JetPack Compose

Jetpack Compose представляет современный тулкит от компании Google для создания приложений под ОС Android на языке программирования Kotlin. Jetpack Compose упрощает написание и обновление визуального интерфейса приложения, предоставляя декларативный подход.

Операционной системе Android более 10 лет. За этот период API и библиотеки для создания приложений под эту ОС много раз обновлялись, дополнялись, одни API устаревали, другие, наоборот, добавлялись в арсенал разработчиков. Но в этоге подобное развитие привело к усложнению платформы. Чтобы упростить разработку, сделать ее более быстрой, простой, упростить поддержку компания Google в мае 2019 года анонсировала новый тулкит - Jetpack Compose. В августе 2020 вышла первая альфа-версия тулкита. А 28 июля 2021 года вышла первая стабильная версия - Jetpack Compose 1.0, которая является текущей на момент написания данной статьи и которая применяется далее в дальнейших статьях данного руководства.

Jetpack совместим с существующим набором библиотек Android, которые можно использовать в стандартных проектах на Java и Kotlin для написания приложений под Android. Отличительной же чертой Jetpack Compose является то, что он предлагает кардинально другой подход к созданию приложений.

Прежде всего, Jetpack Compose предлагает использовать язык Kotlin и все его преимущества. Соответственно для работы с тулкитом необходимо иметь базовые знания данного языка. Для этого можно обратиться к руководству по языку Kotlin на этом сайте.

Jetpack уменьшает объем кода.

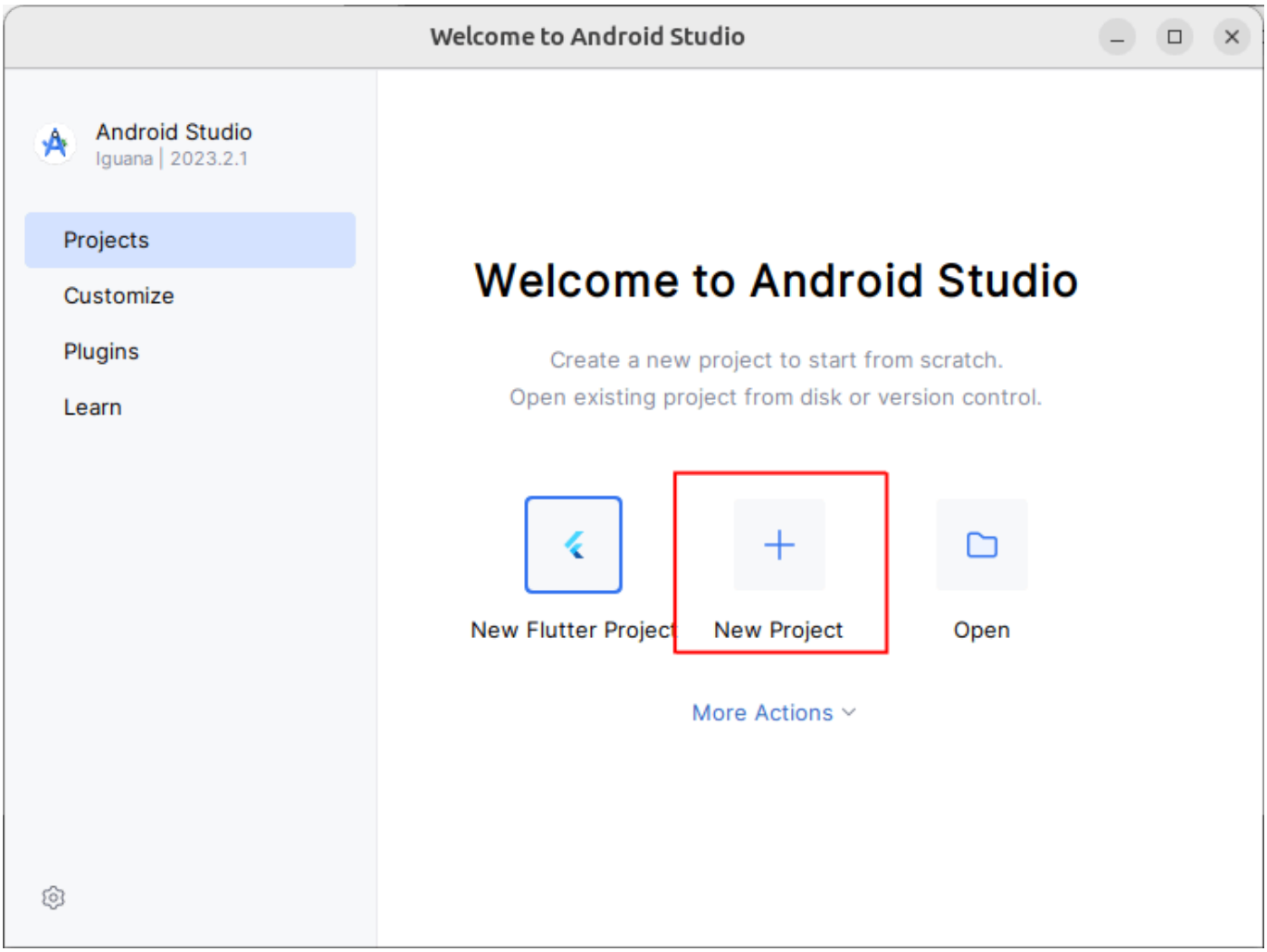
Jetpack Compose предлагает декларативный API, который является более интуитивным.

Ключевой концепцией тулкита Jetpack Compose является composable-функция (функция, которая имеет аннотацию @Composable). Такие функции представляют некоторые части визуального интерфейса, из которых строится приложение. Это упрощает построение и обновление сложных интерфейсов, тестирование и поддержку самих компонентов

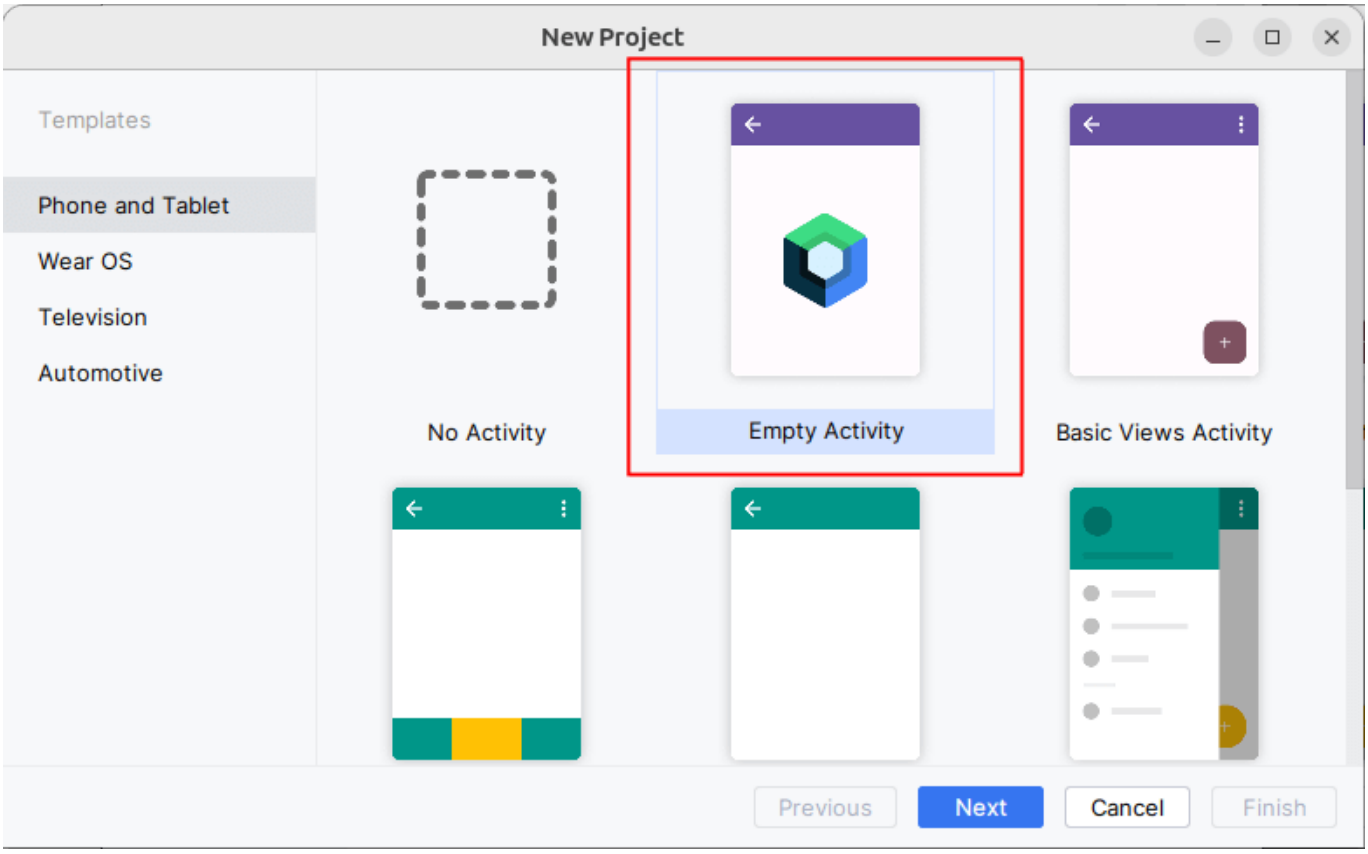
Первый проект на Jetpack Compose

Итак, создадим первый проект с использованем тулкита Jetpack Compose. Для этого запустим среду Android Studio.

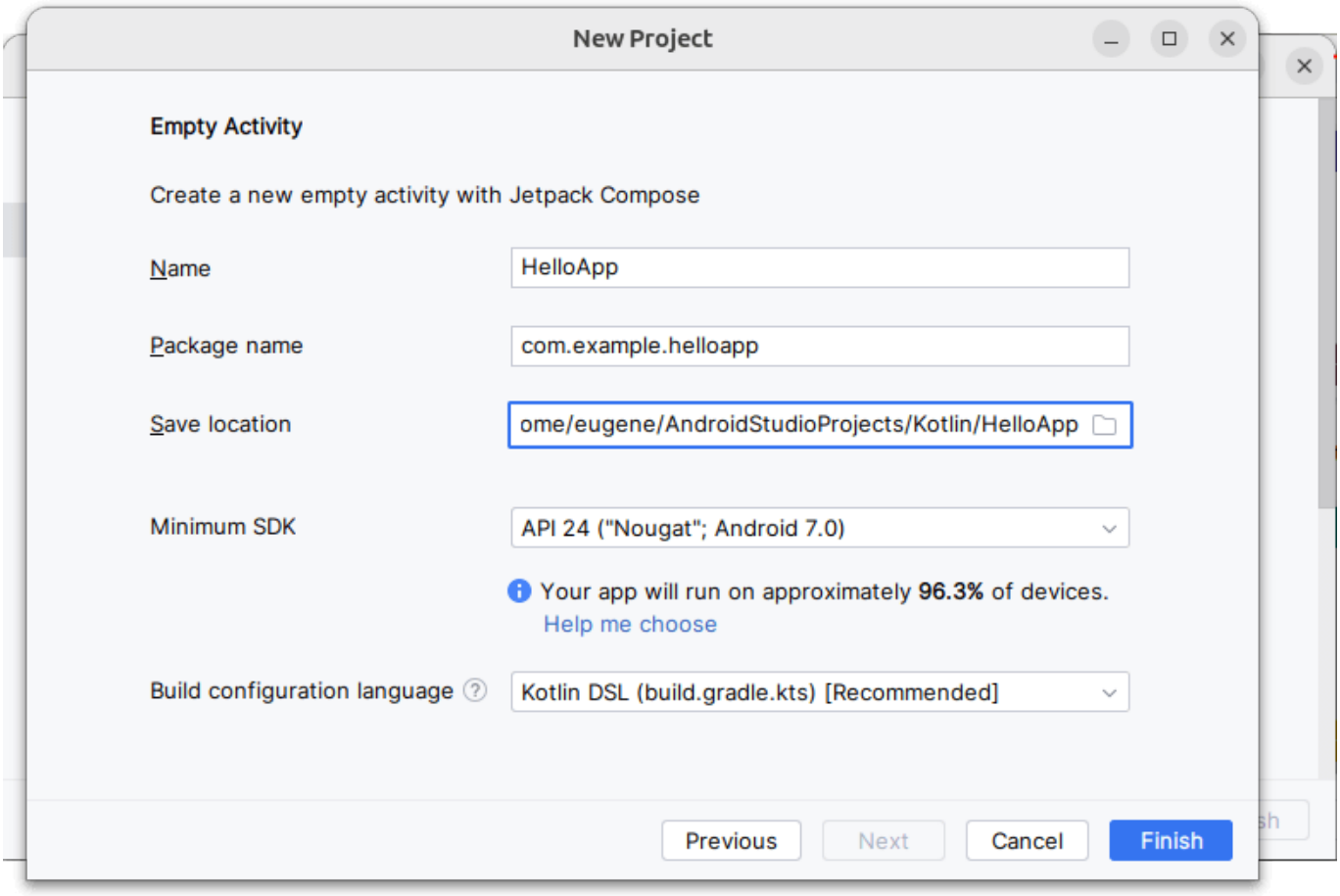
Если мы находимся в окне Welcome to Android Studio, то для создания проекта нажмем на кнопку New Project. Если в Android Studio уже открыт какой-то проект, то выберем в меню File -> New -> New Project.



Далее в открывшемся окне шаблонов проекта выберем пункт Empty Compose Activity и нажмем на кнопку Next.



Затем нам откроется окно настроек проекта:

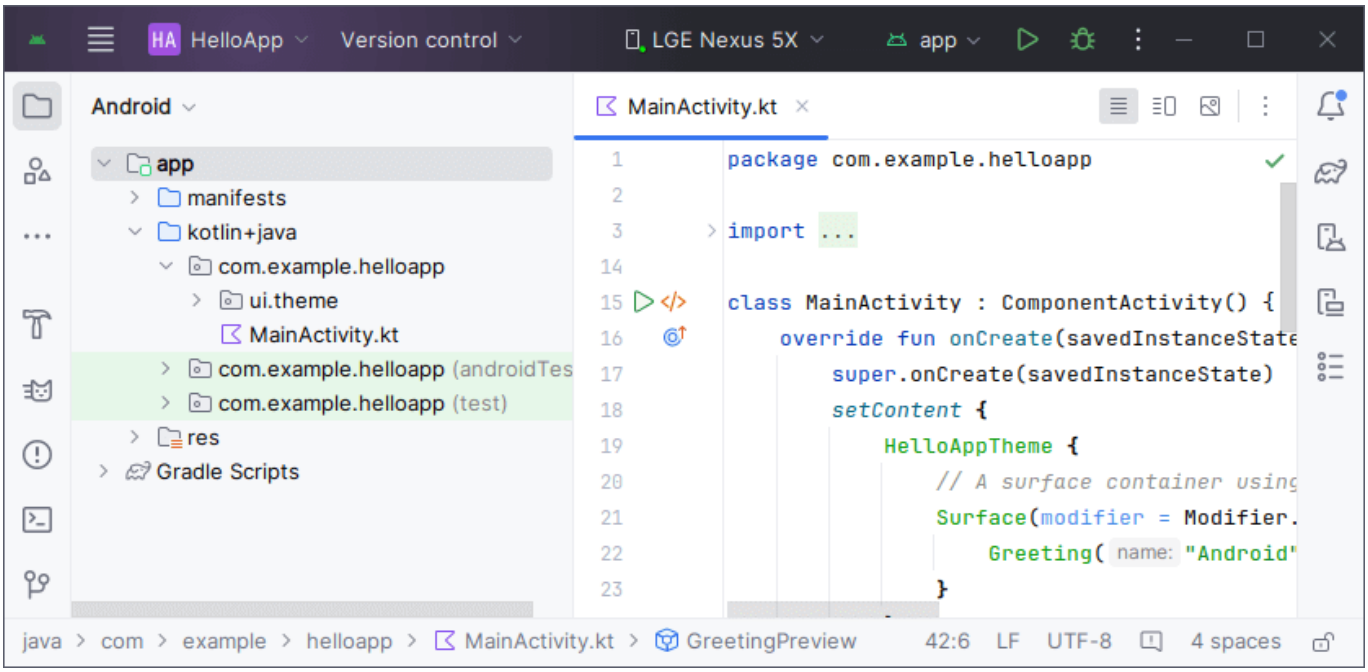


В этом окне мы можем установить начальные настройки проекта:

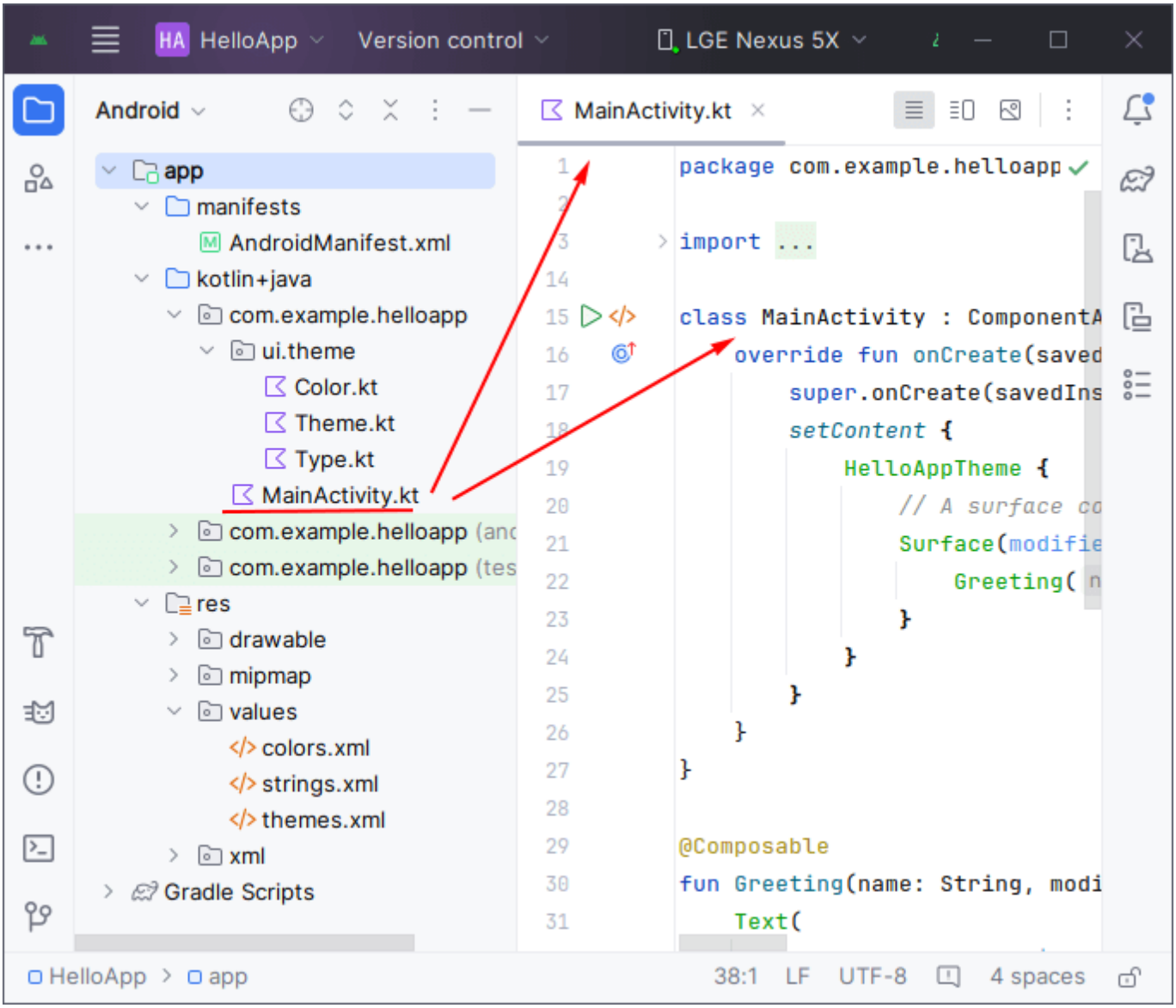
- В поле Name вводится название приложения. Укажем в качестве имени название HelloApp
- В поле Package Name указывается имя пакета, где будет размещаться главный класс приложения. В данном случае для тестовых проектов это значение не играет ольшого значения, поэтому установим com.example.helloapp.
- В поле Save Location установливается расположение файлов проекта на жестком диске. Можно оставить значение по умолчанию.
- В поле Minimum SDK указывается самая минимальная поддерживаемая версия SDK. Оставим значение по умолчанию - API 21: Android 5.0 (Lollipop), которая означает, что наше приложение можно будет запустить начиная с Android 5.0, а это 94% устройств. На более старых устройствах запустить будет нельзя.

Стоит учитывать, что чем выше версия SDK, тем меньше диапазон поддерживаемых устройств.

Далее нажмем на кнопку Finish, и Android Studio создаст новый проект:



Вначале вкратце рассмотрим структуру проекта, что он уже имеет по умолчанию



Проект Android может состоять из различных модулей. По умолчанию, когда мы создаем проект, создается один модуль - app. Модуль по умолчанию включает три компонента:

- manifests: хранит файл манифеста AndroidManifest.xml, который описывает конфигурацию приложения и определяет каждый из компонентов данного приложения.
- java: хранит файлы кода на языке Kotlin, которые структурированы по отдельным пакетам. Так, в папке com.example.helloapp (название которого было указано на этапе создания проекта) имеется по умолчанию файл MainActivity.kt с кодом на языке Kotlin, который представляет класс MainActivity, запускаемый по умолчанию при старте приложения

Также в этой папке определен подкаталог ui.theme, который содержит ряд файлов на языке Kotlin: Color.kt, Shape.kt, Theme.kt и Type.kt, которые определяют ряд вспомогательных типов, функций, переменных, применяемых для создания графического интерфейса.

- res: содержит используемые в приложении ресурсы. Все ресурсы разбиты на подпапки.
    - папка drawable предназначена для хранения изображений, используемых в приложении
    - папки mipmap содержат файлы изображений, которые предназначены для создания иконки приложения при различных разрешениях экрана.
    - папка values хранит различные xml-файлы, содержащие коллекции ресурсов - различных данных, которые применяются в приложении. По умолчанию здесь есть два файла и одна папка:
        - файл colors.xml хранит описание цветов, используемых в приложении
        - файл strings.xml содержит строковые ресурсы, используемые в приложении
        - папки themes хранит две темы приложения - для светлую (дневную) и темную (ночную)

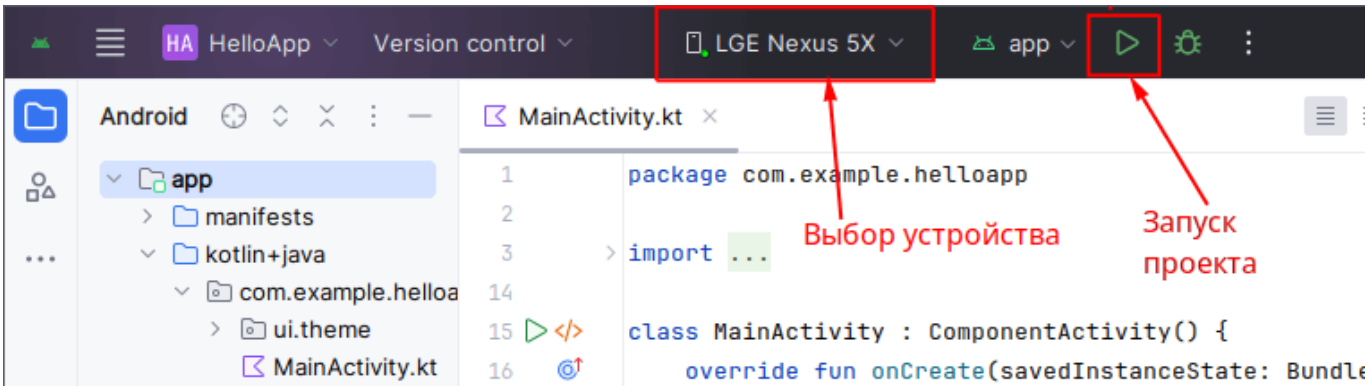
Отдельный элемент Gradle Scripts содержит ряд скриптов Gradle, которые используются при построении приложения.

Во всей этой структуре следует выделить файл MainActivity.kt, который открыт в Android Studio и который содержит логику приложения и собственно с него начинается выполнение приложения.

Запуск проекта

Теперь запустим созданный выше проект. Для запуска и тестирования проекта можно использовать как различные эмуляторы, так и реальные устройства. Для тестирования на реальном устройстве на нем должен быть включен режим разработчика. В моем случае я запущу проект на реальном устройстве.

Для запуска проекта нажмем на зеленую стрелочку на панели инструментов.



И после успешного запуска на устройстве/емуляторе мы сможем насладиться надписью "Hello Android":

Создание визуального интерфейса

При создании проекта, который использует Jetpack Compose, в Android Studio, в проект по умолчанию добавляется файл MainActivity.kt, который содержит одноименный класс MainActivity. Этот класс определяет интерфейс, который мы видим на устройстве/эмуляторе при запуске проекта. Так, по умолчанию графический интерфейс будет выглядеть будет следующим образом:

Рассмотрим основные моменты кода, который добавляется в файл MainActivity.kt и который определяет подобный интерфейс:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.material.MaterialTheme
import androidx.compose.material.Surface
import androidx.compose.material.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.tooling.preview.Preview
import com.example.helloapp.ui.theme.HelloAppTheme

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            HelloAppTheme {
                Surface(color = MaterialTheme.colors.background) {
                    Greeting("Android")
                }
            }
        }
    }
```

```
}

@Composable
fun Greeting(name: String) {
    Text(text = "Hello $name!")
}

@Preview(showBackground = true)
@Composable
fun DefaultPreview() {
    HelloAppTheme {
        Greeting("Android")
    }
}
```

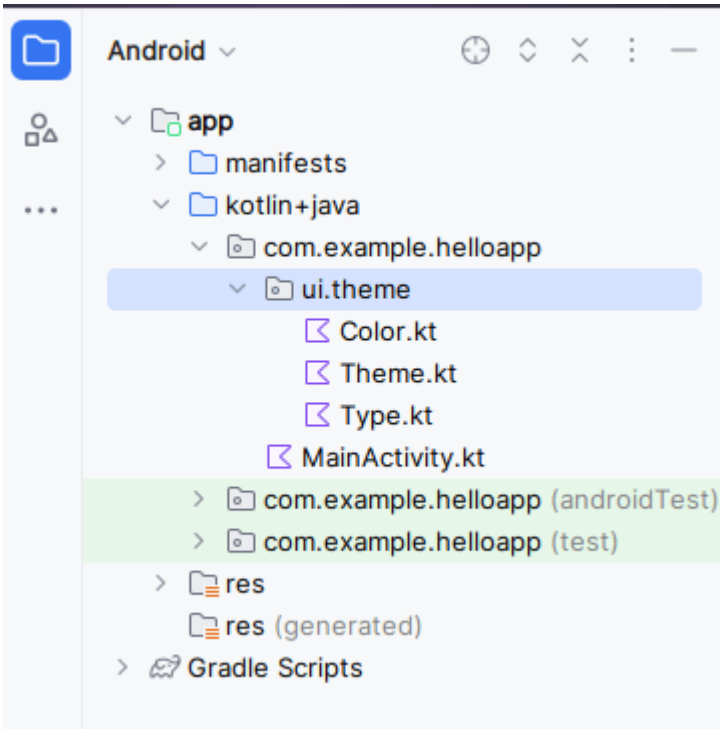
Рассмотрим основные моменты этого кода вкратце. Вначале идет определение пакета, к которому принадлежит класс MainActivity. В моем случае это пакет com.example.helloapp:

```
package com.example.helloapp
```

Далее идут подключаемые пакеты, функционал который использует класс MainActivity:

```
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.material.MaterialTheme
import androidx.compose.material.Surface
import androidx.compose.material.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.tooling.preview.Preview
import com.example.helloapp.ui.theme.HelloAppTheme
```

Обратите внимани на последний подключаемый пакет - он представляет функционал из каталога ui.theme, который также добавляется в проект по умолчанию и который содержит ряд файлов на языке Kotlin: Color.kt, Shape.kt, Theme.kt и Type.kt.



Далее идет определение класса MainActivity

```
class MainActivity : ComponentActivity() {
```

MainActivity наследуется от встроенного класса ComponentActivity. ComponentActivity обеспечивает построение интерфейса из визуальных компонентов и для этого предоставляет минимальный функционал. В частности, ComponentActivity предоставляет метод onCreate(), который вызывается при запуске приложения и создает интерфейс этого приложения.

```
override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent { ......
```

В метод передается объект Bundle, который хранит состояние MainActivity - некоторые значения, которые хранят связанные с MainActivity данные. А в самом методе onCreate() вначале вызывается реализация этого метода из базового класса ComponentActivity.

Для собственно создания интерфейса в onCreate() вызывается другой метод базового класса ComponentActivity - метод setContent(). Этот метод собственно и определяет, какой интерфейс мы увидим на экране устройства. В этот метод передается компонент, предваряемый аннотацией @Composable:

```
setContent {
    HelloAppTheme {
        Surface(color = MaterialTheme.colors.background) {
            Greeting("Android")
        }
```

```
    }
}
```

Что такое компонент @Composable? Объект или функция @Composable представляет центральную концепцию фреймворка Jetpack Compose и, грубо говоря, представляет некотоый визуальный компонент. Причем этот компонент должен быть оформлен в виде функции с аннотацией @Composable. Так, в данном случае это функция HelloAppTheme, которая определена в проекте в файле Theme.kt:

```
@Composable
fun HelloAppTheme(darkTheme: Boolean = isSystemInDarkTheme(), content:
@Composable() () -> Unit) {
    val colors = if (darkTheme) {
        DarkColorPalette
    } else {
        LightColorPalette
    }

    MaterialTheme(
        colors = colors,
        typography = Typography,
        shapes = Shapes,
        content = content
    )
}
```

Эта функция кроме того, что задает визуальный интерфейс, также обеспечивает соответствие приложения текущей теме (светлой или темной) устройства. Для этого она в качестве первого параметра принимает в качестве первого параметра булевое значение, которое указывает, выбрана ли темная тема. В зависимости от значения этого параметра устанавливает соответствующие цвета, которые определены в файле Color.kt. Второй параметр представляет еще один объект @Composable - то есть опять же некоторый визуальный компонент. Далее этот компонент передает в объект MaterialTheme.

Объект MaterialTheme задает визуальный интерфейс в стиле Material Design. Для этого он использует также настройки шрифта в виде объекта Typography из файла Type.kt и настройки формы в виде объекта Shapes из файла Shape.kt

Можно сказать, что фактически за объектом HelloAppTheme прячется объект MaterialTheme, которой устанавливает дизайн в стиле Material Design. Однако что именно передается в сам MaterialTheme?

Если мы вернемся к файлу MainActivity.kt, то увидим что это объект Surface:

```
Surface(color = MaterialTheme.colors.background) {
    Greeting("Android")
}
```

Surface фактически представляет промежуточный компонент, который задает дополнительное оформление в стиле Material Design. В этот компонент передается компонент Greeting, который определен в том же файле:

```
@Composable
fun Greeting(name: String) {
    Text(text = "Hello $name!")
}
```

Этот компонент использует другой встроенный компонент - Text, который представляет некоторый текст. Именно этот текст в итоге мы увидим на экране устройства.

И в данном случае мы видим, что для вывода простой надписи "Hello Android" используется целый набор компонентов @Composable, которые вложены в друг друга по принципу матрешки: Text -> Greeting -> Surface -> MaterialTheme -> HelloAppTheme

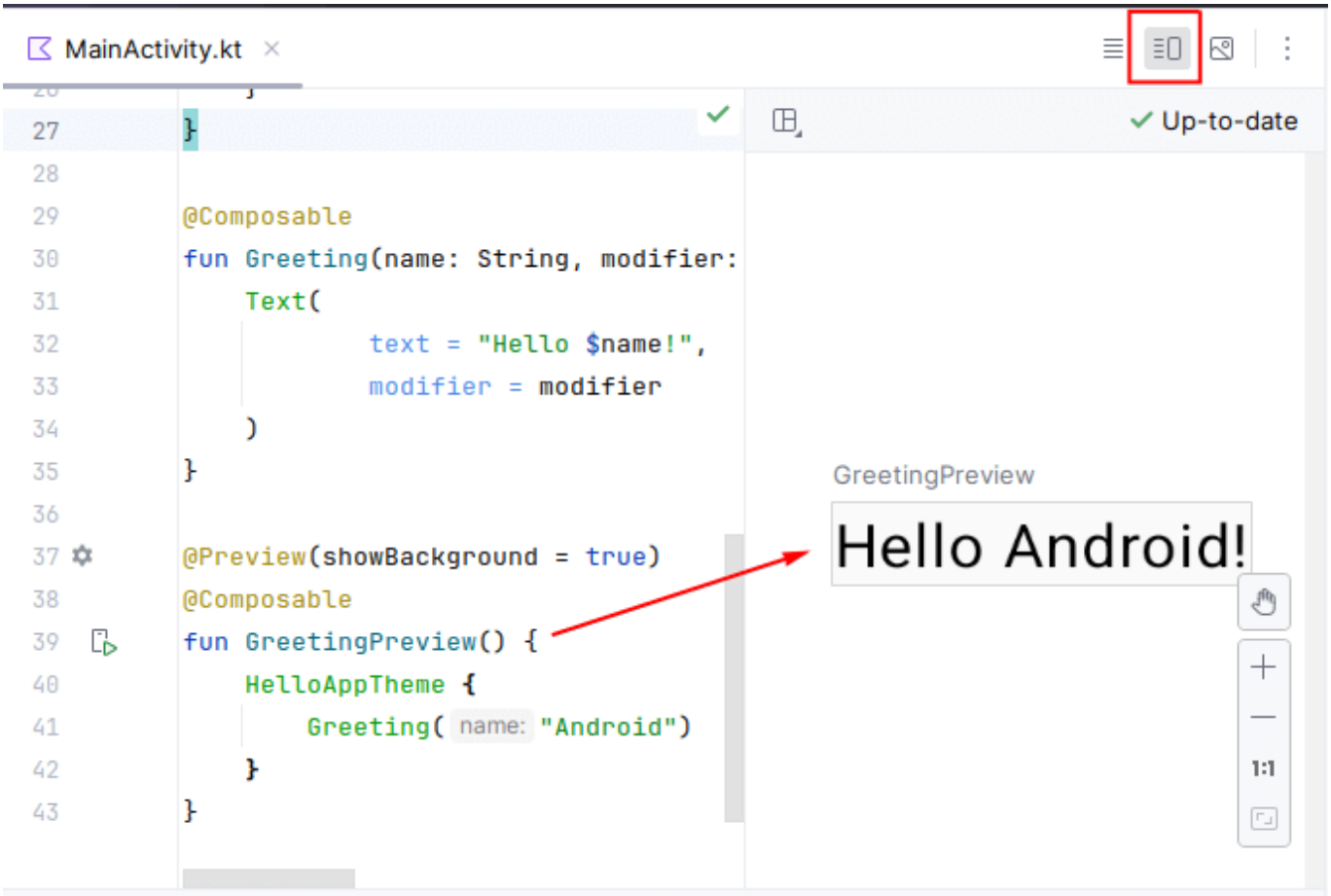
Предварительный вид компонентов

Однако выше не была упомянута еще одна часть кода MainActivity.kt - функция DefaultPreview:

```
@Preview(showBackground = true)
@Composable
fun DefaultPreview() {
    HelloAppTheme {
        Greeting("Android")
    }
}
```

Здесь мы видим, что DefaultPreview - это также аннотацию @Composable, и соответственно может представлять некоторую часть визуального интерфейса. И в реальности он скрывает компонент HelloAppTheme - то есть фактически весь тот же самый интерфейс, что и определен в классе MainActivity. Но важная часть определения этого компонента - аннотация @Preview. Она указывает, что данный компонент будет применяться для предварительного просмотра. То есть, чтобы узнать, как будет выглядеть наш интерфейс, нам необязательно запускать приложение на устройстве/эмуляторе. В случае небольших, но частых изменений это может быть довольно утомительно. А с помощью этого компонента мы можем увидить, что это будет за интерфейс.

Весь интерфейс в компоненте DefaultPreview можно увидеть в Android Studio при просмотре в режиме Design или Split:



Установка корневого компонента

Однако нам в принципе необязательно использовать всю эту комплексную композицию компонентов. Так, в примере выше, если убрать настройку темы приложения, то по сути все, что оно делает - это выводит некоторый текст на экран устройства. И в реальности мы могли бы в этом случае ограничиться одним компонентом Text. Так, изменим код MainActivity.kt следующим образом:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.material.Text
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.unit.sp

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            Text(text = "Hello METANIT.COM!",
                style = TextStyle(
                    fontSize = 22.sp
                )
            )
```

```
            }
        }
    }
```

Компонент, который представляет по сути то, что мы увидим на экране устройства, передается в метод setContent() класса MainActivity. И в данном случае в этот метод сразу передается компонент Text.

У этого компонента устанавливаются в данном случае два свойства. Свойство text представляет выводимый текст. Кроме того, здесь устанавливается свойство style, которое задает стиль текста - по сути как текст будет выглядеть. В качестве значения оно принимает объект TextStyle.

Класс TextStyle позволяет установить самые разные настройки текста. Здесь же устанавливается только свойство fontSize, которое задает размер шрифта. В качестве значения оно принимает числовое значение в единицах sp.

В итоге приложение будет выглядеть следующим образом: