Unidad 5

Rodriguez Segura Fernando

Actividad 1

1 Introducción

El codelab *Introducción a las corrutinas en el Playground de Kotlin* es una guía práctica proporcionada por Google Developer Training. Su objetivo es enseñar a los desarrolladores a utilizar las corrutinas de Kotlin para realizar tareas simultáneas sin bloquear la interfaz de usuario de la aplicación, mejorando así la experiencia del usuario.

2 Objetivos del Codelab

Al completar este codelab, los participantes aprenderán a:

- Comprender el concepto de simultaneidad y su importancia en el desarrollo de aplicaciones Android.
- Utilizar las corrutinas de Kotlin para simplificar la programación asíncrona.
- Implementar la simultaneidad estructurada y entender su relevancia.

3 Requisitos Previos

Para aprovechar al máximo este codelab, se recomienda tener:

- Conocimientos básicos de Kotlin, incluyendo funciones y lambdas.
- Experiencia en la creación de programas simples en Kotlin con una función `main()`.
- Acceso a Internet para utilizar el Playground de Kotlin.

4 Contenido del Codelab

4.1 Código Síncrono

Se inicia con un programa sencillo en Kotlin que muestra un pronóstico del tiempo de forma secuencial:

Listing 1: Programa Síncrono

```
fun main() {
    println("Weather forecast")
    println("Sunny")
}
```

Este código se ejecuta de manera secuencial, mostrando primero "Weather forecast" y luego "Sunny".

4.2 Introducción de Retrasos

Para simular una tarea asincrónica, se introduce un retraso utilizando la función `delay()` de las corrutinas:

Listing 2: Programa con Retraso

```
import kotlinx.coroutines.*

fun main() {
    runBlocking {
        println("Weather forecast")
        delay(1000)
        println("Sunny")
    }
}
```

Aquí, la ejecución se suspende durante 1 segundo antes de imprimir "Sunny".

4.3 Simultaneidad Estructurada

Se explica el concepto de simultaneidad estructurada, que permite gestionar de manera eficiente las tareas concurrentes, asegurando que todas las corrutinas se completen antes de finalizar el programa.

5 Conclusión

Este codelab proporciona una introducción práctica a las corrutinas en Kotlin, permitiendo a los desarrolladores comprender y aplicar la simultaneidad estructurada en sus aplicaciones Android. Al dominar estas técnicas, se mejora la capacidad de las aplicaciones para realizar múltiples tareas simultáneamente sin comprometer la experiencia del usuario.

6 Introducción

El proyecto **Race Tracker** es una aplicación educativa desarrollada en Kotlin utilizando Jetpack Compose. Su propósito principal es enseñar el uso de corrutinas para manejar operaciones simultáneas de forma sencilla, sin bloquear la interfaz de usuario. En esta aplicación, se simula una carrera entre dos jugadores cuyo progreso se actualiza periódicamente.

7 Funcionalidad Principal

La aplicación permite:

- Iniciar la carrera para que ambos jugadores avancen simultáneamente.
- Pausar la carrera en cualquier momento.
- Reiniciar la carrera a su estado inicial.

El progreso de cada jugador se muestra en tiempo real en la interfaz de usuario mediante barras de progreso.

8 Descripción del Código

8.1 Uso de Corrutinas

El proyecto hace uso de corrutinas de Kotlin para simular la carrera de manera concurrente. Cada jugador avanza en una corrutina independiente que actualiza su progreso en intervalos regulares sin bloquear el hilo principal.

Listing 3: Ejemplo simplificado de corrutina para un jugador

```
import kotlinx.coroutines.*

fun startRace() {
    GlobalScope.launch {
        while (playerProgress < maxProgress) {
            delay(100L)  // Simula tiempo de avance
            playerProgress += 1
            // Actualiza UI con nuevo progreso
        }
    }
}
```

8.2 Interfaz de Usuario con Jetpack Compose

La interfaz se construye declarativamente con Compose, usando componentes como `Button` y `LinearProgressIndicator` para mostrar el estado de la carrera y permitir la interacción.

Listing 4: Ejemplo simplificado de UI Compose

```
@Composable
fun RaceScreen() {
    Column {
        LinearProgressIndicator(progress = playerOneProgress)
        LinearProgressIndicator(progress = playerTwoProgress)

        Row {
            Button(onClick = { startRace() }) { Text("Start") }
```

```
            Button(onClick = { pauseRace() }) { Text("Pause") }
            Button(onClick = { resetRace() }) { Text("Reset") }
        }
    }
}
```

9 Estructura del Proyecto

- `MainActivity.kt`: Punto de entrada de la aplicación donde se configura la UI y se manejan los estados.
- `RaceViewModel.kt`: Contiene la lógica de negocio y controla las corrutinas para la simulación de la carrera.
- `ui/`: Contiene los archivos relacionados con la interfaz de usuario implementada con Compose.
- Archivos de configuración de Gradle para la construcción y dependencias del proyecto.

10 Introducción

El proyecto **Mars Photos** es una aplicación educativa desarrollada en Kotlin utilizando Jetpack Compose. Su propósito principal es enseñar a los desarrolladores cómo consumir datos de una API REST, manejar la deserialización de JSON y cargar imágenes desde la web en una aplicación Android moderna.

11 Funcionalidad Principal

La aplicación permite:

- Consultar imágenes reales de la superficie de Marte capturadas por los rovers de la NASA.
- Mostrar las imágenes en una interfaz de usuario construida con Jetpack Compose.
- Manejar la carga de imágenes de manera eficiente utilizando la biblioteca Coil.
- Deserializar los datos JSON obtenidos de la API utilizando Kotlinx Serialization.

12 Tecnologías Utilizadas

- **Jetpack Compose**: *Toolkit* moderno para construir interfaces de usuario en Android.
- **Retrofit**: Biblioteca para realizar solicitudes HTTP y consumir servicios RESTful.
- **Kotlinx Serialization**: Biblioteca para la serialización y deserialización de datos en formato JSON.
- **Coil**: Biblioteca para la carga de imágenes en Android.
- **Coroutines**: Para manejar operaciones asincrónicas de manera eficiente.

13 Estructura del Proyecto

- `MainActivity.kt`: Actividad principal que configura la interfaz de usuario y maneja la navegación.
- `MarsApiService.kt`: Interfaz que define las solicitudes a la API de imágenes de Marte.
- `MarsPhoto.kt`: Modelo de datos que representa una imagen de Marte.
- `MarsPhotoRepository.kt`: Repositorio que maneja la obtención de datos desde la API.
- `MarsPhotoViewModel.kt`: ViewModel que proporciona los datos a la interfaz de usuario.
- `ui/`: Carpeta que contiene los componentes de la interfaz de usuario construidos con Jetpack Compose.

14 Ejemplo de Código

14.1 Definición del Servicio Retrofit

Listing 5: Definición del Servicio Retrofit

```
interface MarsApiService {
    @GET("photos")
    suspend fun getPhotos(): List<MarsPhoto>
}
```

14.2 Modelo de Datos MarsPhoto

Listing 6: Modelo de Datos MarsPhoto

```
@Serializable
data class MarsPhoto(
    val id: String,
    val imgSrc: String
)
```

14.3 Carga de Imágenes con Coil

Listing 7: Carga de Imágenes con Coil

```
@Composable
fun MarsImage(imageUrl: String) {
    val painter = rememberImagePainter(imageUrl)
    Image(painter = painter, contentDescription = null)
}
```

Actividad 2

15 Introducción

El proyecto **Mars Photos** es una aplicación educativa desarrollada en Kotlin utilizando Jetpack Compose. Su propósito principal es enseñar a los desarrolladores cómo consumir datos de una API REST, manejar la deserialización de JSON y cargar imágenes desde la web en una aplicación Android moderna.

16 Funcionalidad Principal

La aplicación permite:

- Consultar imágenes reales de la superficie de Marte capturadas por los rovers de la NASA.
- Mostrar las imágenes en una interfaz de usuario construida con Jetpack Compose.
- Manejar la carga de imágenes de manera eficiente utilizando la biblioteca Coil.
- Deserializar los datos JSON obtenidos de la API utilizando Kotlinx Serialization.

17 Tecnologías Utilizadas

- **Jetpack Compose**: *Toolkit* moderno para construir interfaces de usuario en Android.
- **Retrofit**: Biblioteca para realizar solicitudes HTTP y consumir servicios RESTful.
- **Kotlinx Serialization**: Biblioteca para la serialización y deserialización de datos en formato JSON.
- **Coil**: Biblioteca para la carga de imágenes en Android.
- **Coroutines**: Para manejar operaciones asincrónicas de manera eficiente.

18 Estructura del Proyecto

- `MainActivity.kt`: Actividad principal que configura la interfaz de usuario y maneja la navegación.
- `MarsApiService.kt`: Interfaz que define las solicitudes a la API de imágenes de Marte.
- `MarsPhoto.kt`: Modelo de datos que representa una imagen de Marte.
- `MarsPhotoRepository.kt`: Repositorio que maneja la obtención de datos desde la API.
- `MarsPhotoViewModel.kt`: ViewModel que proporciona los datos a la interfaz de usuario.
- `ui/`: Carpeta que contiene los componentes de la interfaz de usuario construidos con Jetpack Compose.

19 Ejemplo de Código

19.1 Definición del Servicio Retrofit

Listing 8: Definición del Servicio Retrofit

```
interface MarsApiService {
    @GET("photos")
    suspend fun getPhotos(): List<MarsPhoto>
}
```

19.2 Modelo de Datos MarsPhoto

Listing 9: Modelo de Datos MarsPhoto

```
@Serializable
data class MarsPhoto(
    val id: String,
    val imgSrc: String
)
```

19.3 Carga de Imágenes con Coil

Listing 10: Carga de Imágenes con Coil

```
@Composable
fun MarsImage(imageUrl: String) {
    val painter = rememberImagePainter(imageUrl)
    Image(painter = painter, contentDescription = null)
}
```

20 Introducción

El proyecto **Amphibians App** es una aplicación educativa desarrollada en Kotlin utilizando Jetpack Compose. Su propósito principal es enseñar a los desarrolladores cómo consumir datos de una API REST, manejar la deserialización de JSON y cargar imágenes desde la web en una aplicación Android moderna.

21 Funcionalidad Principal

La aplicación permite:

- Consultar información sobre diferentes especies de anfibios desde una API proporcionada por el curso.
- Mostrar las imágenes y descripciones de los anfibios en una interfaz de usuario construida con Jetpack Compose.

- Manejar la carga de imágenes de manera eficiente utilizando la biblioteca Coil.
- Deserializar los datos JSON obtenidos de la API utilizando Kotlinx Serialization.

22 Tecnologías Utilizadas

- **Jetpack Compose**: Toolkit moderno para construir interfaces de usuario en Android.
- **Retrofit**: Biblioteca para realizar solicitudes HTTP y consumir servicios RESTful.
- **Kotlinx Serialization**: Biblioteca para la serialización y deserialización de datos en formato JSON.
- **Coil**: Biblioteca para la carga de imágenes en Android.
- **Coroutines**: Para manejar operaciones asincrónicas de manera eficiente.

23 Estructura del Proyecto

- `MainActivity.kt`: Actividad principal que configura la interfaz de usuario y maneja la navegación.
- `AmphibiansApiService.kt`: Interfaz que define las solicitudes a la API de anfibios.
- `Amphibian.kt`: Modelo de datos que representa una especie de anfibio.
- `AmphibiansRepository.kt`: Repositorio que maneja la obtención de datos desde la API.
- `AmphibiansViewModel.kt`: ViewModel que proporciona los datos a la interfaz de usuario.
- `ui/`: Carpeta que contiene los componentes de la interfaz de usuario construidos con Jetpack Compose.

24 Ejemplo de Código

24.1 Definición del Servicio Retrofit

Listing 11: Definición del Servicio Retrofit

```
interface AmphibiansApiService {
    @GET("amphibians")
    suspend fun getAmphibians(): List<Amphibian>
}
```

24.2 Modelo de Datos Amphibian

Listing 12: Modelo de Datos Amphibian

```
@Serializable
data class Amphibian(
    val name: String,
    val type: String,
    val description: String,
    @SerialName("img_src") val imgSrc: String
)
```

24.3 Carga de Imágenes con Coil

Listing 13: Carga de Imágenes con Coil

```
@Composable
fun AmphibianImage(imageUrl: String) {
    val painter = rememberImagePainter(imageUrl)
    Image(painter = painter, contentDescription = null)
}
```

25 Conclusión

El proyecto Amphibians App es una excelente referencia para aprender a consumir datos de una API REST en una aplicación Android moderna utilizando Kotlin y Jetpack Compose. Además, demuestra cómo manejar la deserialización de JSON y la carga eficiente de imágenes desde la web.