Documentación: Fundamentos de Kotlin

Fernando Rodriguez

1. Ruta de aprendizaje 1

Introducción

Este documento proporciona una visión detallada de los fundamentos avanzados del lenguaje de programación Kotlin, esenciales para el desarrollo de aplicaciones Android utilizando Jetpack Compose. A lo largo de este módulo, se exploran conceptos como condicionales, nulabilidad, clases, objetos, funciones de orden superior y expresiones lambda.

Condicionales en Kotlin

Las estructuras condicionales permiten ejecutar bloques de código basados en condiciones específicas. En Kotlin, se utilizan principalmente las siguientes:

- `if`: Evalúa una condición y ejecuta un bloque de código si la condición es verdadera.
- `else`: Se utiliza junto con `if` para ejecutar un bloque de código cuando la condición es falsa.
- `when`: Funciona como una alternativa más poderosa a `if-else`, permitiendo evaluar múltiples condiciones.

```
val numero = 10

if (numero > 5) {
    println("El n mero es mayor que 5")
} else {
    println("El n mero es menor o igual a 5")
}
```

Listing 1: Uso de condicionales en Kotlin

Nulabilidad en Kotlin

Kotlin introduce el concepto de nulabilidad para evitar errores comunes relacionados con referencias nulas. Para declarar una variable que puede ser nula, se utiliza el operador `?`.

```
val nombre: String? = null
```

Listing 2: Declaración de variables anulables

Para acceder a propiedades o métodos de una variable anulable, se utiliza el operador seguro `?.`, que solo ejecuta la acción si la variable no es nula.

```
val longitud = nombre?.length
```

Listing 3: Uso del operador seguro

Clases y Objetos en Kotlin

Kotlin es un lenguaje orientado a objetos que permite la creación de clases y objetos de manera sencilla.

```
class Persona(val nombre: String, var edad: Int)

val persona = Persona("Juan", 30)
```

Listing 4: Definición de una clase y creación de un objeto

Las clases pueden contener propiedades y métodos que definen el comportamiento y estado de los objetos.

Funciones de Orden Superior y Expresiones Lambda

Kotlin permite el uso de funciones de orden superior, es decir, funciones que pueden recibir otras funciones como parámetros o devolverlas. Las expresiones lambda son una forma concisa de definir funciones anónimas.

```
fun operar(a: Int, b: Int, operacion: (Int, Int) -> Int): Int {
    return operacion(a, b)
}

val suma: (Int, Int) -> Int = { x, y -> x + y }

val resultado = operar(5, 3, suma)
```

Listing 5: Uso de funciones de orden superior y expresiones lambda

Práctica: Fundamentos de Kotlin

Para consolidar los conocimientos adquiridos, se recomienda realizar ejercicios prácticos que impliquen:

- Implementar estructuras condicionales para resolver problemas específicos.
- Crear y manipular variables anulables.
- Definir clases y objetos con propiedades y métodos.

- Utilizar funciones de orden superior y expresiones lambda en situaciones prácticas.

Estas prácticas ayudarán a familiarizarse con la sintaxis y las mejores prácticas de Kotlin.

Conclusión

Este módulo proporciona una base sólida en los fundamentos avanzados de Kotlin, esenciales para el desarrollo de aplicaciones Android modernas. Al comprender cómo utilizar condicionales, manejar la nulabilidad, trabajar con clases y objetos, y aplicar funciones de orden superior y expresiones lambda, los desarrolladores pueden comenzar a construir aplicaciones más complejas y funcionales utilizando Jetpack Compose.

2. Ruta de aprendizaje 2

Introducción

Este documento proporciona una guía detallada para agregar un botón interactivo en una aplicación Android utilizando Jetpack Compose. Aprenderás cómo responder a los clics del usuario y cómo utilizar el depurador de Android Studio para inspeccionar y depurar el estado de tu aplicación en tiempo de ejecución.

Aplicación de lanzamiento de dados

En este módulo, se construye una aplicación llamada *Dice Roller*, que permite a los usuarios lanzar un dado y mostrar el resultado. Esta aplicación sirve como base para aprender a manejar interacciones del usuario y actualizar la interfaz de usuario en respuesta a esas interacciones.

Creación de una aplicación interactiva de lanzamiento de dados

Para construir la aplicación *Dice Roller*, se siguen los siguientes pasos:

1. **Definir el estado del dado:** Se utiliza una variable para almacenar el valor actual del dado.

2. **Crear una función para generar un número aleatorio:** Esta función simula el lanzamiento de un dado generando un número aleatorio entre 1 y 6.

3. **Diseñar la interfaz de usuario:** Se utiliza un `Column` composable para organizar los elementos de la interfaz, incluyendo un `Button` que, al ser presionado, invoca la función de lanzamiento del dado y actualiza el valor mostrado.

```
@Composable
fun DiceRollerApp() {
    var diceNumber by remember { mutableStateOf(1) }

    Column(
        modifier = Modifier
            .fillMaxSize()
            .wrapContentSize(Alignment.Center)
    ) {
        Text(text = "Lanzar dado", style = MaterialTheme.
           typography.h4)
        Spacer(modifier = Modifier.height(16.dp))
        Button(onClick = { diceNumber = (1..6).random() }) {
            Text("Lanzar")
        }
        Spacer(modifier = Modifier.height(16.dp))
        Text(text = "Resultado: $diceNumber", style =
           MaterialTheme.typography.h5)
    }
}
```

Listing 6: Ejemplo de implementación de la aplicación Dice Roller

Explicación del código:

- `var diceNumber by remember  mutableStateOf(1)` : Declara una variable de estado que almacena el valor actual del dado.
- `Button(onClick =  diceNumber = (1..6).random() )`: Define un botón que, al ser presionado, genera un número aleatorio entre 1 y 6 y actualiza el estado del dado.

- `Text(text = Resultado:` $diceNumber$") : $Muestra el valor actual del dado en la interfaz de usuario.$

Uso del depurador en Android Studio

El depurador de Android Studio es una herramienta poderosa que permite inspeccionar y depurar el estado de tu aplicación en tiempo de ejecución. Para utilizar el depurador:

1. Establece puntos de interrupción en tu código haciendo clic en el margen izquierdo junto al número de línea.
2. Ejecuta tu aplicación en modo de depuración.
3. Cuando la ejecución alcance un punto de interrupción, la aplicación se detendrá y podrás inspeccionar las variables, ver la pila de llamadas y controlar la ejecución paso a paso.

El uso del depurador te ayuda a comprender el flujo de tu aplicación y a identificar y corregir errores de manera eficiente.

Práctica: Comportamiento al hacer clic

Para consolidar los conocimientos adquiridos, se recomienda realizar ejercicios prácticos que impliquen:

- Implementar botones que realicen diferentes acciones en respuesta a los clics del usuario.
- Utilizar el depurador para inspeccionar el estado de la aplicación y comprender su comportamiento en tiempo de ejecución.
- Experimentar con la actualización dinámica de la interfaz de usuario en respuesta a las interacciones del usuario.

Estas prácticas ayudarán a familiarizarse con la creación de interfaces de usuario interactivas y el uso de herramientas de depuración en Android Studio.

Conclusión

Este módulo proporciona una base sólida para crear aplicaciones interactivas en Android utilizando Jetpack Compose. Al comprender cómo manejar las interacciones del usuario y utilizar el depurador de Android Studio, los desarrolladores pueden construir aplicaciones más dinámicas y fáciles de mantener.

3. Ruta de aprendizaje 3

Introducción

Este documento proporciona una guía detallada para crear una aplicación de calculadora de propinas utilizando Jetpack Compose. La aplicación permite al usuario ingresar el total de la cuenta y seleccionar el porcentaje de propina deseado, mostrando el monto de la propina calculado en tiempo real.

Aplicación de calculadora de propinas

La aplicación consta de los siguientes elementos principales:

- **Entrada de texto para el total de la cuenta:** Permite al usuario ingresar el monto total de la cuenta.
- **Selector de porcentaje de propina:** Permite al usuario seleccionar el porcentaje de propina deseado.
- **Botón para calcular la propina:** Al presionar este botón, se calcula el monto de la propina basado en el total de la cuenta y el porcentaje seleccionado.
- **Texto que muestra el monto de la propina:** Muestra el monto de la propina calculado.

Implementación en Jetpack Compose

A continuación, se presenta una implementación básica de la aplicación utilizando Jetpack Compose:

```
@Composable
fun TipCalculator() {
    var totalAmount by remember { mutableStateOf("") }
    var tipPercentage by remember { mutableStateOf(15) }
    var tipAmount by remember { mutableStateOf(0.0) }

    Column(
        modifier = Modifier
            .padding(16.dp)
            .fillMaxWidth()
    ) {
        TextField(
            value = totalAmount,
            onValueChange = { totalAmount = it },
            label = { Text("Total de la cuenta") },
            keyboardOptions = KeyboardOptions.Default.copy(
               keyboardType = KeyboardType.Number)
        )
        Spacer(modifier = Modifier.height(8.dp))
        Row(
            horizontalArrangement = Arrangement.spacedBy(8.dp)
        ) {
            Text("Porcentaje de propina:")
            Text("${tipPercentage}%")
        }
        Slider(
            value = tipPercentage.toFloat(),
            onValueChange = { tipPercentage = it.toInt() },
            valueRange = 0f..30f,
            steps = 29
        )
        Spacer(modifier = Modifier.height(8.dp))
        Button(
            onClick = {
                val total = totalAmount.toDoubleOrNull() ?: 0.0
                tipAmount = total * tipPercentage / 100
            }
        ) {
            Text("Calcular propina")
        }
        Spacer(modifier = Modifier.height(8.dp))
        Text("Monto de la propina: $${"%.2f".format(tipAmount)}")
    }
}
```

Listing 7: Implementación de la calculadora de propinas

Explicación del código:

- `totalAmount`, `tipPercentage`, `tipAmount`: Variables de estado que almacenan el total de la cuenta, el porcentaje de propina y el monto de la propina calculado, respectivamente.
- `TextField`: Composable que permite al usuario ingresar el total de la cuenta.
- `Slider`: Composable que permite al usuario seleccionar el porcentaje de propina deseado.
- `Button`: Composable que, al ser presionado, calcula el monto de la propina basado en el total de la cuenta y el porcentaje seleccionado.
- `Text`: Composable que muestra el monto de la propina calculado.

Prácticas recomendadas

Al desarrollar aplicaciones con Jetpack Compose que interactúan con la interfaz de usuario y el estado, se recomienda:

- Utilizar `remember` y `mutableStateOf` para almacenar y recordar el estado entre recomposiciones.
- Manejar las entradas del usuario de manera eficiente, validando y formateando los datos según sea necesario.
- Proporcionar retroalimentación inmediata al usuario, como mostrar el monto de la propina calculado en tiempo real.
- Mantener una estructura de código limpia y modular, utilizando funciones composables para representar componentes reutilizables de la interfaz de usuario.

Conclusión

Este módulo proporciona una base sólida para crear aplicaciones interactivas en Android utilizando Jetpack Compose. Al comprender cómo manejar el estado y las interacciones del usuario, los desarrolladores pueden construir aplicaciones más dinámicas y receptivas.