



Jetpack Compose

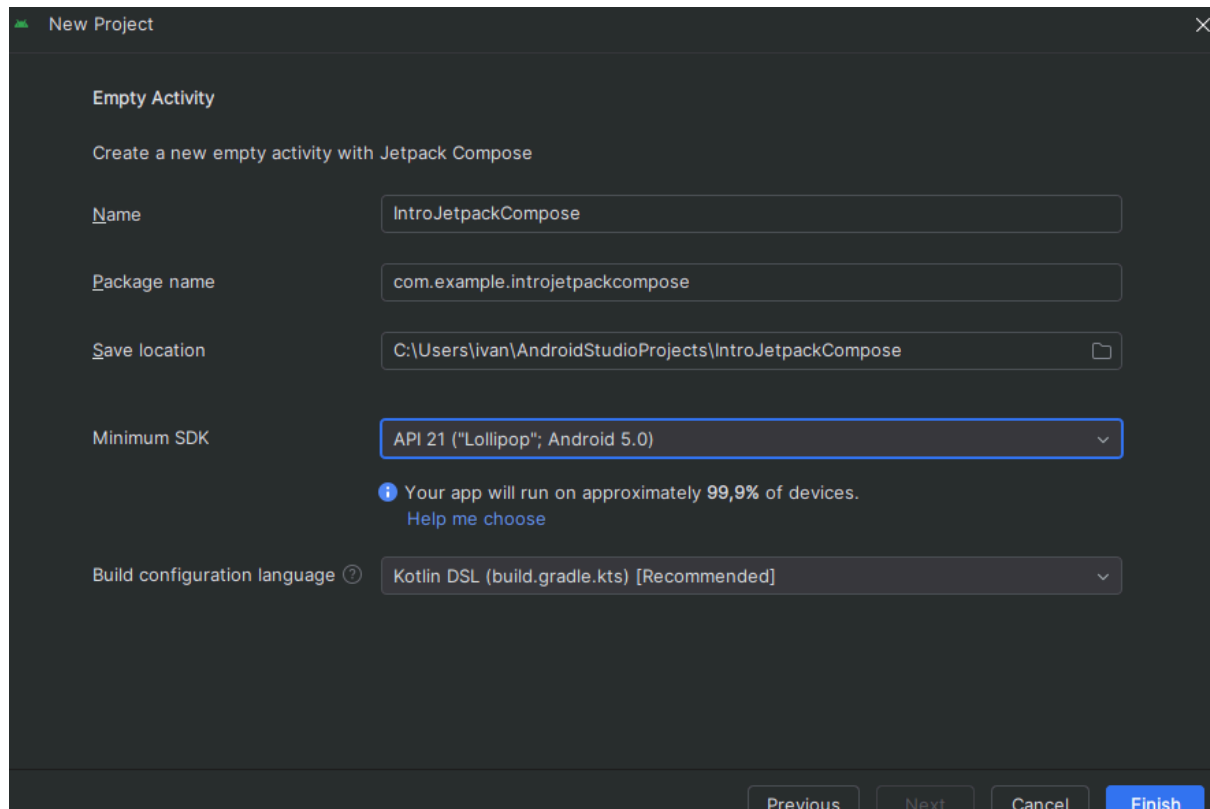
Índice

- 1- Creación del proyecto → página 3
- 2- Primer Composable → página 4
- 3- Vista Preview → página 5
- 4- Modificadores → página 7
- 5- Estructura del código → página 9
- 6- Previsualización avanzada → página 11

1- Creación del proyecto

Creamos el proyecto con una Empty Activity.
Ponemos el nombre que en mi caso será IntroJetpackCompose.

Ajustamos el Minimum SDK



2- Primer Composable

Nos vamos a [mainActivity.kt](#) y creamos la función composable saludar con un fontSize que será para ajustar el tamaño de la fuente

```
@Composable
fun Saludar(name: String, modifier: Modifier = Modifier){
    Text(
        text = "Hola $name!",
        fontSize = 10.sp,
        modifier = modifier
    )
}
```

3- Vista Preview

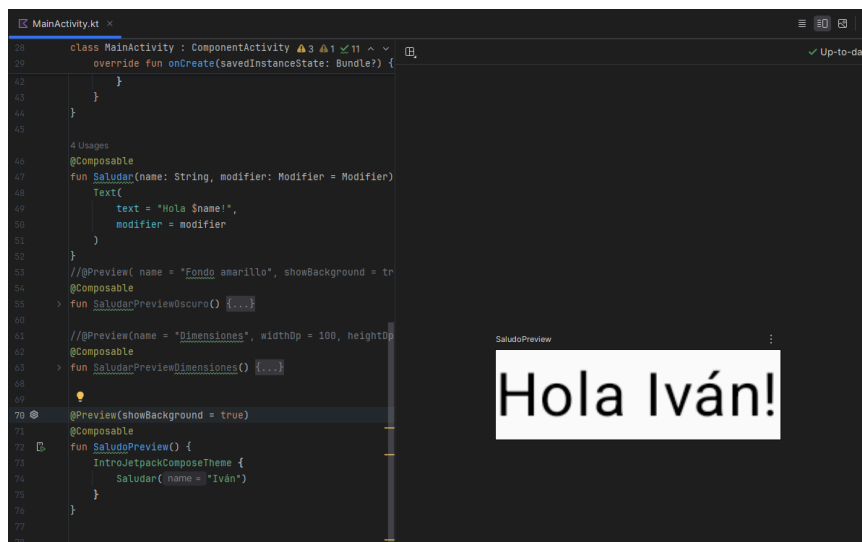
Crearemos una función adicional llamada SaludarPreview para poder ejecutar el código y poder visualizarlo.

Para poder ver esta función sin que corra el programa entero usaremos la anotación @Preview

```
@Preview(showBackground = true)
@Composable
fun SaludoPreview() {
    IntroJetpackComposeTheme {
        Saludar( name = "Iván")
    }
}
```

Hay dos tipos de poder ver el resultado:

Split → Que divide la pantalla en dos, en una mitad muestra el código del programa y en la otra mitad la preview



Design → Ocupa toda la pantalla la preview que quieres realizar:



4- Modificadores

Un Modifier en Compose es un objeto que se pasa a los Composables para modificar colores, tamaños, posiciones, espaciados en la función composable

Creación de una función llamada BotonPersonalizado para hacer una prueba de lo que se puede hacer con los modificadores

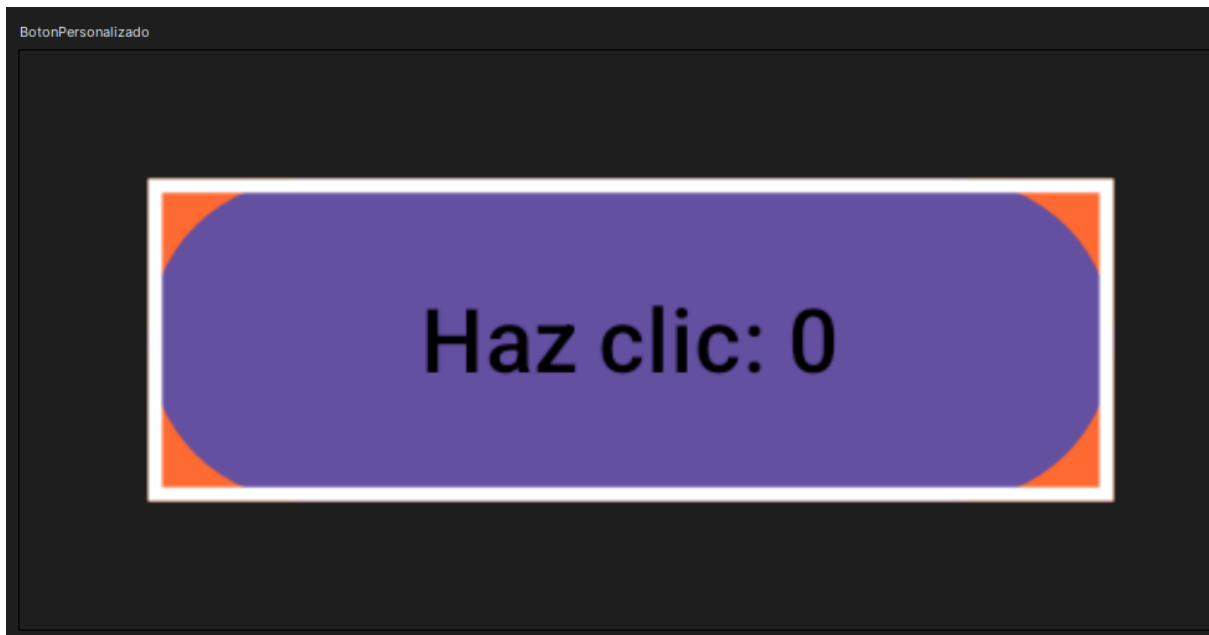
```
@Composable
fun BotonPersonalizado(){

    var contador by remember { mutableStateOf( value = 0) }

    Button( onClick = {
        contador++
    },
        modifier = Modifier.padding( all = 20.dp)
            .size( width = 150.dp, height = 50.dp)
            .background(Color( color = 0xFFFF6D33))
            .border( width = 2.dp, Color.White)
            .fillMaxWidth()
    ) { Text(text = "Haz clic: $contador",
        color = Color.Black)
    }

}
```

El resultado con el `@preview` es el siguiente:

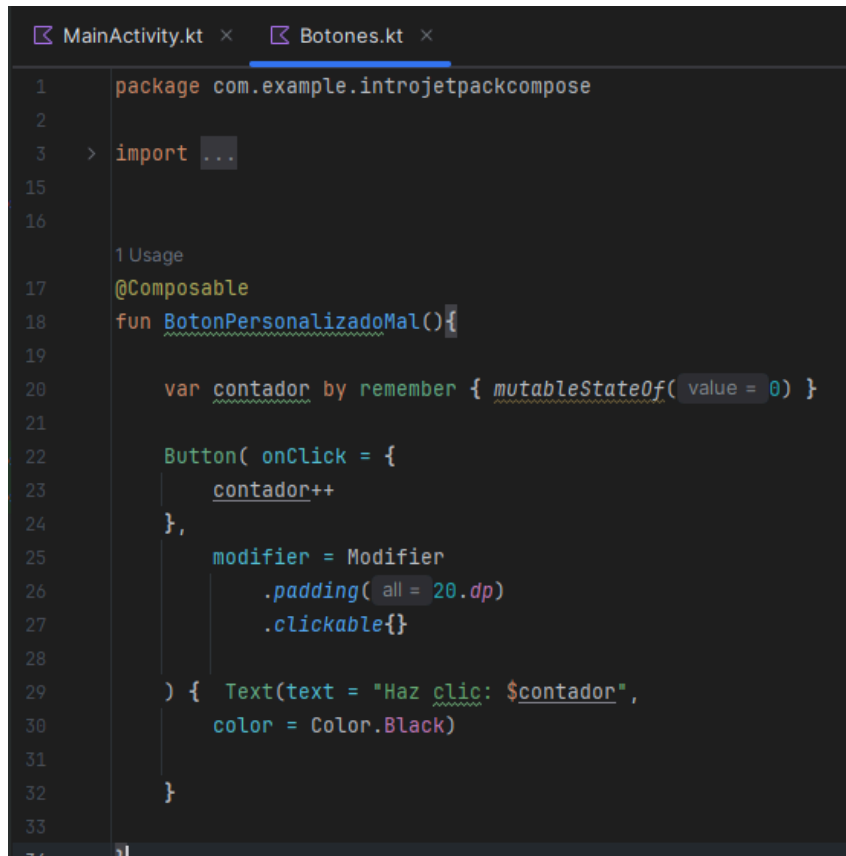


Si `clickable` va antes del `padding`: El área que puede ser clickeada es la del tamaño original del botón sin el `padding`.

Si `padding` va antes de `clickable`: El área clickeable se extiende para incluir también el `padding`, es decir, todo el espacio incluido en el `padding` también será clickeable.

5- Estructura del código

Creación de un archivo llamado [Botones.kt](#) para meter una función llamada `botonPersonalizadoMal()` para hacer la prueba de llamarlo desde el `MainActivity.kt`



```

1 package com.example.introjetpackcompose
2
3 > import ...
15
16
17 1 Usage
18 @Composable
19 fun BotonPersonalizadoMal() {
20     var contador by remember { mutableStateOf( value = 0) }
21
22     Button( onClick = {
23         contador++
24     },
25         modifier = Modifier
26             .padding( all = 20.dp)
27             .clickable{}
28     ) { Text(text = "Haz clic: $contador",
29         color = Color.Black)
30     }
31 }
32
33
34

```

vemos que desde MainActivity podemos llamarlo sin problemas:



```

1 Usage
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            IntroJetpackComposeTheme {
                Scaffold(modifier = Modifier.fillMaxSize()) { innerPadding ->
                    Saludar(
                        name = "Android",
                        modifier = Modifier.padding( paddingValues = innerPadding)
                    )
                    BotonPersonalizadoMal()
                }
            }
        }
    }
}

```

Esta práctica de cómo dividir funciones en diferentes archivos sirve bastante para organizar el código y tener así una mejor estructura al hacer proyectos mayores.

También puede tener una mayor lectura del código ya que al saber dónde y cómo se organiza es mucho más sencillo encontrar las cosas que buscas. Así si se trabajase en equipo en algún momento del proyecto ninguna persona se desconcentra al intentar buscar código o de dónde vienen ciertas funciones.

6- Previsualización avanzada

Para crear diferentes preview solo hace falta modificar el preview de la función que quieres modificar, en mi caso he creado 3 funciones iguales pero con diferente preview

```
@Preview(showBackground = true)
@Composable
fun SaludarPreview() {
    IntroJetpackComposeTheme {
        Saludar( name = "Iván")
    }
}
```

```
@Preview(
    name = "Fondo amarillo",
    showBackground = true,
    backgroundColor = 0xFFFFEB3B)
@Composable
fun SaludarPreview0sucro() {
    IntroJetpackComposeTheme {
        Saludar( name = "Pepe")
    }
}
```

```
@Preview(name = "Dimensiones",widthDp = 100, heightDp = 60, showBackground = true)
@Composable
fun SaludarPreviewDimensiones() {
    IntroJetpackComposeTheme {
        Saludar( name = "Nicolás")
    }
}
```

Los resultados son estos:

