JETPACK COMPOSE II 2025

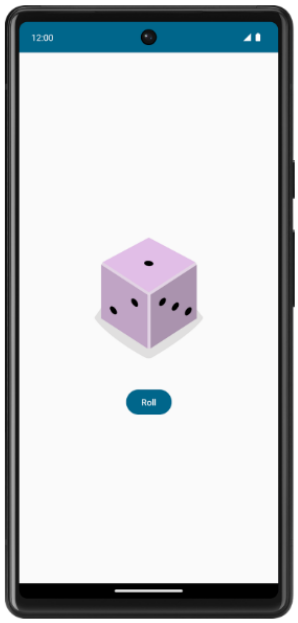
BEGOÑA RODRÍGUEZ FERRERAS
brodfer@gmail.com

Aplicaciones interactivas

En la sesión anterior vimos algunos aspectos básicos de Jetpack Compose y hoy vamos a seguir aprendiendo detalles de la creación de aplicaciones e interfaces.

Cómo agregar un botón a una aplicación

Para ir viendo diferentes aspectos de aplicaciones interactivas vamos a crear nuestro propio dado. Como veis es una imagen de un dado (serán 6 imágenes y un botón). Según pulsemos el botón la imagen cambiará.



Creamos un proyecto que se llame RodarDado (recordar seleccionar la actividad con el logo de Jetpack Compose).

Eliminamos los datos que no nos interesen del HolaMundo y los modificamos con nuestro esqueleto de funciones.

```
@Composable
fun DadoConBotonEImagen(modifier: Modifier = Modifier) {

}

@Preview(showBackground = true)
@Composable
fun AppRodarDado() {
    RodarDadoTheme {
        DadoConBotonEImagen(Modifier)
    }
}
```

En la clase anterior vimos como centrar texto, ahora vemos cómo podemos centrar nuestros elementos.

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            RodarDadoTheme {
                Scaffold(modifier = Modifier.fillMaxSize()) { innerPadding ->
                    DadoConBotonEImagen(modifier = Modifier
                        .padding(innerPadding)
                        .fillMaxSize()
                        .wrapContentSize(Alignment.Center)
                    )
                }
            }
        }
    }
```

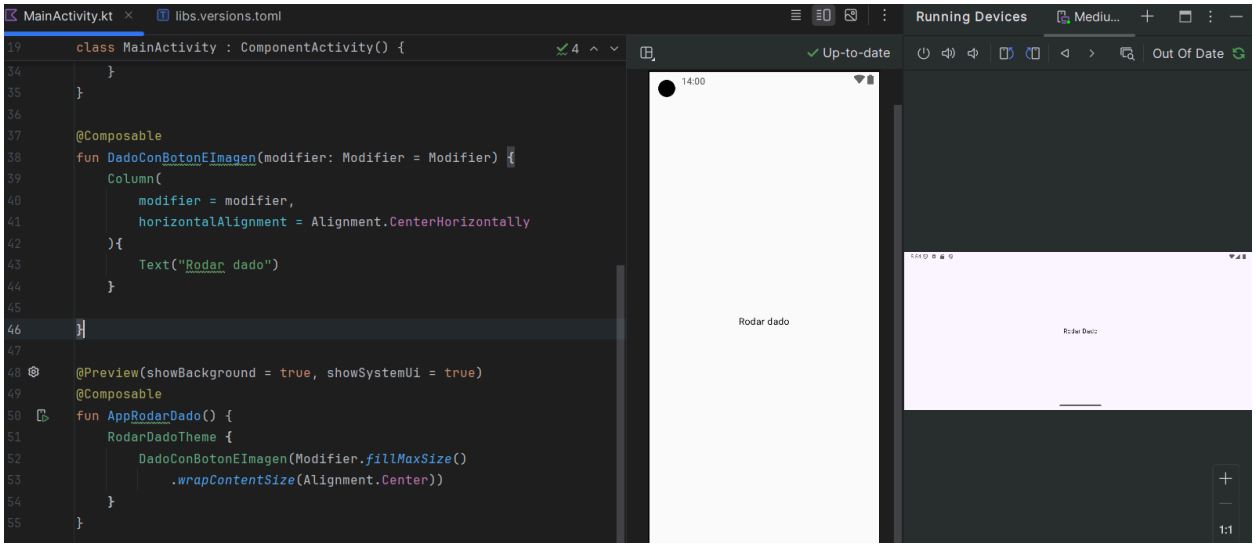
Es importante destacar que el dado y el botón están centrados en la pantalla. El método **wrapContentSize()** especifica que **el espacio disponible debe ser al menos tan grande como los componentes que contiene**. Sin embargo, como se usa el método **fillMaxSize()**, si los componentes dentro del diseño son más pequeños que el espacio disponible, se puede pasar un **objeto Alignment** al método wrapContentSize() que **especifica el modo en que se deben alinear los componentes dentro del espacio disponible**.

Como comenté el día anterior tenemos dos elementos que queremos organizar en una columna y dicha columna será el primer elemento secundario en nuestra función composable, por lo que es a la que le vamos a pasar el modificador:

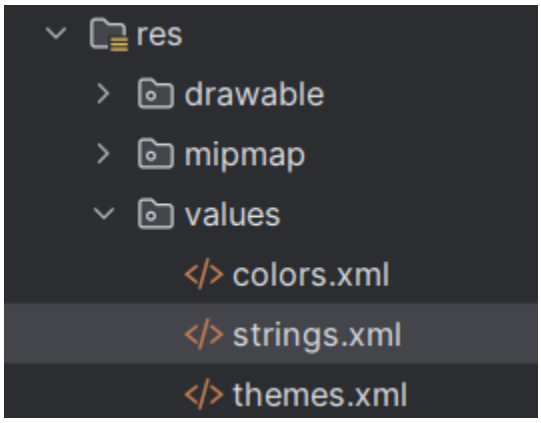
```
@Composable
fun DadoConBotonEImagen(modifier: Modifier = Modifier) {
    Column(
        modifier = modifier,
        horizontalAlignment = Alignment.CenterHorizontally
    ){
        Text("Rodar dado")
    }

}
```

Esto garantiza que los elementos secundarios dentro de la columna estén centrados en la pantalla del dispositivo con respecto al ancho.

Nota: Si giramos la pantalla permanecerán centrados también



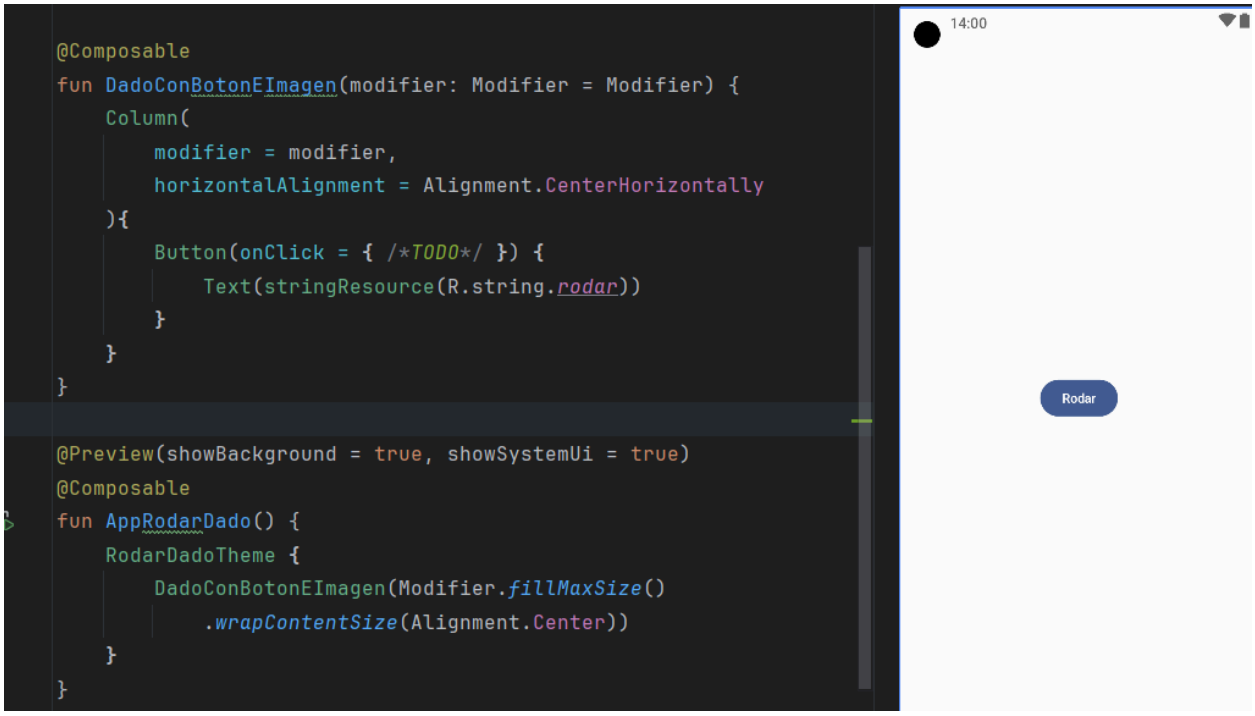
El día anterior estuvimos trabajando con cadenas directamente, pero lo propio es que metamos las cadenas en strings.xml por si el día de mañana queremos traducir nuestra app a otros idiomas. Vamos a res>values>strings.xml y añadimos la cadena rodar:




```
<resources>
    <string name="app_name">RodarDado</string>
    <string name="rodar">Rodar</string>
</resources>
```

```
@Composable
fun DadoConBotonEImagen(modifier: Modifier = Modifier) {
    Column(
        modifier = modifier,
        horizontalAlignment = Alignment.CenterHorizontally
    ){
        Button(onClick = { /*TODO*/ }) {
            Text(stringResource(R.string.rodar))
        }
    }
}
```

Ya aparece nuestro primer botón en la pantalla, aunque todavía no haga nada:



Podéis descargar las imágenes del dado aquí: https://github.com/google-developer-training/basic-android-kotlin-compose-training-dice-roller/raw/main/dice_images.zip

Importamos con el Resource Manager y con control pulsado seleccionamos las 6. No vamos a marcar la densidad en este caso porque son xml.

La imagen del dado debería aparecer encima del botón Roll. **Compose coloca los componentes de la IU de manera inherente de forma secuencial**. En otras palabras, **el elemento de componibilidad que se declara primero se muestra en primer lugar**. Eso podría significar que la primera declaración se muestra arriba, o antes, del elemento de componibilidad que se declara después. Los elementos de componibilidad dentro de un elemento Column aparecerán uno encima o debajo del otro en el dispositivo. En esta app, se usa Column para apilar elementos componibles de manera vertical. Por lo tanto, el elemento que se declare primero dentro de la función Column() se mostrará antes que el elemento declarado posteriormente en la misma función Column(). Así que agregamos la función Image encima del botón:

```
Image(
    painter = painterResource(R.drawable.dice_1), contentDescription = "1"
)
Button(onClick = { /*TODO*/ }) {
    Text(stringResource(R.string.rodar))
}
```

Nota: **Si no ponéis el contentDescription os va a dar error.** Lo propio es poner 1 en este caso.

Lógica del dado

Observas el elemento Button de componibilidad. Notarás que se le pasa un parámetro onClick configurado como un par de llaves con el comentario /*TODO*/ dentro de las llaves. Las llaves, en este caso, representan lo que se conoce como una lambda; **el área dentro de las llaves es el cuerpo de lambda**.

Una lambda es un literal de función, que es como cualquier otra función, pero en lugar de declararse por separado con la palabra clave fun, se escribe intercalada y se pasa como una expresión.

¿Qué es lo que queremos hacer? Calcular un número aleatorio entre 1 y 6. Recordad que, en Kotlin, **los rangos se designan con dos puntos entre el primer número del rango y el último número del rango.** Definimos entonces un rango de la siguiente forma y llamamos al método random sobre dicho rango (1..6).random()

```
@Composable
fun DadoConBotonEImagen(modifier: Modifier = Modifier) {
    var result = 1
    Column(
        modifier = modifier,
        horizontalAlignment = Alignment.CenterHorizontally
    ){
        Image(
            painter = painterResource(R.drawable.dice_1), contentDe
        )
        Button(onClick = { result = (1 ≤ .. ≤ 6).random() }) {
            Text(stringResource(R.string.rodar))
        }
    }
}
```

Cómo agregar un condicional a la app de lanzamiento de dados

Con esto el botón ya haría algo, pero no se ve reflejado en el programa. Lo que queremos es que la variable result y la imagen consiguientemente se modifiquen.

Los elementos de componibilidad no tienen **estado** de forma predeterminada, lo que significa que **no tienen un valor y el sistema los puede volver a componer en cualquier momento**, lo que hace que se restablezca el valor. Sin embargo, Compose proporciona una forma conveniente de evitarlo. **Las funciones de componibilidad pueden almacenar un objeto en la memoria con el elemento componible remember.** Un valor calculado por remember se almacena en la composición durante la composición inicial, y el valor almacenado se muestra durante la recomposición.

El elemento **remember** componible requiere que se pase una función. La función **mutableStateOf()** muestra un **elemento observable**. Más adelante, aprenderemos más sobre los elementos observables, pero, por ahora, esto **significa que cuando cambia el valor de la variable result, se activa una recomposición, se refleja el valor del resultado y se actualiza la IU**.

Ahora, cuando se presiona el botón, se actualiza la variable result con un valor del número al azar. Debajo de la creación de instancias de la variable result, crea una variable imageResource inmutable establecida en una expresión **when** que acepte una variable result y, luego, establece cada resultado posible en su elemento de diseño.

```
@Composable
fun DadoConBotonEImagen(modifier: Modifier = Modifier) {
    var result by remember {mutableStateOf(1)}
    val imageResource = when (result) {
        1 -> R.drawable.dice_1
        2 -> R.drawable.dice_2
        3 -> R.drawable.dice_3
        4 -> R.drawable.dice_4
        5 -> R.drawable.dice_5
        else -> R.drawable.dice_6
    }
    Column(
        modifier = modifier,
        horizontalAlignment = Alignment.CenterHorizontally
    ){
        Image(
            painter = painterResource(imageResource), contentDescription = result.toString()
        )
        Button(onClick = { result = (1 ≤ .. ≤ 6).random() }) {
            Text(stringResource(R.string.rodar))
        }
    }
}
```

Introducción al estado en Compose

El **estado** de una app es cualquier valor que puede cambiar con el tiempo. Esta definición es muy amplia y abarca desde una base de datos hasta una variable en tu app.

Todas las apps para Android muestran un estado al usuario. Estos son algunos ejemplos de estado de las apps para Android:

- Un mensaje que se muestra cuando no se puede establecer una conexión de red.
- Formularios, como formularios de registro. Puedes completar y enviar tu estado.
- Controles que se pueden presionar, como botones. El estado puede ser *no presionado*, *se está presionando* (animación de la pantalla) o *presionado* (una acción `onClick`).