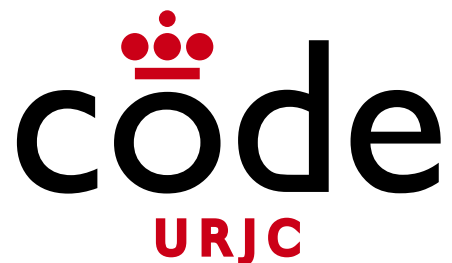


Desarrollo de Aplicaciones para Dispositivos
Móviles

Bloque II: Desarrollo Nativo

**Tema 2.2: Introducción a Kotlin y
Jetpack**



©2023

Michel Maes

Algunos derechos reservados

Este documento se distribuye bajo la licencia
“Atribución-CompartirIgual 4.0 Internacional”
de Creative Commons Disponible en
<https://creativecommons.org/licenses/by-sa/4.0/deed.es>

Definición

- Lenguaje de programación basado en la JVM
- Desarrollado por JetBrains (IntelliJ IDEA) en 2012
- En 2017 es adoptado por Google como lenguaje oficial de Android
- Claro, conciso, interoperable con Java y diseñado para la concurrencia

Sintaxis

- Es un lenguaje tipado y orientado a objetos

```
var count: Int = 10
```

- El lenguaje trata de simplificar el código de manera compacta

```
if (count == 42) {  
    println("I have the answer.")  
} else if (count > 35) {  
    println("The answer is close.")  
} else {  
    println("The answer eludes me.")  
}
```



```
val answerString: String = if (count == 42) {  
    "I have the answer."  
} else if (count > 35) {  
    "The answer is close."  
} else {  
    "The answer eludes me."  
}  
println(answerString)
```

Sintaxis

- Por defecto, no permite que una variable sea *nulla*

```
var name: String = null
```

- Se puede definir una variable como *anulable*

```
var name: String? = null
```

Sintaxis

- Veremos ventajas de su sintaxis más adelante
- Para aprender Kotlin:

Básico: <https://developer.android.com/kotlin/learn>

Aplicado a Android: <https://developer.android.com/kotlin/common-patterns>

Documentación completa: <https://kotlinlang.org/docs/getting-started.html>

Introducción a Jetpack Compose

Jetpack Compose

```
@Composable
fun JetpackCompose() {
    Card {
        var expanded by remember { mutableStateOf(false) }
        Column(Modifier.clickable { expanded = !expanded }) {
            Image(painterResource(R.drawable.jetpack_compose))
            AnimatedVisibility(expanded) {
                Text(
                    text = "Jetpack Compose",
                    style = MaterialTheme.typography.h2,
                )
            }
        }
    }
}
```



<https://developer.android.com/jetpack>

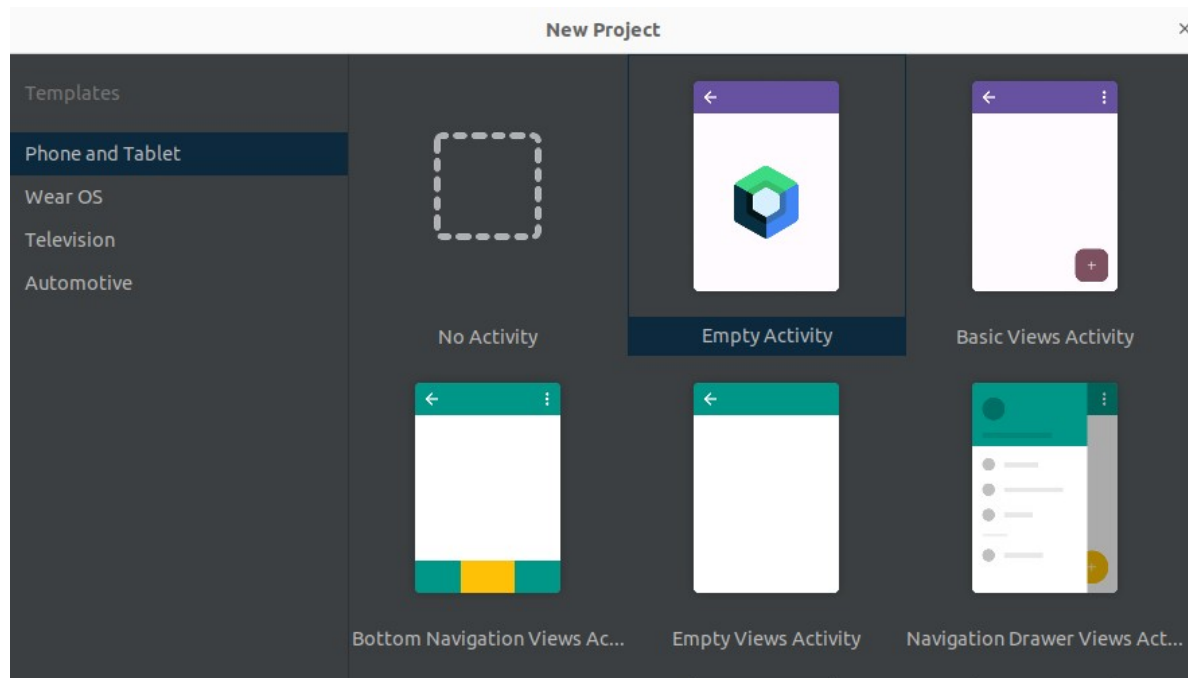
Definición

- Framework (*marco de trabajo/conjunto de librerías/conjunto de prácticas*) para el desarrollo Android
- Oficial de Android, mantenido por Google
- Versión 1.0 lanzada en 2021
- Completamente integrado con Android Studio

Introducción a Jetpack Compose

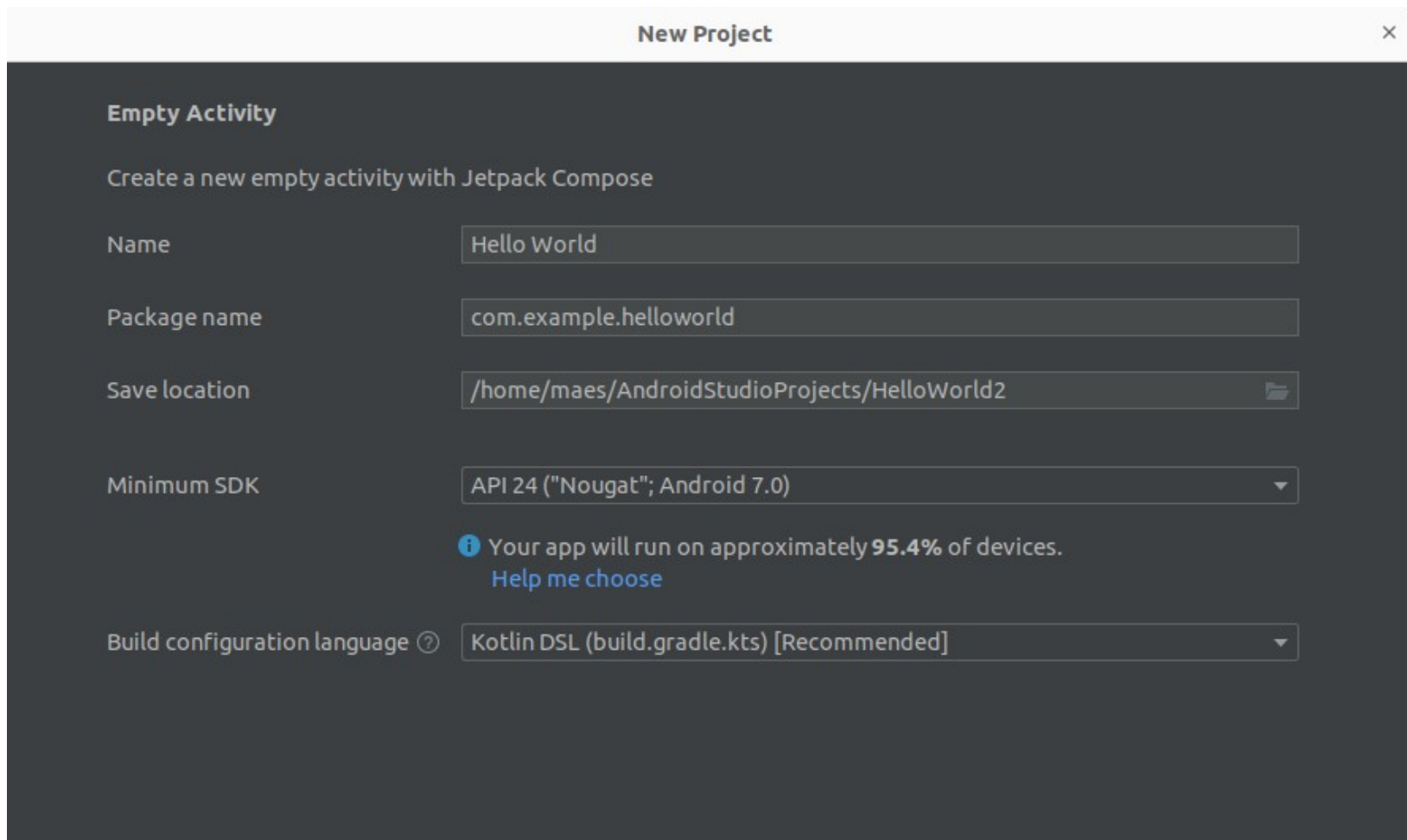
Creación de un nuevo proyecto Jetpack

- Abrimos Android Studio
- File > New > New project > Empty Activity



Introducción a Jetpack Compose

Creación de un nuevo proyecto Jetpack

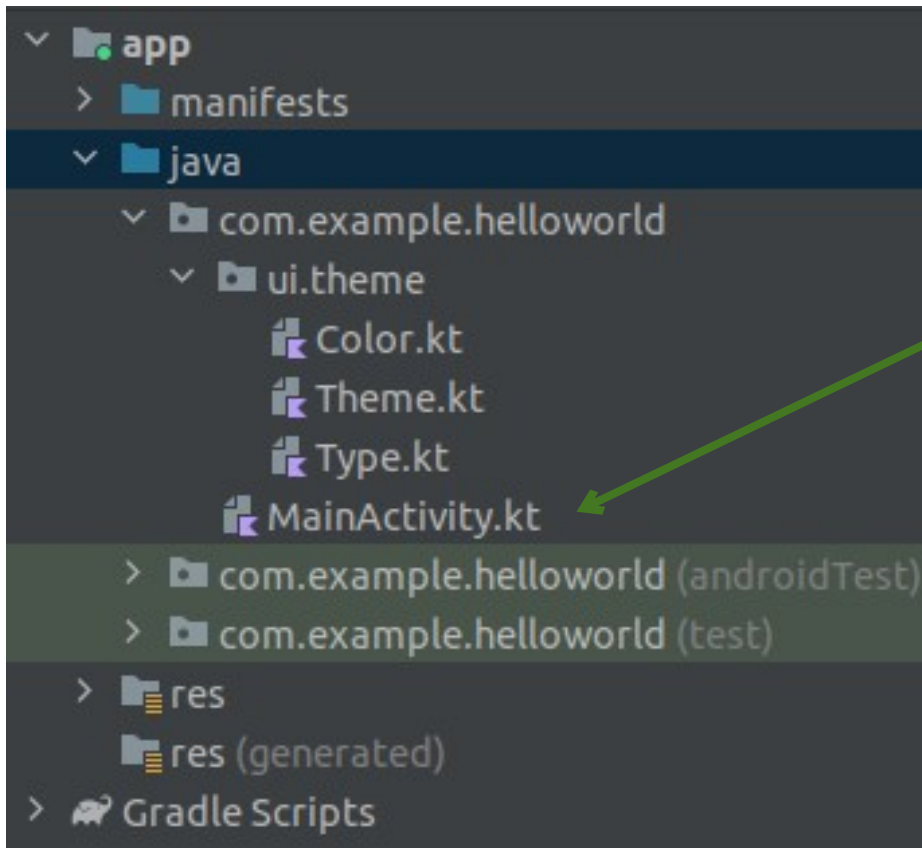


The screenshot shows the 'New Project' dialog in Android Studio. The dialog has a title bar with 'New Project' and a close button. The main content area is dark gray and contains the following fields and options:

- Empty Activity**: A section header.
- Create a new empty activity with Jetpack Compose**: A descriptive text.
- Name**: A text field containing 'Hello World'.
- Package name**: A text field containing 'com.example.helloworld'.
- Save location**: A text field containing '/home/maes/AndroidStudioProjects/HelloWorld2' with a folder icon on the right.
- Minimum SDK**: A dropdown menu showing 'API 24 ("Nougat"; Android 7.0)'.
- Information**: A blue information icon followed by the text 'Your app will run on approximately 95.4% of devices.' and a link 'Help me choose'.
- Build configuration language**: A dropdown menu showing 'Kotlin DSL (build.gradle.kts) [Recommended]' with a question mark icon on the left.

Introducción a Jetpack Compose

Estructura de carpetas



Clase principal
MainActivity.kt

Introducción a Jetpack Compose

MainActivity.tk

```
package com.example.helloworld

import ...

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            HelloWorldTheme {
                // A surface container using the 'background' color attribute
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colorScheme.background
                ) {
                    Greeting("World")
                }
            }
        }
    }
}

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

MainActivity hereda de ComponentActivity, que representa una **Actividad**, que se podría entender como una pantalla que el usuario ve

Introducción a Jetpack Compose

MainActivity.tk

```
package com.example.helloworld

import ...

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            HelloWorldTheme {
                // A surface container using the 'background' color attribute
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colorScheme.background
                ) {
                    Greeting("World")
                }
            }
        }
    }
}

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

Es necesario que llame al constructor del padre para que la actividad se inicie correctamente

Introducción a Jetpack Compose

MainActivity.tk

```
package com.example.helloworld

import ...

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            HelloWorldTheme {
                // A surface container using the 'background' color
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colorScheme.background
                ) {
                    Greeting("World")
                }
            }
        }
    }
}

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

La función setContent recoge el contenido que se añadirá a la actividad

Introducción a Jetpack Compose

MainActivity.tk

```
package com.example.helloworld

import ...

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            HelloWorldTheme {
                // A surface container using the 'background' color attribute
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colorScheme.background
                ) {
                    Greeting("World")
                }
            }
        }
    }
}

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

El contenido vendrá recogido sobre un tema de diseño (*HelloWorldTheme*).

Introducción a Jetpack Compose

MainActivity.tk

```
package com.example.helloworld

import ...

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            HelloWorldTheme {
                // A surface container using the 'background' color attribute
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colorScheme.background
                ) {
                    Greeting("World")
                }
            }
        }
    }
}

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

Surface es un contenedor de otros elementos. Podemos pasarle propiedades como un modificador para fijar su tamaño o el color de fondo

Introducción a Jetpack Compose

MainActivity.tk

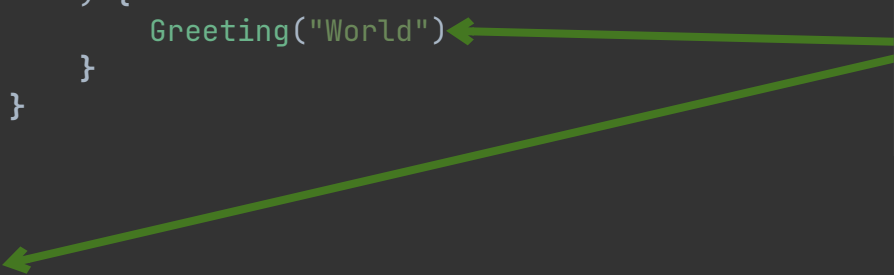
```
package com.example.helloworld

import ...

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            HelloWorldTheme {
                // A surface container using the 'background' color attribute
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colorScheme.background
                ) {
                    Greeting("World")
                }
            }
        }
    }
}

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

Dentro de Surface
podremos definir
componentes propios que
hayamos creado con la
anotación **@Composable**



Introducción a Jetpack Compose

MainActivity.tk

```
package com.example.helloworld

import ...

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            HelloWorldTheme {
                // A surface container using the 'background' color from the theme
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colorScheme.background
                ) {
                    Greeting("World")
                }
            }
        }
    }
}

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

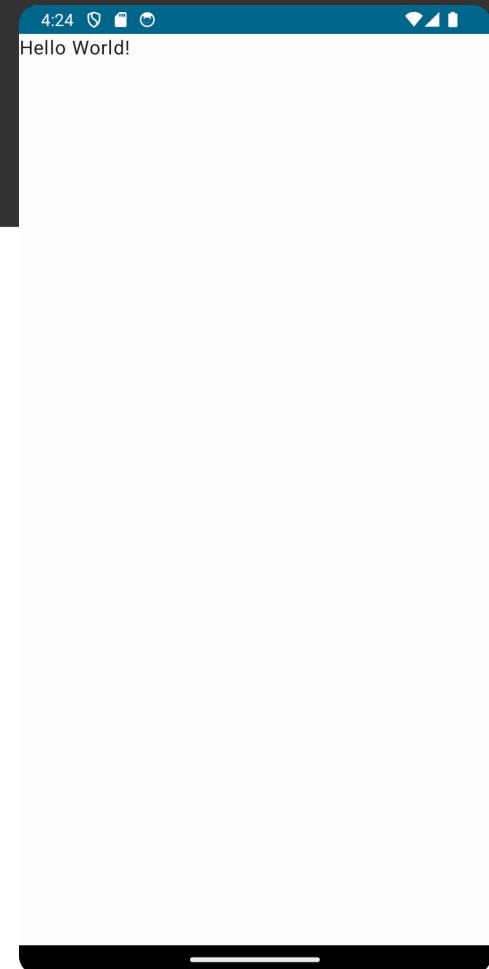
Nuestro componente
Greeting mostrará un
texto por pantalla con la
función Text()

Introducción a Jetpack Compose

MainActivity.tk

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

- Si ejecutamos la app utilizando el emulador (Run>Run app) veremos el resultado
- La llamada a Text(miTexto) es **declarativa**, nosotros configuramos el texto y el modificador (opcional).
- Jetpack Compose se ocupa de que se muestre como texto en pantalla

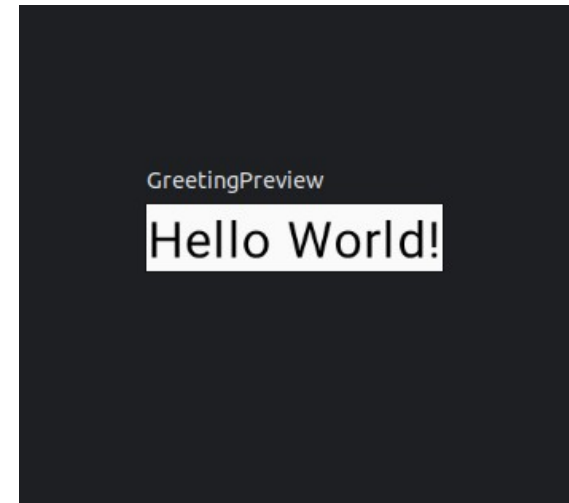


Introducción a Jetpack Compose

MainActivity.tk

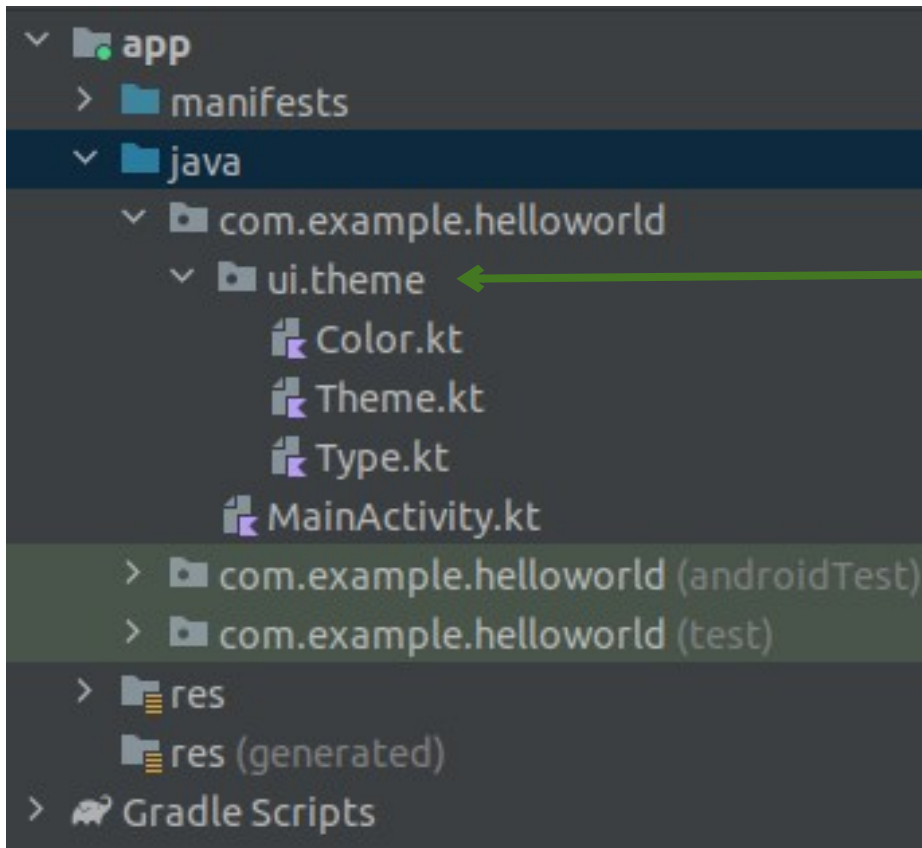
```
@Preview(showBackground = true)
@Composable
fun GreetingPreview() {
    HelloWorldTheme {
        Greeting("World")
    }
}
```

- A la hora de diseñar, recargar la app puede resultar pesado.
- Podemos crear un componente auxiliar (por ejemplo GreetingPreview) anotado con **@Preview** desde el que llamar a nuestros componentes
- Obtendremos una visualización en tiempo real de la edición



Introducción a Jetpack Compose

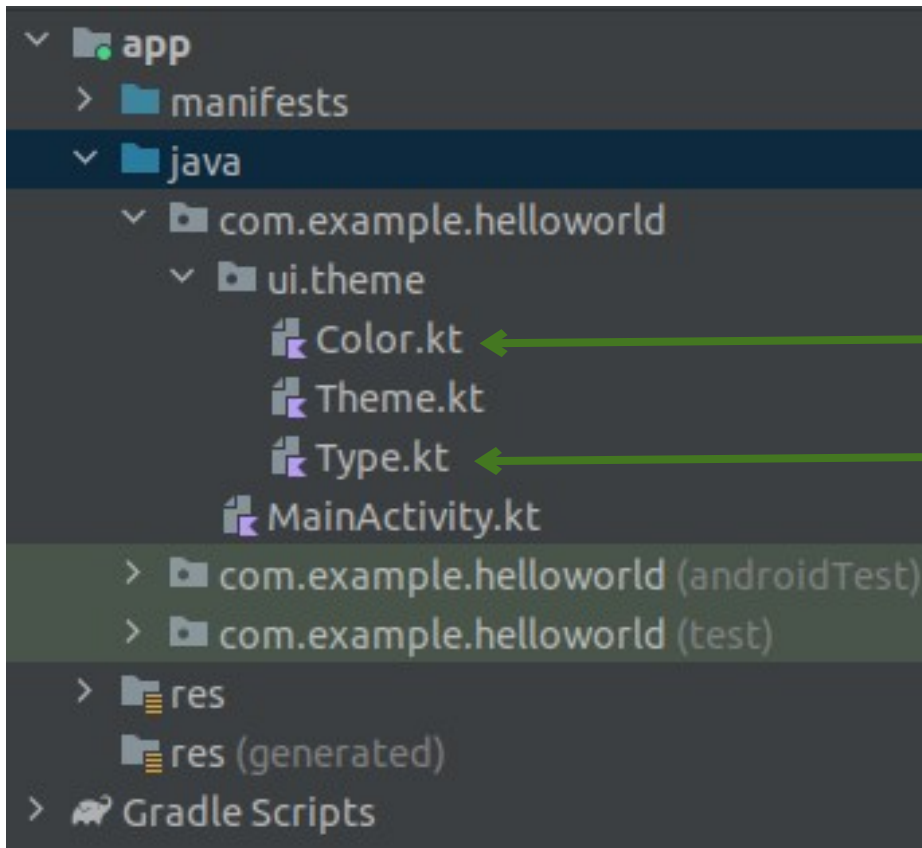
Estructura de carpetas



Carpeta con los
estilos de la
aplicación

Introducción a Jetpack Compose

Estructura de carpetas



Color.kt define los colores que se utilizarán en otras clase de estilo

Type.kt define la tipografía (fuentes) que se usarán en *Theme.kt*

Introducción a Jetpack Compose

ui.theme.Color.tk

```
package com.example.helloworld.ui.theme

import androidx.compose.ui.graphics.Color

val Purple80 = Color(0xFFD0BCFF)
val PurpleGrey80 = Color(0xFFCCC2DC)
val Pink80 = Color(0xFFE8BFD0)
val Purple40 = Color(0xFF66BB6A)
val PurpleGrey40 = Color(0xFF546E7A)
val Pink40 = Color(0xFF4FC3F7)
```

ui.theme.Theme.tk

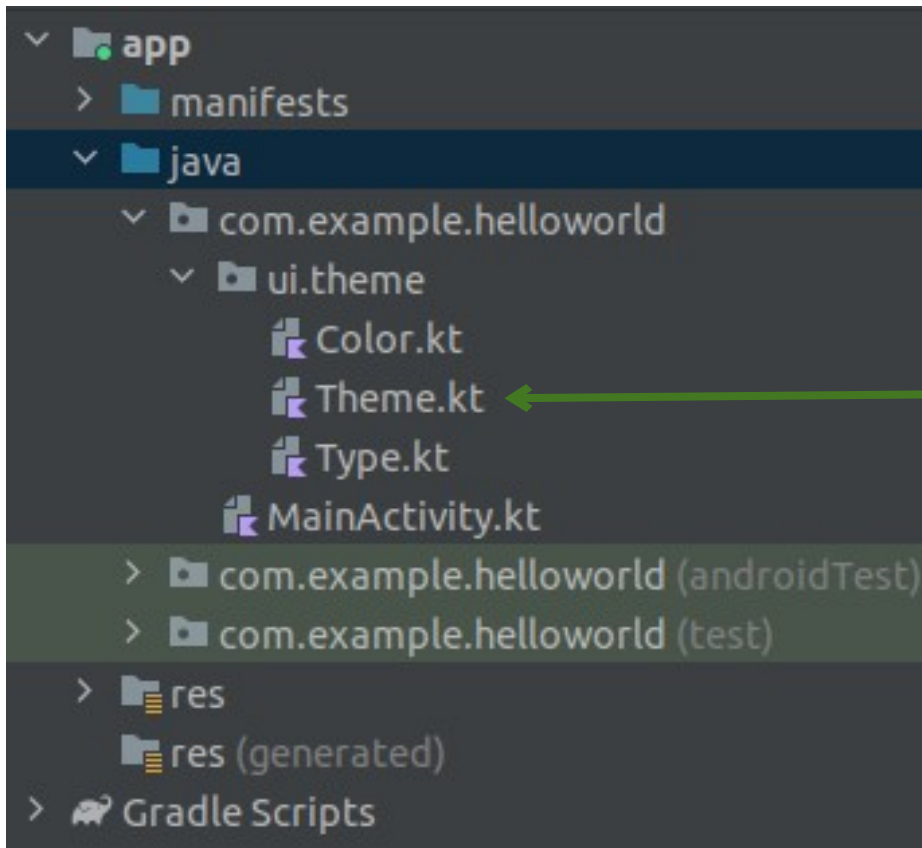
```
package com.example.helloworld.ui.theme

import ...

// Set of Material typography styles to start with
val Typography = Typography(
    bodyLarge = TextStyle(
        fontFamily = FontFamily.Default,
        fontWeight = FontWeight.Normal,
        fontSize = 16.sp,
        lineHeight = 24.sp,
        letterSpacing = 0.5.sp
    )
)
```

Introducción a Jetpack Compose

Estructura de carpetas



Theme.kt define los estilos que se utilizarán

Introducción a Jetpack Compose

ui.theme.Theme.tk

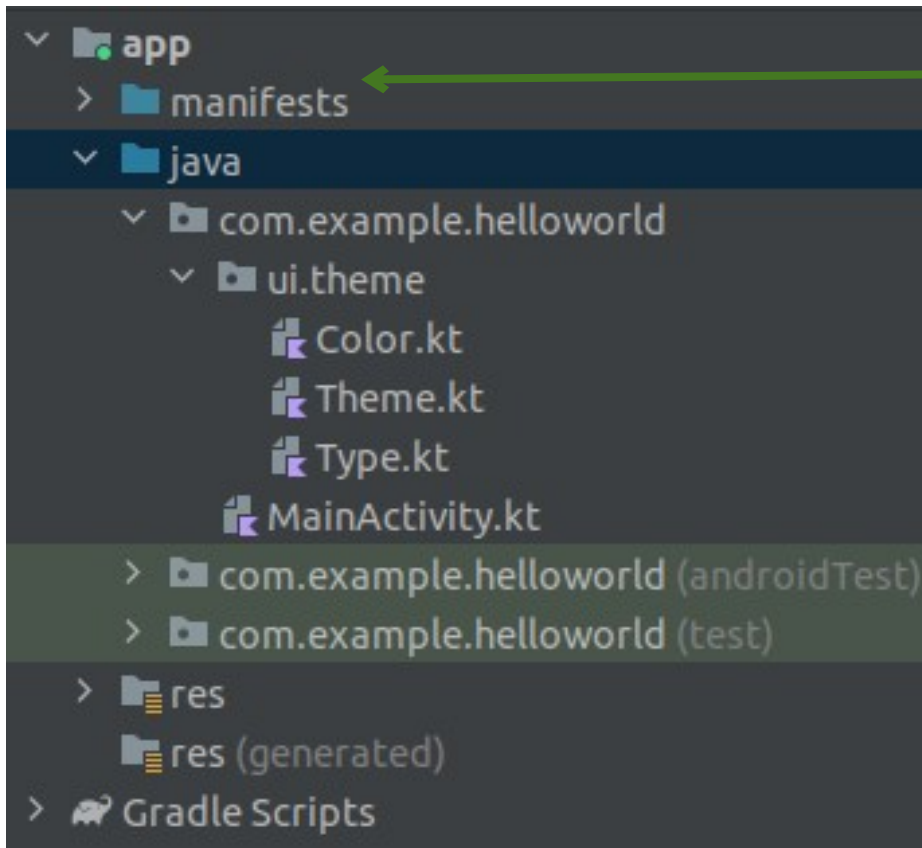
```
@Composable
fun HelloWorldTheme(
    darkTheme: Boolean = isSystemInDarkTheme(),
    dynamicColor: Boolean = true,
    content: @Composable () → Unit
) {
    val colorScheme = when {
        dynamicColor && Build.VERSION.SDK_INT ≥ Build.VERSION_CODES.S → {
            val context = LocalContext.current
            if (darkTheme) dynamicDarkColorScheme(context) else dynamicLightColorScheme(context)
        }
        darkTheme → DarkColorScheme
        else → LightColorScheme
    }
    val view = LocalView.current
    if (!view.isInEditMode) {
        SideEffect {
            val window = (view.context as Activity).window
            window.statusBarColor = colorScheme.primary.toArgb()
            WindowCompat.getInsetsController(window, view).isAppearanceLightStatusBars = darkTheme
        }
    }

    MaterialTheme(
        colorScheme = colorScheme,
        typography = Typography,
        content = content
    )
}
```

Un tema es un
@Composable que
aplicará los estilos a los
componentes que contenga

Introducción a Jetpack Compose

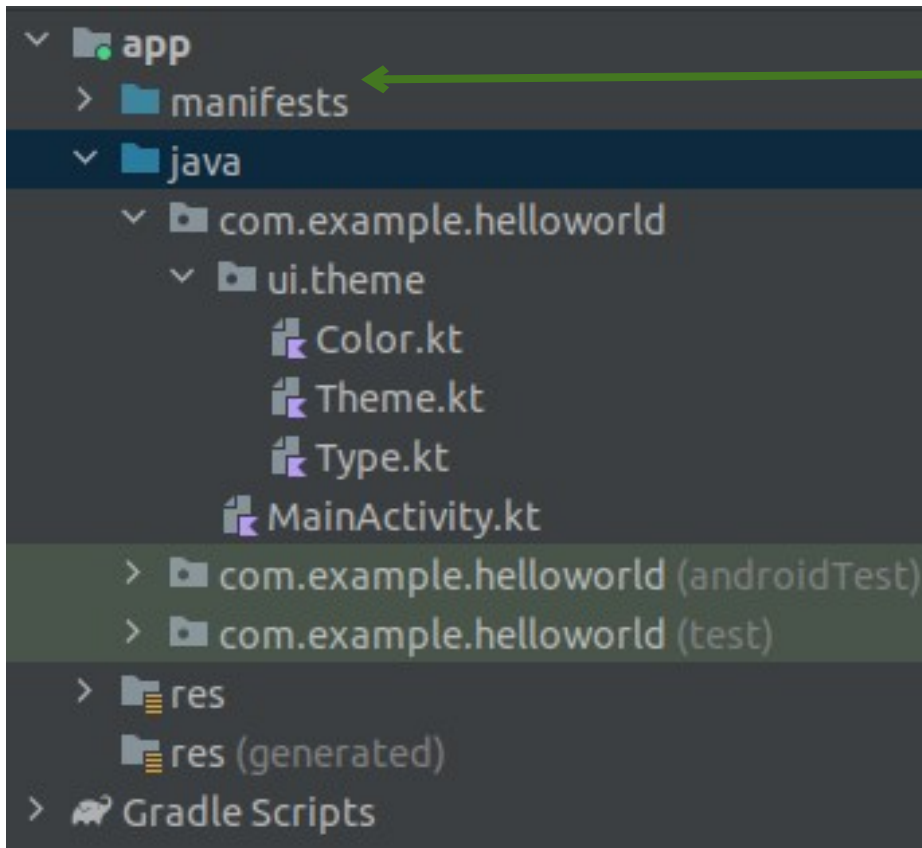
Estructura de carpetas



La carpeta *manifests* contiene el `AndroidManifest.xml`

Introducción a Jetpack Compose

Estructura de carpetas



La carpeta *manifests* contiene el `AndroidManifest.xml`

Estructura de carpetas

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

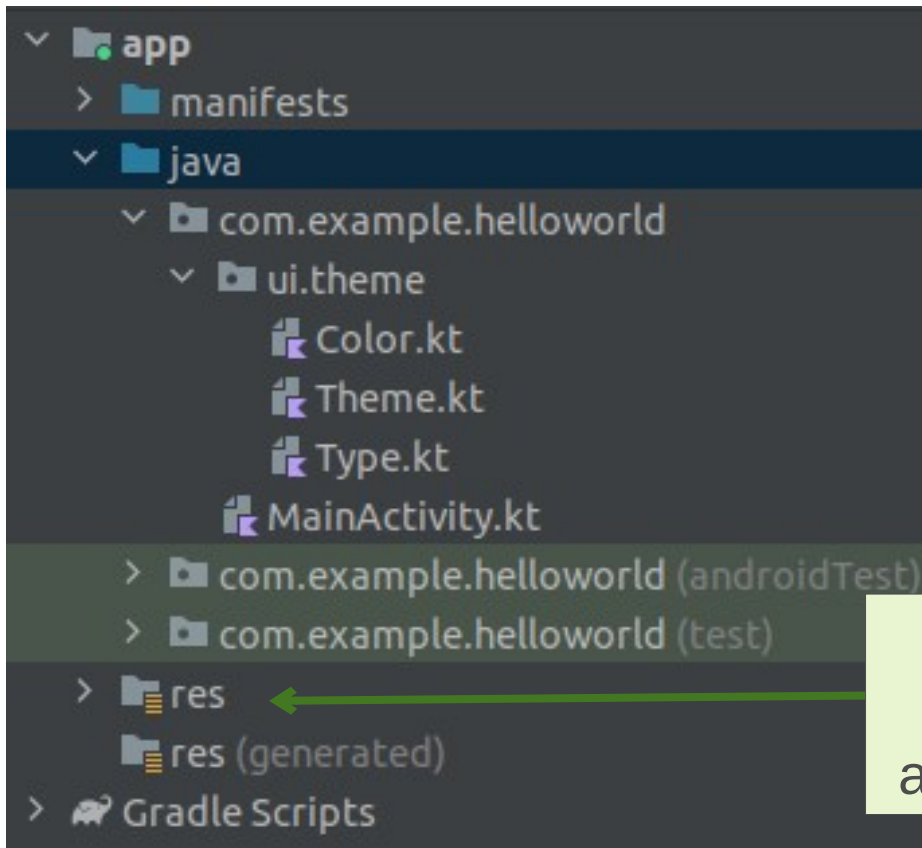
    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.HelloWorld"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:label="@string/app_name"
            android:theme="@style/Theme.HelloWorld">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
```

En el manifiesto se declara:

- El nombre del paquete
- Los componentes de la aplicación
- Permisos de la aplicación
- Funciones de hardware/software

Introducción a Jetpack Compose

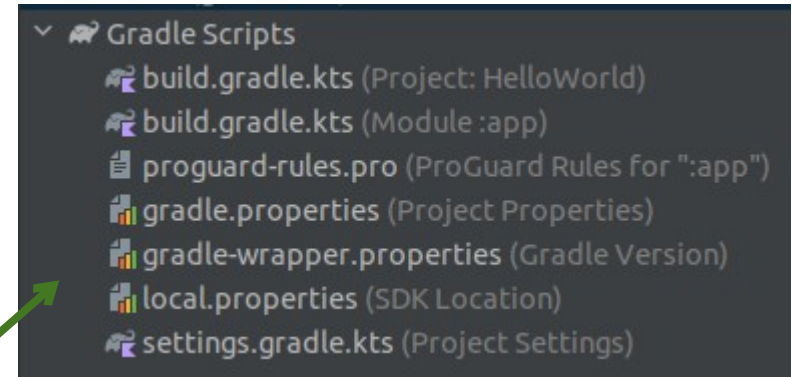
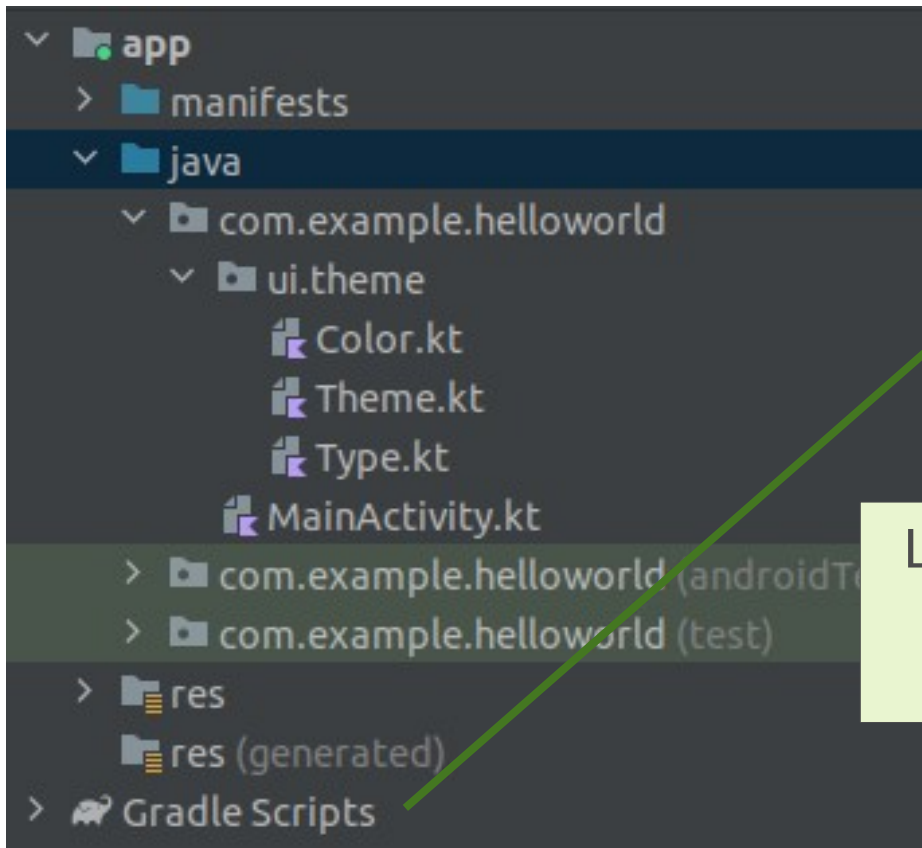
Estructura de carpetas



La carpeta *res* contiene cualquier asset que necesite nuestra aplicación (por ejemplo, imágenes)

Introducción a Jetpack Compose

Estructura de carpetas

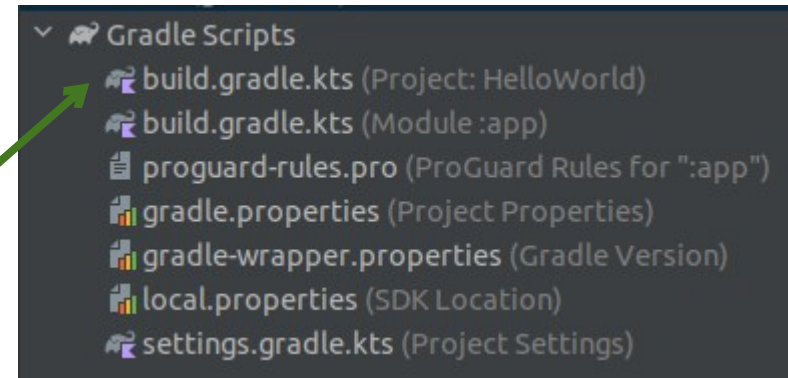


La carpeta Gradle Scripts contiene los ficheros de configuración de nuestro proyecto

Introducción a Jetpack Compose

Estructura de carpetas

build.gradle.kts (Project)
contiene los plugins utilizados a nivel de proyecto (por defecto, para Android y Kotlin)

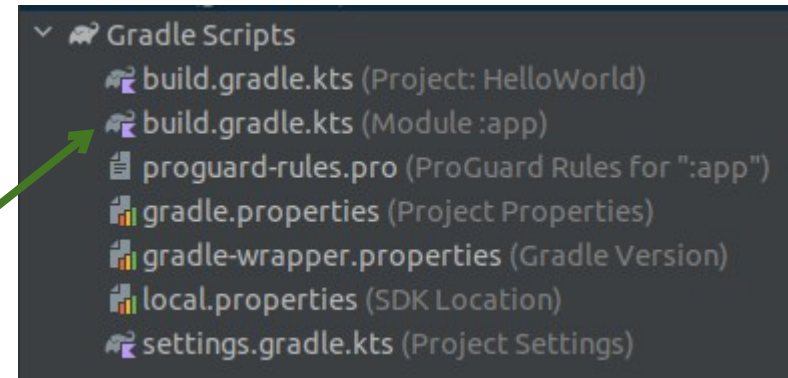


```
// Top-level build file where you can add configuration options common to all sub-projects/modules.  
plugins {  
    id("com.android.application") version "8.1.1" apply false  
    id("org.jetbrains.kotlin.android") version "1.8.10" apply false  
}
```

Introducción a Jetpack Compose

Estructura de carpetas

build.gradle.kts (Module)
contiene los plugins utilizados a nivel de modulo (por defecto, para Android y Kotlin)

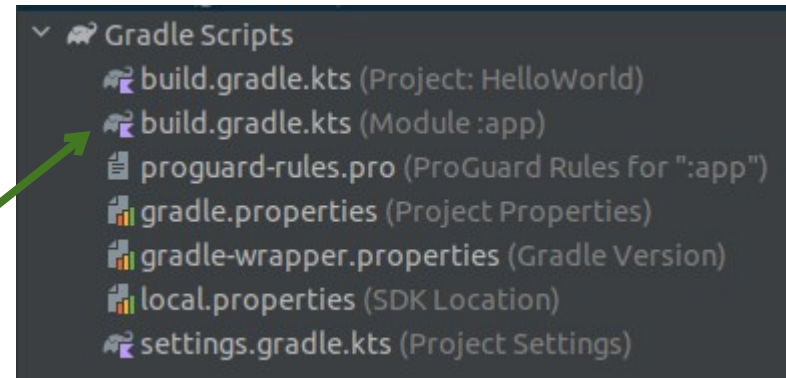


```
// Top-level build file where you can add configuration options common to all sub-projects/modules.  
plugins {  
    id("com.android.application") version "8.1.1" apply false  
    id("org.jetbrains.kotlin.android") version "1.8.10" apply false  
}
```


Introducción a Jetpack Compose

Estructura de carpetas

build.gradle.kts (Module)
también contiene la configuración
básica de nuestro proyecto
android junto a sus dependencias

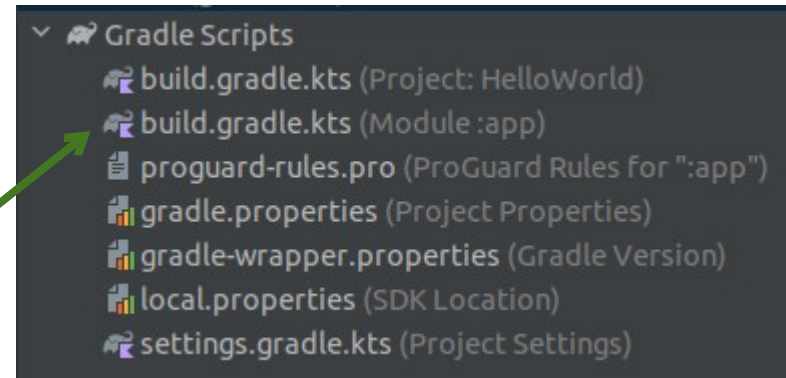


```
android {  
    namespace = "com.example.helloworld"  
    compileSdk = 33  
  
    defaultConfig {  
        applicationId = "com.example.helloworld"  
        minSdk = 24  
        targetSdk = 33  
        versionCode = 1  
        versionName = "1.0"  
  
        testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"  
        vectorDrawables {  
            useSupportLibrary = true  
        }  
    }  
}
```

Introducción a Jetpack Compose

Estructura de carpetas

build.gradle.kts (Module)
también contiene la configuración
básica de nuestro proyecto
android junto a sus dependencias

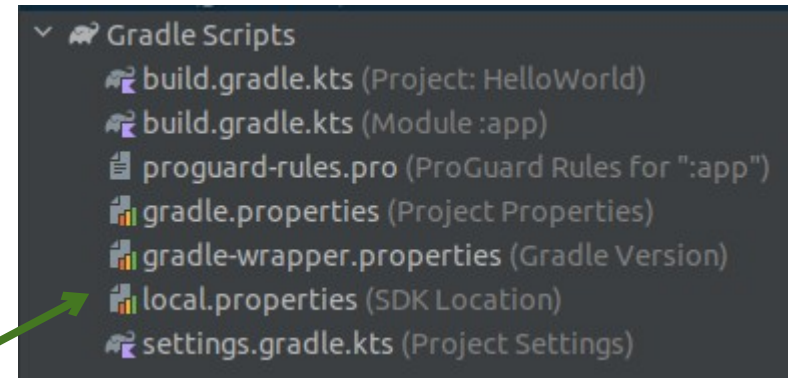


```
dependencies {  
    implementation("androidx.core:core-ktx:1.9.0")  
    implementation("androidx.lifecycle:lifecycle-runtime-ktx:2.6.2")  
    implementation("androidx.activity:activity-compose:1.7.2")  
    implementation(platform("androidx.compose:compose-bom:2023.03.00"))  
    implementation("androidx.compose.ui:ui")  
    implementation("androidx.compose.ui:ui-graphics")  
    implementation("androidx.compose.ui:ui-tooling-preview")  
    implementation("androidx.compose.material3:material3")  
    testImplementation("junit:junit:4.13.2")  
    androidTestImplementation("androidx.test.ext:junit:1.1.5")  
    androidTestImplementation("androidx.test.espresso:espresso-core:3.5.1")  
    androidTestImplementation(platform("androidx.compose:compose-bom:2023.03.00"))  
    androidTestImplementation("androidx.compose.ui:ui-test-junit4")  
}
```

Introducción a Jetpack Compose

Estructura de carpetas

local.properties es un fichero autogenerado por Android Studio con la ruta al SDK. Deberemos dejarlo fuera al trabajar de forma colaborativa (con Git por ejemplo)



```
## This file is automatically generated by Android Studio.
# Do not modify this file -- YOUR CHANGES WILL BE ERASED!
#
# This file should *NOT* be checked into Version Control Systems,
# as it contains information specific to your local configuration.
#
# Location of the SDK. This is only used by Gradle.
# For customization when using a Version Control System, please read the
# header note.
sdk.dir=/home/maes/Android/Sdk
```