamily**. И у него вы нашли свойство – **Cursive**. Поэтому нужно было добавить следующую строчку:

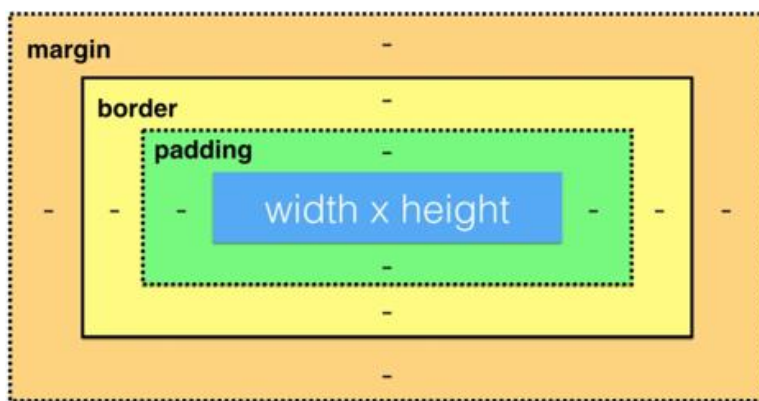

```
fontFamily = FontFamily.Cursive
```

Шаг 8. Modifier

Сейчас мы видим, что наш текст расположен близко к краям, и было бы неплохо добавить отступы:



Если вы учили **html** и **css** то слышали про такое свойство как **padding**. **Padding** в HTML — это свойство, которое устанавливает внутренние отступы (поля) вокруг содержимого элемента. Оно создаёт пространство между содержимым и границей элемента, чтобы текст не прилегал плотно к краю. Вот вам картинка, если забыли:

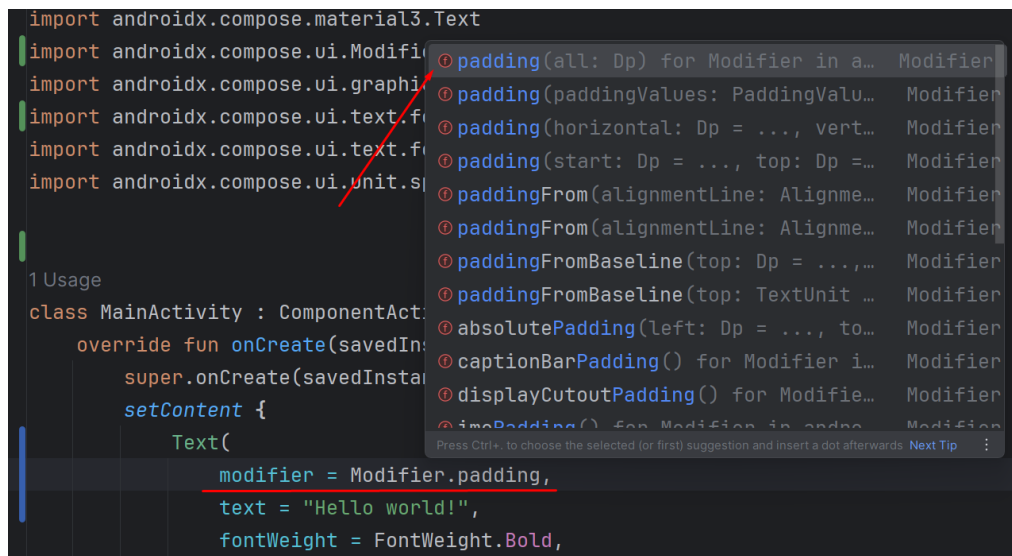


В **Android Studio** - **padding** означает отступы внутри элемента. Если мы хотим установить отступы для нашего текста, то у функции **Text** его не обнаружим. Это связано с тем, что отступы присутствуют у большого количества элементов (кнопки, поля, текст, картинки и т.д.). Поэтому, чтобы не добавлять одинаковые свойства UI был придуман интерфейс **Modifier**:

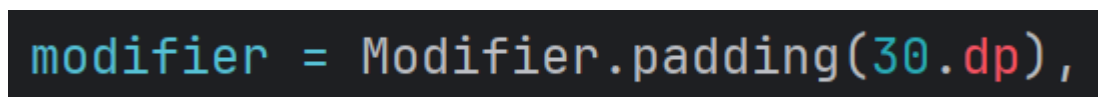
```
@Composable
fun Text(
    text: String,
    modifier: Modifier = Modifier,
    color: Color = Color.Unspecified,
```

Он присутствует у почти всех **Composable**-функциях, и отвечает за различные настройки. Мы можем вызывать у него различные методы и изменять отображение элементов.

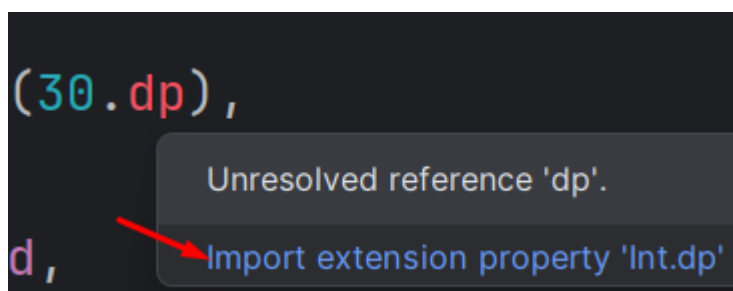
Добавим тексту **отступы** со всех сторон. Для этого добавьте в самое начало **modifier**, начните вводить **padding**, и увидите несколько вариантов, можете выбрать любой:



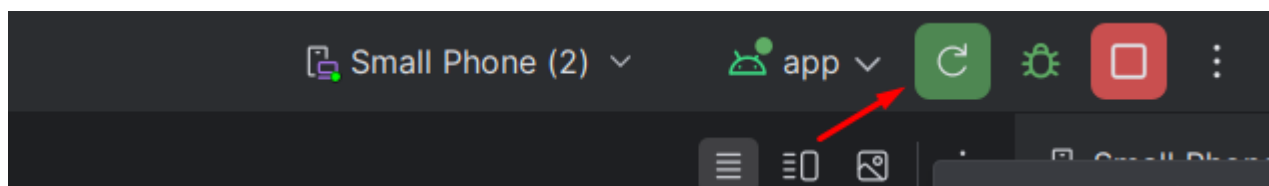
Поставьте, например **30.dp**:



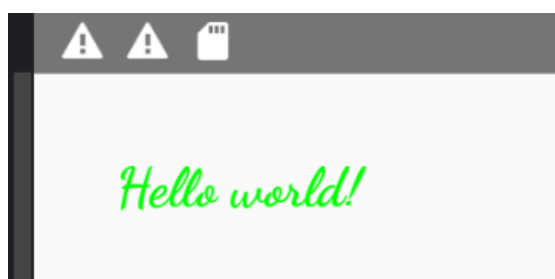
И импортируйте отсутствующий пакет:



Перезапустите приложение:



И видим результат:



Теперь попробуйте самостоятельно указать разные отступы, передав 4 аргумента:

```
modifier = Modifier.padding(start = 30.dp, top = 16.dp, end = 25.dp, bottom = 32.dp),
```

- **android:paddingStart** — для левого (стартового) края вида;
- **android:paddingTop** — для верхнего края;
- **android:paddingEnd** — для правого (конечного) края;
- **android:paddingBottom** — для нижнего края.

Start и End называются так (а не left и right, как нам привычно), потому что есть регионы где запись идёт справа налево, поэтому указываются такие аргументы.

Также у **Modifier** мы можем вызывать несколько настроек. Для этого достаточно поставить точку. Давайте поменяем цвет фона:

```
Text(  
    modifier = Modifier  
        .padding(start = 30.dp, top = 16.dp, end = 25.dp, bottom = 32.dp)  
        .background(color = Color.Red),
```

Перезапустите приложение и обратите внимание как у нас отобразилось:



Хотя вы скорее всего ожидали увидеть следующее отображение:



Это связано с тем, что последовательность функций влияет на результат. Сейчас у нас сначала идёт **padding**, потом **background** — отступы будут с фоном. А если мы хотим, чтобы сначала шёл **background**, а затем **padding** — фон только у текста, отступы пустые, то поменяем последовательность:

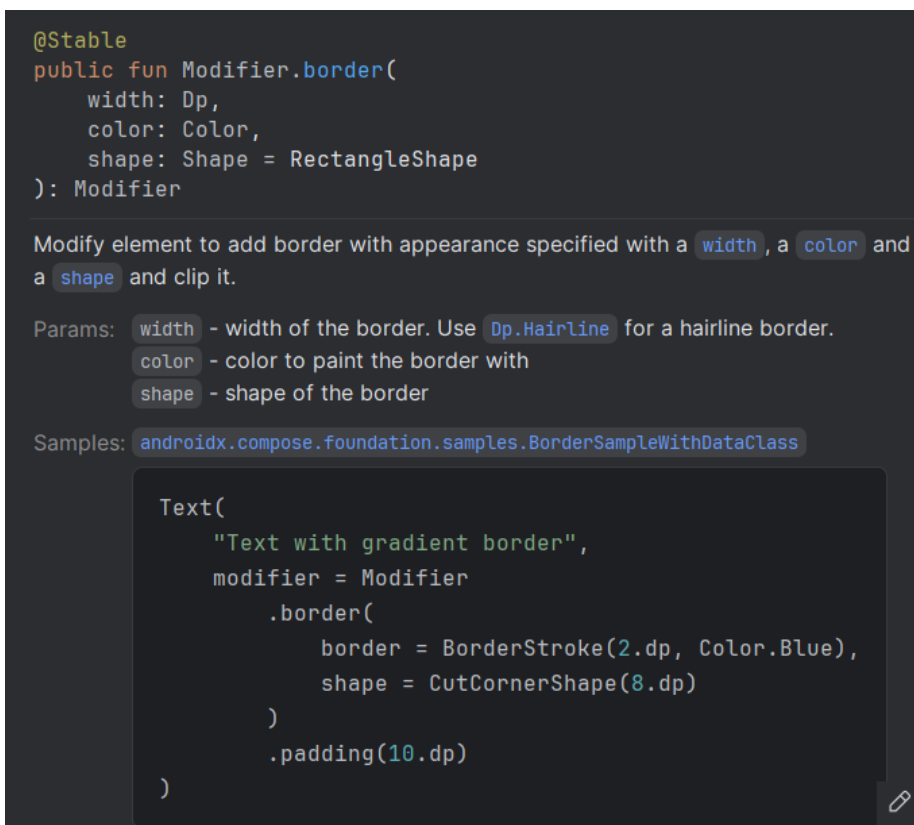
```
modifier = Modifier
    .background(color = Color.Red)
    .padding(start = 30.dp, top = 16.dp, end = 25.dp, bottom = 32.dp),
```

Перезапустите приложение.

Попробуйте самостоятельно добавить обводку (**border**):



Для набора значений, вы всегда можете воспользоваться подсказкой, которая появляется при наведении:



И последний модификатор, который мы с вами установим это **width**:

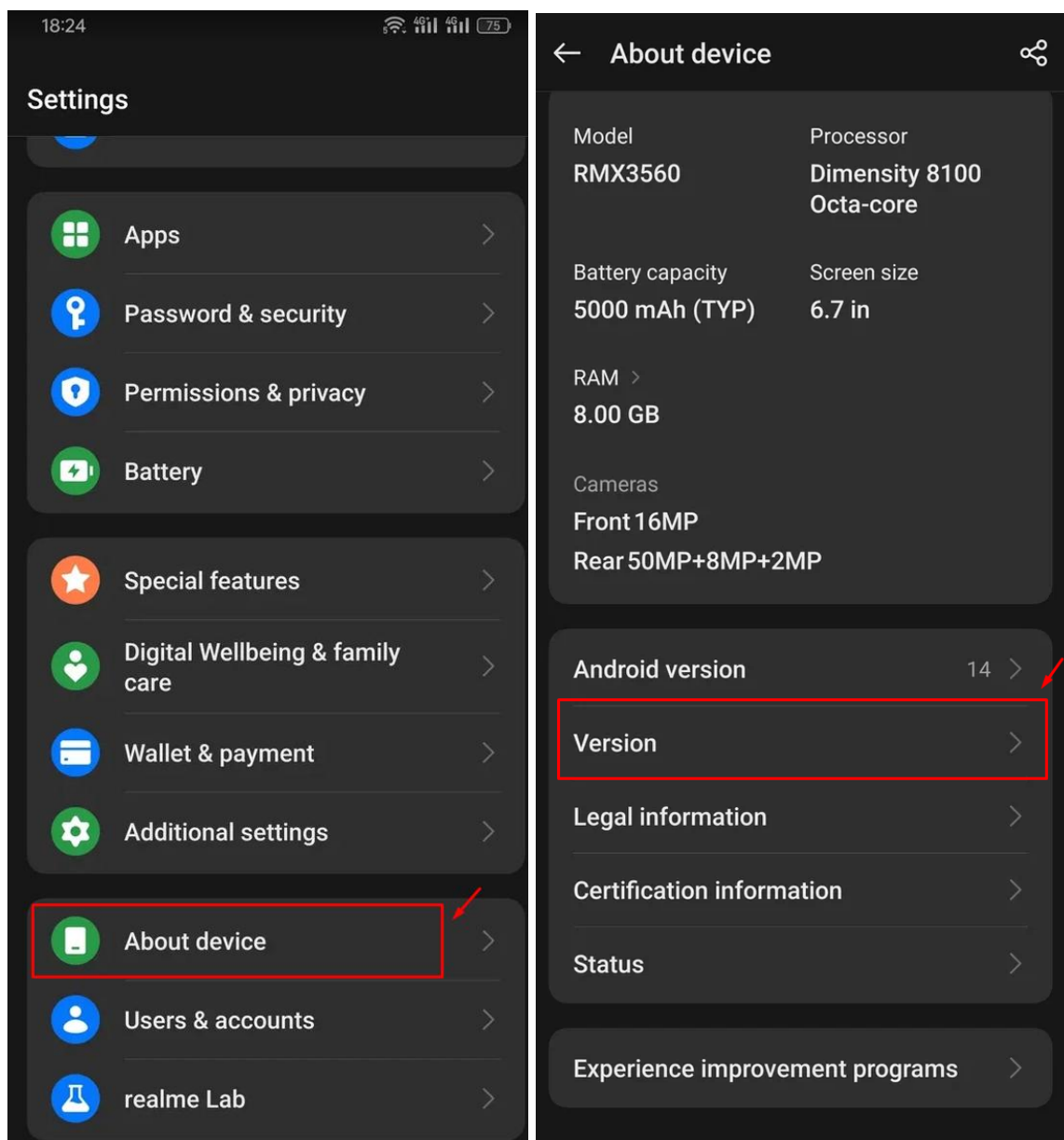
```
.width(width = 60.dp)
```

Шаг 9. Подключаем Android-устройство

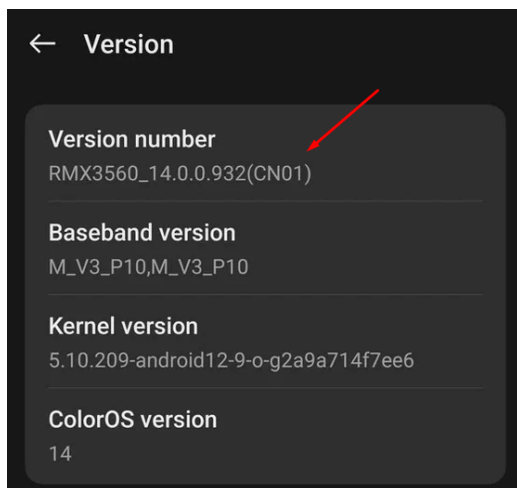
Далее мы подключим наше Android-устройство, чтобы протестировать непосредственно в нём приложение. Чтобы Android Studio могла взаимодействовать с вашим устройством Android, необходимо включить отладку по USB в настройках параметров разработчика устройства.

Чтобы отобразить параметры разработчика и включить отладку по USB. У разных моделей может отличаться процедура выполнения. В данном примере реализация на моём телефоне

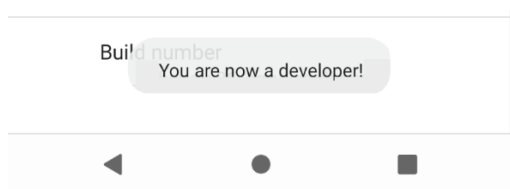
На устройстве Android нажмите «Настройки» > «О телефоне» (Settings > About device), Крутите вниз, и переходите в меню Version:



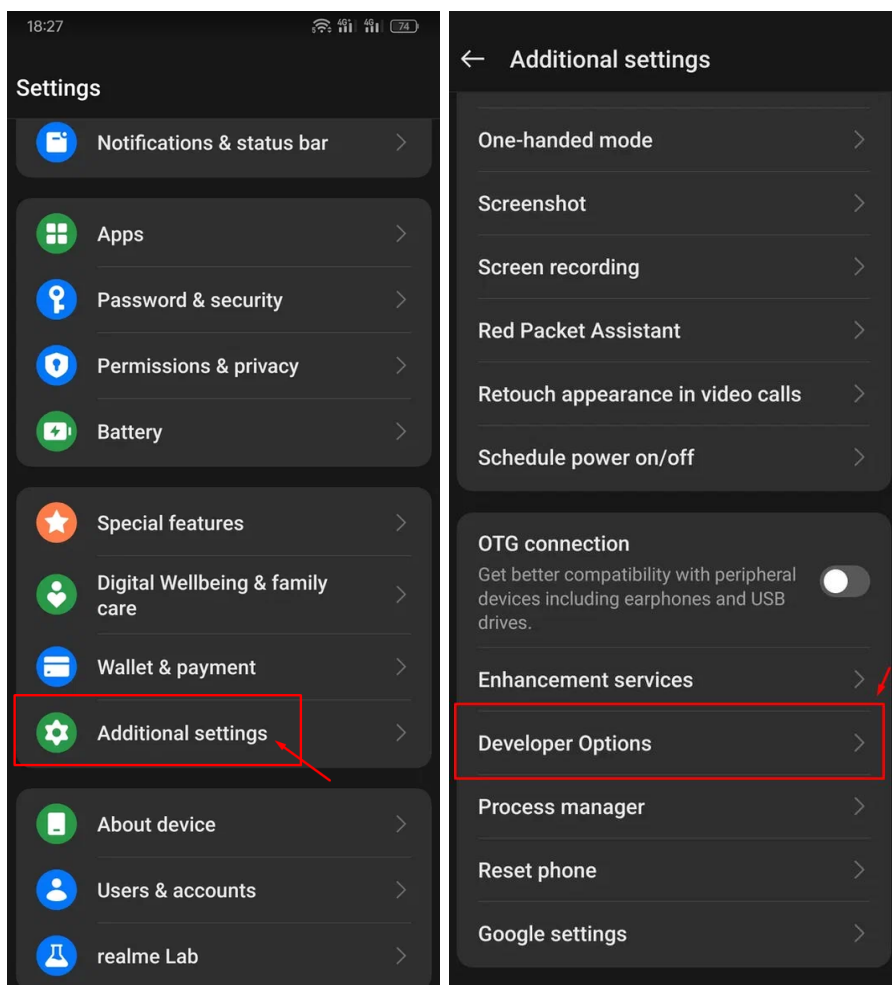
Нажмите на «Версия сборки» (Version number) 7 раз:



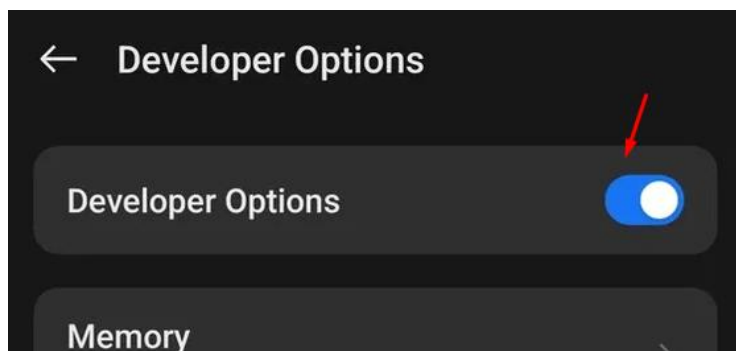
Высветится необходимость ввести пароль. введите пароль или пин-код вашего устройства. Вы поймете, что все прошло успешно, когда увидите сообщение « **Вы теперь разработчик!**» (You are now a developer!)



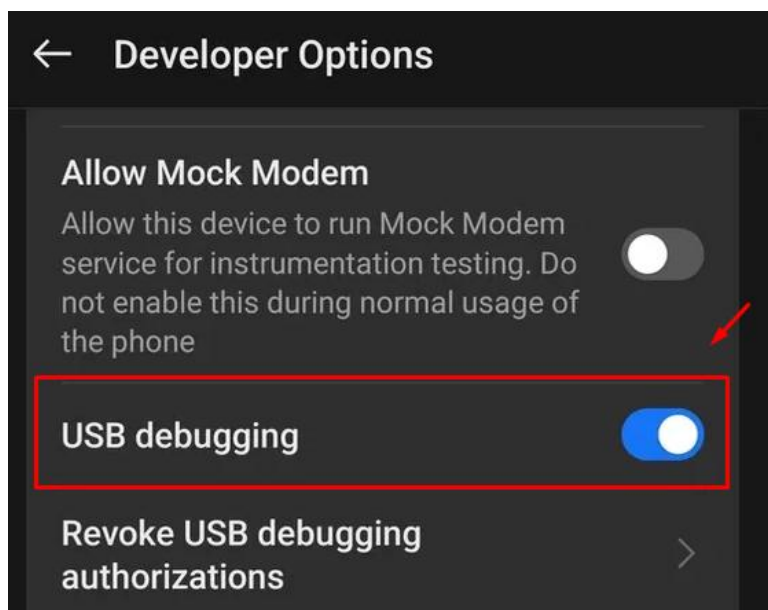
Вернитесь в «Настройки» и нажмите «Система» – «Расширенные настройки» (System – Additional settings) и выберите там Параметры разработчика (Developer Options):



Проверьте что они включены:

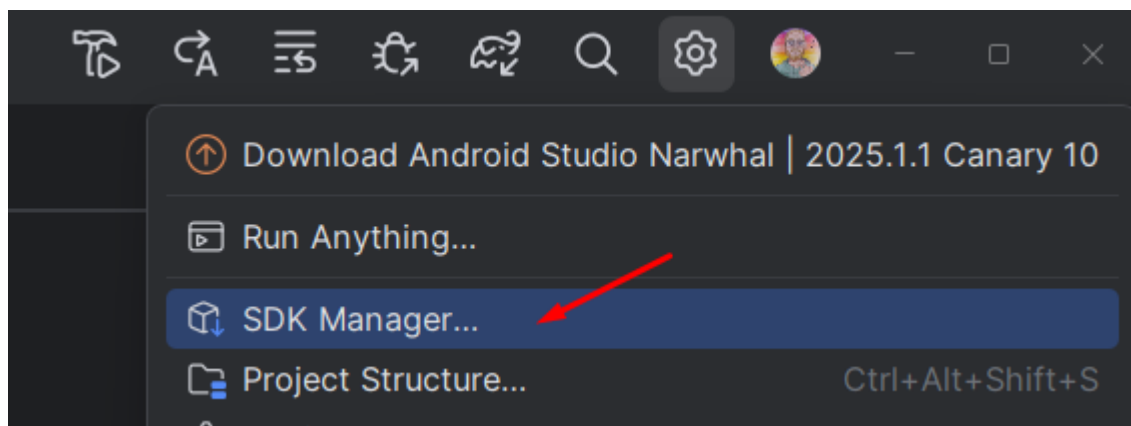


Затем коснитесь переключателя «Отладка по USB»(USB debugging) , чтобы включить его:

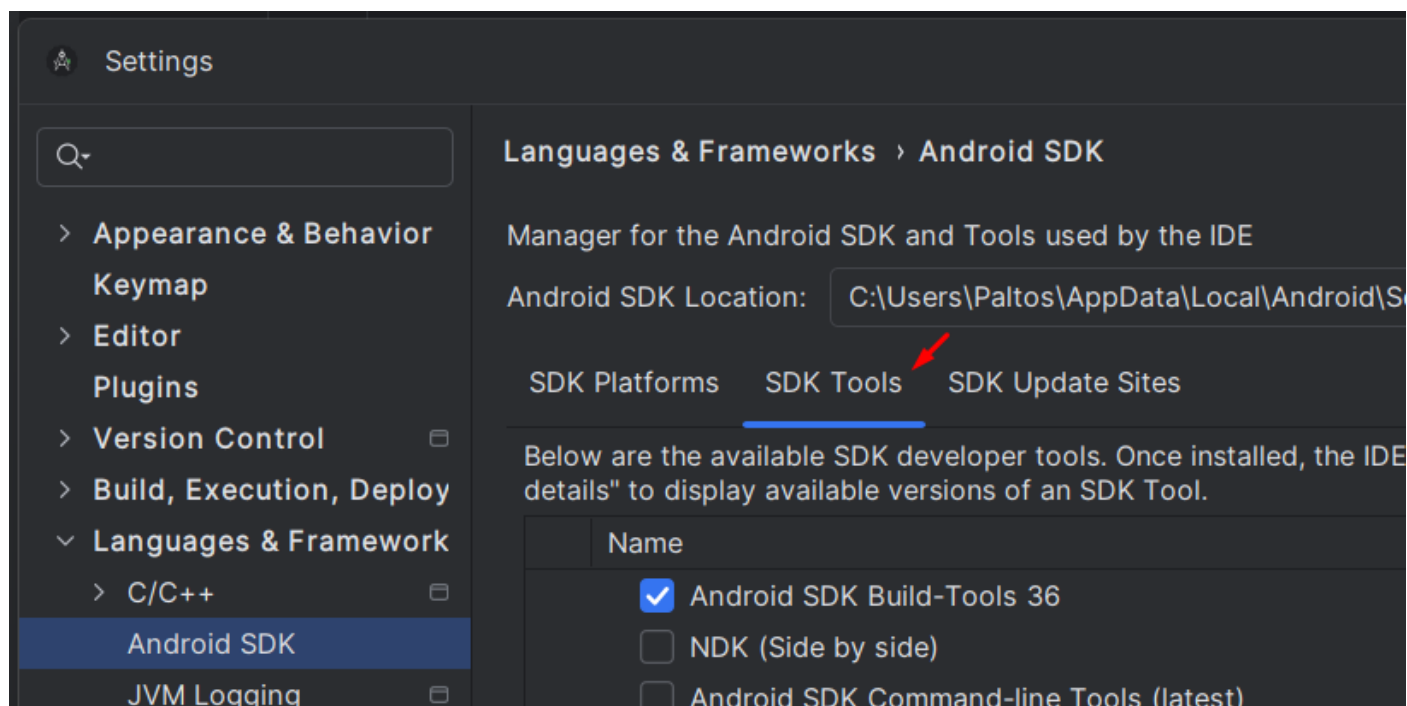


Далее вам необходимо установить драйвер USB-устройства, прежде чем вы сможете запустить приложение на физическом устройстве.

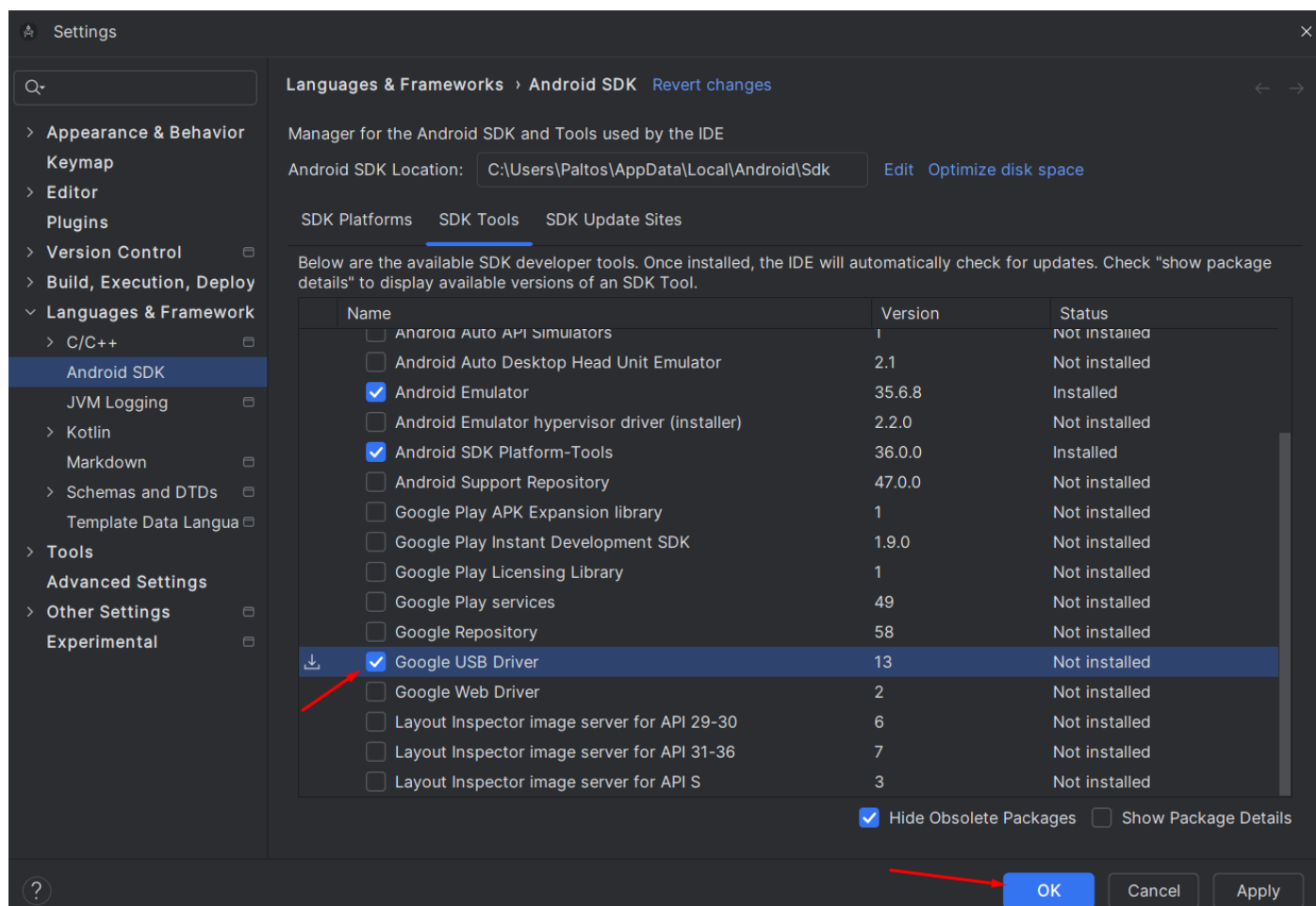
В **Android Studio** нажмите **Tools > SDK Manager**:



Перейдите на вкладку «**SDK Tools**»:

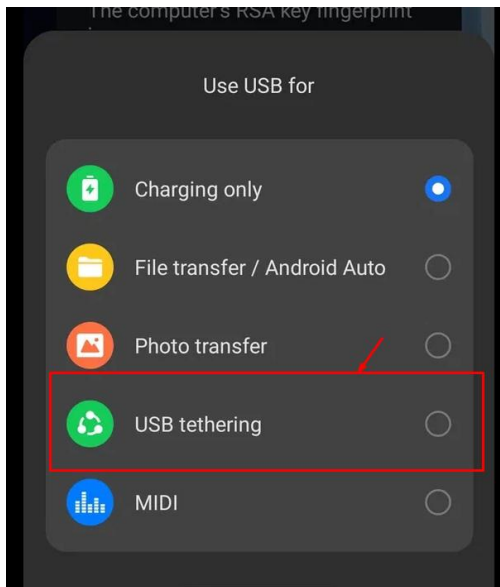



Выберите **Google USB Driver** и нажмите **OK**.

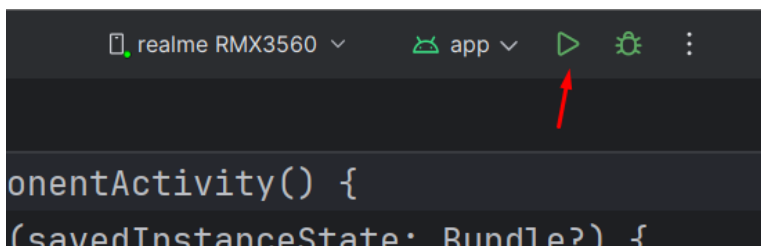


После этого файлы драйвера будут загружены в каталог `android_sdk\extras\google\usb_driver`. Теперь вы можете подключиться и запустить свое приложение из Android Studio.

Подключите свое устройство Android к компьютеру с помощью кабеля USB. На вашем устройстве должно появиться диалоговое окно, в котором вас попросят разрешить отладку по USB (**USB tethering**):



В Android Studio на вашем компьютере убедитесь, что ваше устройство выбрано в раскрывающемся списке. Нажмите .



Android Studio установит приложение на ваше устройство и запустит его.

В следующей лабораторной работе мы продолжим работу с нашим проектом.