Más aspectos básicos de Kotlin

Rodriguez Segura Fernando

1

Introducción

Esta documentación resume los contenidos de la primera ruta de aprendizaje de la Unidad 3 del curso *Aspectos básicos de Android con Compose* ofrecido por Google. Esta sección se titula **Más aspectos básicos de Kotlin** y tiene como objetivo profundizar en conceptos intermedios del lenguaje de programación Kotlin que son esenciales para el desarrollo de aplicaciones Android modernas usando Jetpack Compose.

Contenido de la Ruta de Aprendizaje

1. Genéricos, Objetos y Extensiones

En esta sección se exploran los siguientes conceptos fundamentales:

- **Clases Genéricas:** Permiten crear clases y funciones que trabajan con cualquier tipo de dato.
- **Clases `enum`:** Definen conjuntos de constantes con comportamiento específico.
- **Clases de Datos (`data class`):** Se usan para almacenar datos y generan automáticamente funciones útiles como `toString()` o `equals()`.
- **Objetos (`object`):** Se usan para definir clases singleton.
- **Funciones de Alcance:** Funciones como `let`, `apply`, `run`, `also` y `with` ayudan a organizar y simplificar el código.

2. Uso de Colecciones en Kotlin

Kotlin provee un conjunto poderoso de estructuras de datos:

- **Listas (`List`):** Conjuntos ordenados de elementos.
- **Conjuntos (`Set`):** Colecciones sin elementos repetidos.
- **Mapas (`Map`):** Estructuras clave-valor.
- **Arreglos (`Array`):** Estructura de tamaño fijo para datos homogéneos.

3. Funciones de Orden Superior con Colecciones

Kotlin permite operar de forma funcional sobre colecciones:

- `map`: Transforma elementos.
- `filter`: Filtra elementos según una condición.
- `forEach`: Itera sobre todos los elementos.
- `any`, `all`, `none`: Verifican condiciones sobre las colecciones.

4. Práctica: Clases y Colecciones

Se propone un ejercicio práctico en el *Kotlin Playground* para poner en práctica el uso de clases, objetos, y operaciones sobre colecciones. Esta práctica permite reforzar el uso de estructuras y funciones clave en el lenguaje Kotlin.

5. Cuestionario

Finalmente, se presenta un cuestionario para evaluar los conocimientos adquiridos. Al completarlo exitosamente, el estudiante puede obtener la insignia correspondiente que valida su progreso en la ruta de aprendizaje.

Conclusión

Esta ruta de aprendizaje es esencial para dominar aspectos más avanzados de Kotlin, facilitando la creación de aplicaciones Android más estructuradas, eficientes y modernas. Estos fundamentos permiten aprovechar el potencial completo de Jetpack Compose y el ecosistema Android actual.

2

Introducción

Esta documentación resume los contenidos de la segunda ruta de aprendizaje de la Unidad 3 del curso *Aspectos básicos de Android con Compose* ofrecido por Google. Esta sección se titula **Crear una lista desplazable** y tiene como objetivo enseñar cómo implementar listas desplazables en aplicaciones Android utilizando Jetpack Compose.

Contenido de la Ruta de Aprendizaje

1. Crear una clase de datos para los elementos de la lista

Se define una clase de datos llamada `Affirmation` que contiene dos propiedades: `stringResourceId` y `imageResourceId`, ambas de tipo `Int`. Estas propiedades representan los recursos de texto e imagen que se mostrarán en cada elemento de la lista.

2. Crear una tarjeta de Material Design

Se implementa una función componible `AffirmationCard` que utiliza el componente `Card` de Material Design para mostrar una imagen y un texto. Dentro de la tarjeta, se utiliza una `Column` para organizar verticalmente una `Image` y un `Text`.

3. Crear una lista desplazable con LazyColumn

Se crea una función componible `AffirmationList` que utiliza `LazyColumn` para mostrar una lista de elementos `Affirmation`. `LazyColumn` permite cargar elementos de manera eficiente y proporciona desplazamiento automático.

4. Mostrar la lista en la aplicación

En la función `AffirmationsApp`, se llama a `AffirmationList` y se le pasa una lista de afirmaciones obtenida de una clase `Datasource`. Se utiliza `Surface` y `Modifier` para aplicar estilos y padding adecuados a la interfaz de usuario.

Conclusión

Esta ruta de aprendizaje proporciona una guía paso a paso para crear listas desplazables en aplicaciones Android utilizando Jetpack Compose. Al aprender a definir clases de datos, crear tarjetas de Material Design y utilizar `LazyColumn`, los desarrolladores pueden construir interfaces de usuario modernas y eficientes.

3

Introducción

Esta documentación resume los contenidos de la tercera ruta de aprendizaje de la Unidad 3 del curso *Aspectos básicos de Android con Compose* ofrecido por Google. Esta sección se titula **Compila apps fabulosas** y tiene como objetivo enseñar cómo mejorar la apariencia y la experiencia del usuario en aplicaciones Android utilizando Jetpack Compose.

Contenido de la Ruta de Aprendizaje

1. Introducción a Material Design con Compose

Se presenta un video opcional que introduce los principios de Material Design y cómo se integran con Jetpack Compose para crear interfaces de usuario coherentes y atractivas.

2. Temas de Material con Jetpack Compose

En este codelab, se aprende a aplicar temas de Material Design en una aplicación de Compose, personalizando aspectos como colores, formas y tipografías para mantener una identidad visual consistente.

3. Animación simple con Jetpack Compose

Este codelab enseña a agregar animaciones simples a una aplicación utilizando Jetpack Compose, mejorando la interactividad y la experiencia del usuario.

4. Práctica: Compila apps de superhéroes

En este ejercicio práctico, se aplica lo aprendido sobre Material Design y animaciones para construir una aplicación que muestra una lista de superhéroes, consolidando los conocimientos adquiridos.

Conclusión

Esta ruta de aprendizaje proporciona las herramientas necesarias para crear aplicaciones Android más atractivas y modernas. Al dominar los temas de Material Design y las animaciones en Jetpack Compose, los desarrolladores pueden mejorar significativamente la experiencia del usuario en sus aplicaciones.

Codigo

Introducción

Este documento describe la estructura y funcionamiento del archivo principal de una aplicación Android desarrollada en Kotlin utilizando Jetpack Compose y Material 3. El archivo central es `MainActivity.kt`, ubicado en el paquete `com.example.material_design`. El objetivo de la aplicación es mostrar diferentes componentes visuales como listas, tarjetas, grids y barras superiores, empleando buenas prácticas de Compose.

Estructura General

El archivo contiene la clase principal `MainActivity`, varias funciones composables (`@Composable`) y estructuras de datos auxiliares como `enum class`, `data class` y listas.

Clase Principal: MainActivity

```
class MainActivity : ComponentActivity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    enableEdgeToEdge()
    setContent {
      Material_designTheme {
        Scaffold(modifier = Modifier.fillMaxSize()) { innerPadding ->
          Greeting(
            name = "Fernando",
            modifier = Modifier.padding(innerPadding)
          )
          Activity()
          AffirmationsApp()
          TopicGrid()
        }
      }
    }
  }
}
```

- Sobrescribe el método `onCreate` para establecer el contenido de la UI.
- Aplica el tema personalizado definido en `Material_designTheme`.
- Usa un `Scaffold` para estructurar la pantalla, incluyendo varias funciones composables.

Estructuras de Datos

Enum: Daypart

```
enum class Daypart {
  MORNING,
  AFTERNOON,
  EVENING,
}
```

Define los momentos del día para clasificar eventos.

Data Class: Event

```
data class Event(
  val title: String,
  val description: String? = null,
  val daypart: Daypart,
  val durationInMinutes: Int,
)
```

Representa un evento con título, descripción opcional, momento del día y duración en minutos.

Lista de Eventos

```
val events = mutableListOf<Event>(...)
```

Contiene ejemplos de eventos con distintos atributos.

Funciones Composables Principales

Greeting

Muestra un saludo personalizado.

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

Activity

Realiza operaciones de agrupamiento y filtrado sobre los eventos y muestra información por consola.

```
@Composable
fun Activity(){
  println("You have ${shortEvents.size} short events.")
  val groupedEvents = events.groupBy { it.daypart }
  groupedEvents.forEach { (daypart, events) ->
    println("$daypart: ${events.size} events")
  }
  println("Last event of the day: ${events.last().title}")
}
```

AffirmationsApp y AffirmationList

Muestran una lista de afirmaciones motivacionales usando tarjetas.

```
@Composable
fun AffirmationsApp() {
  AffirmationList(
    affirmationList = Datasource().loadAffirmations(),
  )
}
```

TopicGrid y TopicCard

Muestran un grid de temas usando tarjetas.

```
@Composable
fun TopicGrid(modifier: Modifier = Modifier) {
  LazyVerticalGrid(
    columns = GridCells.Fixed(2),
    ...
  ) {
    items(Datasource2.topics) { topic ->
      TopicCard(topic)
    }
  }
}
```

WoofApp y Componentes de Perros

Ejemplo de una lista de mascotas usando tarjetas, expandibles para ver detalles.

```
@Composable
fun WoofApp() {
  Scaffold(
    topBar = {
      WoofTopAppBar()
    }
  ) { it ->
    LazyColumn(contentPadding = it) {
      items(dogs) {
        DogItem(
          dog = it,
          modifier = Modifier.padding(dimensionResource(R.dimen.
              padding_small))
        )
      }
    }
  }
}
```

Incluye componentes como `DogIcon`, `DogInformation` y `DogHobby`.

Previews

Se incluyen funciones con `@Preview` para mostrar la UI en modo claro y oscuro, y previsualizar tarjetas individuales.

Buenas Prácticas

- Uso de temas y estilos globales.
- Separación de lógica y UI en funciones composables reutilizables.
- Agrupación y filtrado de datos antes de mostrarlos en la UI.
- Uso de previews para facilitar el diseño visual.

Dependencias y Recursos

- Jetpack Compose y Material 3.
- Recursos de imágenes y cadenas en `res/`.
- Clases de datos y fuentes externas como `Datasource`, `Datasource2`, `HeroesRepository`, etc.

1. Estructura General

El archivo proporciona una lista animada de héroes, donde cada elemento es una tarjeta (`Card`) con información y una imagen. Se emplean animaciones de entrada tanto para la lista como para cada elemento individual.

2. Componente Principal: HeroesList

```
@OptIn(ExperimentalAnimationApi::class)
@Composable
fun HeroesList(
  heroes: List<Hero>,
  modifier: Modifier = Modifier,
  contentPadding: PaddingValues = PaddingValues(0.dp),
) {
  val visibleState = remember {
    MutableTransitionState(false).apply {
      targetState = true
    }
  }

  AnimatedVisibility(
```

```
    visibleState = visibleState,
    enter = fadeIn(
      animationSpec = spring(dampingRatio = DampingRatioLowBouncy)
    ),
    exit = fadeOut(),
    modifier = modifier
  ) {
    LazyColumn(contentPadding = contentPadding) {
      itemsIndexed(heroes) { index, hero ->
        HeroListItem(
          hero = hero,
          modifier = Modifier
            .padding(horizontal = 16.dp, vertical = 8.dp)
            .animateEnterExit(
              enter = slideInVertically(
                animationSpec = spring(
                  stiffness = StiffnessVeryLow,
                  dampingRatio = DampingRatioLowBouncy
                ),
                initialOffsetY = { it * (index + 1) }
              )
            )
        )
      }
    }
  }
}
```

- Utiliza `AnimatedVisibility` para animar la aparición de la lista con `fadeIn`.
- Cada elemento de la lista (`HeroListItem`) se anima individualmente con `slideInVertically`, generando una entrada escalonada (staggered).
- Recibe como parámetros la lista de héroes, modificadores y el padding.

3. Componente: HeroListItem

```
@Composable
fun HeroListItem(
  hero: Hero,
  modifier: Modifier = Modifier
) {
  Card(
    elevation = CardDefaults.cardElevation(defaultElevation = 2.dp),
    modifier = modifier,
  ) {
    Row(
      modifier = Modifier
        .fillMaxWidth()
        .padding(16.dp)
        .sizeIn(minHeight = 72.dp)
```

```
    ) {
      Column(modifier = Modifier.weight(1f)) {
        Text(
          text = stringResource(hero.nameRes),
          style = MaterialTheme.typography.displaySmall
        )
        Text(
          text = stringResource(hero.descriptionRes),
          style = MaterialTheme.typography.bodyLarge
        )
      }
      Spacer(Modifier.width(16.dp))
      Box(
        modifier = Modifier
          .size(72.dp)
          .clip(RoundedCornerShape(8.dp))
      ) {
        Image(
          painter = painterResource(hero.imageRes),
          contentDescription = null,
          alignment = Alignment.TopCenter,
          contentScale = ContentScale.FillWidth
        )
      }
    }
  }
}
```

- Muestra la información del héroe y su imagen dentro de una tarjeta.
- Usa `Column` para los textos y `Box` para la imagen, aplicando esquinas redondeadas.
- El diseño es responsivo y mantiene un tamaño mínimo para la fila.

4. Previews

Incluye previews para facilitar la visualización en tiempo de desarrollo:

```
@Preview("Light Theme")
@Preview("Dark Theme", uiMode = Configuration.UI_MODE_NIGHT_YES)
@Composable
fun HeroPreview() { ... }

@Preview("Heroes List")
@Composable
fun HeroesPreview() { ... }
```

- `HeroPreview`: Muestra una tarjeta individual de héroe.
- `HeroesPreview`: Muestra la lista completa de héroes utilizando los datos de `HeroesRepository`.

5. Dependencias y Recursos

- Jetpack Compose, Material 3.
- Recursos de strings e imágenes referenciados por ID.
- `Hero` y `HeroesRepository` como modelo de datos.
- Temas personalizados mediante `SuperheroesTheme`.

6. Dog

6.1. Definición

```
data class Dog(
  @DrawableRes val imageResourceId: Int,
  @StringRes val name: Int,
  val age: Int,
  @StringRes val hobbies: Int
)
```

`imageResourceId` Identificador de recurso drawable para la imagen del perro.

`name` Identificador de recurso string para el nombre del perro.

`age` Edad del perro (entero).

`hobbies` Identificador de recurso string para la descripción o pasatiempos del perro.

6.2. Lista de Perros

```
val dogs = listOf(
  Dog(R.drawable.koda, R.string.dog_name_1, 2, R.string.dog_description_1)
    ,
  ...
)
```

Lista predefinida de objetos `Dog` que incluye los datos de varios perros, cada uno con su imagen, nombre, edad y descripción.

7. Hero

7.1. Definición

```
data class Hero(
  @StringRes val nameRes: Int,
  @StringRes val descriptionRes: Int,
  @DrawableRes val imageRes: Int
)
```

`nameRes` Identificador de recurso string para el nombre del héroe.

`descriptionRes` Identificador de recurso string para la descripción del héroe.

`imageRes` Identificador de recurso drawable para la imagen del héroe.

8. Topic

8.1. Definición

```
data class Topic(
  @StringRes val name: Int,
  val availableCourses: Int,
  @DrawableRes val imageRes: Int
)
```

`name` Identificador de recurso string para el nombre del tema.

`availableCourses` Número de cursos disponibles para este tema.

`imageRes` Identificador de recurso drawable para la imagen asociada al tema.

El archivo `strings.xml` contiene todos los recursos de texto utilizados en la aplicación Android `Material_design`. Estos recursos permiten la internacionalización y centralización de los textos usados en la interfaz de usuario, tarjetas, descripciones, nombres, y otros elementos.

9. Secciones de Recursos

Nombre del recurso	**Valor/Descripción**
Generales	
app_name	Material_design
Afirmaciones Motivacionales	
affirmation1	I am strong.
affirmation2	I believe in myself.
affirmation3	Each day is a new opportunity to grow and be a better version of myself.

Nombre del recurso	Valor/Descripción
affirmation4	Every challenge in my life is an opportunity to learn from.
affirmation5	I have so much to be grateful for.
affirmation6	Good things are always coming into my life.
affirmation7	New opportunities await me at every turn.
affirmation8	I have the courage to follow my heart.
affirmation9	Things will unfold at precisely the right time.
affirmation10	I will be present in all the moments that this day brings.
Temas (Topics)	
architecture	Architecture
automotive	Automotive
biology	Biology
crafts	Crafts
business	Business
culinary	Culinary
design	Design
ecology	Ecology
engineering	Engineering
fashion	Fashion
finance	Finance
film	Film
gaming	Gaming
geology	Geology
drawing	Drawing
history	History
journalism	Journalism
law	Law
lifestyle	Lifestyle
music	Music
painting	Painting
photography	Photography
physics	physics
tech	Tech
Perros (Dog Cards)	
dog_name_1	Koda
dog_name_2	Lola
dog_name_3	Frankie
dog_name_4	Nox
dog_name_5	Faye
dog_name_6	Bella
dog_name_7	Moana
dog_name_8	Tzeitel
dog_name_9	Leroy
dog_description_1	Eating treats on the terrace
dog_description_2	Barking at Daddy

Nombre del recurso	Valor/Descripción
dog_description_3	Stealing socks
dog_description_4	Meeting new animals
dog_description_5	Digging in the garden
dog_description_6	Chasing sea foam
dog_description_7	Bothering her paw-rents
dog_description_8	Sunbathing
dog_description_9	Sleeping in dangerous places
about	About:
years_old	%d years old
expand_button_content_description	Set more or less information about a dog.
Héroes (Hero Cards)	
hero1	Nick the Night and Day
description1	The Jetpack Hero
hero2	Reality Protector
description2	Understands the absolute truth
hero3	Andre the Giant
description3	Mimics the light and night to blend in
hero4	Benjamin the Brave
description4	Harnesses the power of canary to develop bravely
hero5	Magnificent Maru
description5	Effortlessly glides in to save the day
hero6	Dynamic Yasmine
description6	Ability to shift to any form and energize

Conclusión

Este archivo sirve como un ejemplo integral de cómo estructurar una aplicación Compose moderna, incluyendo manejo de listas, tarjetas, agrupamiento de datos y uso de temas de Material Design. Las prácticas aquí descritas promueven código modular, reutilizable y fácil de mantener.