Consumo de API REST

Modelar JSON a orientación a objetos

{desafío}
latam_

Construir una aplicación Android que consume un servicio REST actualizando la interfaz de usuario, acorde al lenguaje Kotlin y a la librería Retrofit

- Unidad 1:
  Acceso a datos en Android
- Unidad 2:
  Consumo de API REST
- Unidad 3:
  Testing
- Unidad 4:
  Distribución del aplicativo Android

{desafío} latam_

¿Qué aprenderás en esta sesión?

- *Modelar JSON a orientación a objetos*

¿Cuál crees tú que es el propósito de modelar datos JSON como objetos en lenguajes de programación?

{desafío}
latam_

/* Modelar JSON a orientación a objetos */

{desafío}
latam_

JSON a orientación a objetos

Modelar datos JSON como objetos en un lenguaje de programación orientado a objetos te permite trabajar con los datos de una manera más estructurada y conveniente. De esta forma, puedes representar los datos JSON como clases, con propiedades que corresponden a las claves de los datos JSON y valores que corresponden a los valores de las claves.

JSON a orientación a objetos

Por ejemplo, considera los siguientes datos JSON que representan una lista de estudiantes con sus nombres, calificaciones y direcciones:

```
[
  {
    "name": "Alice",
    "grade": 85,
    "address": { "street": "123 Main St", "city": "Anytown", "state": "CA" }
  },
  {
    "name": "Bob",
    "grade": 90,
    "address": { "street": "456 Elm St", "city": "Anothertown", "state": "NY" }
  }
]
```

En Kotlin...

```
data class Address(
val street: String,
val city: String,
val state: String
)

data class Student(
val name: String,
val grade: Int,
val address: Address
)

val students = listOf(
    Student("Alice", 85, Address("123 Main St", "Anytown", "CA")),
    Student("Bob", 90, Address("456 Elm St", "Anothertown", "NY"))
)
```

Propiedades clase Address

Aquí, la clase `Address` modela la dirección de cada estudiante y contiene tres propiedades: `street`, `city` y `state`. La clase `Student` modela a cada estudiante y contiene tres propiedades: nombre, grado y dirección. La lista de alumnos (`students`) se crea como una Lista de objetos de `Student`.

Con esta implementación, puedes acceder y manipular fácilmente los datos como objetos. Por ejemplo, puede acceder al nombre del primer estudiante como `student[0].name`, y puede cambiar la dirección del segundo estudiante de la siguiente manera:

```
students[1].address = Address("789 Oak St", "Yetanothertown", "FL")
```

Métodos en las clases

Puede agregar métodos a las clases para realizar operaciones en los datos. Por ejemplo, puedes agregar un método a la clase `Student` para calcular la calificación promedio de todos los estudiantes:

```
fun averageGrade(students: List<Student>): Double {
    return students.map { it.grade }.average()
}

val average = averageGrade(students)
println("The average grade is $average")
```

Al modelar los datos JSON como objetos, puede trabajar con los datos de una manera más intuitiva y estructurada, lo que hace que su código sea más fácil de leer, mantener y ampliar.

Demostración "Lista de libros"

Demo - Lista de libros

Considera los siguientes datos JSON que representan una lista de libros, cada uno con su título, autor, año de publicación y una lista de géneros:

```
[
  {
    "title": "The Great Gatsby",
    "author": "F. Scott Fitzgerald",
    "year": 1925,
    "genres": ["Fiction", "Romance", "Tragedy"]
  },
  {
    "title": "To Kill a Mockingbird",
    "author": "Harper Lee",
    "year": 1960,
    "genres": ["Fiction", "Classics", "Drama"]
  }
]
```

Demo - Lista de libros

En Kotlin, puede modelar estos datos JSON como las siguientes clases de datos:

```
data class Book(
    val title: String,
    val author: String,
    val year: Int,
    val genres: List<String>
)

val books = listOf(
    Book("The Great Gatsby", "F. Scott Fitzgerald", 1925, listOf("Fiction", "Romance",
"Tragedy")),
    Book("To Kill a Mockingbird", "Harper Lee", 1960, listOf("Fiction", "Classics", "Drama"))
)
```

Demo - Lista de libros

Con esta implementación, puedes acceder y manipular fácilmente los datos como objetos. Por ejemplo, puedes acceder al título del primer libro como `books[0].title`, y puede agregar un nuevo género al segundo libro de la siguiente manera:

```
books[1].genres += "Literary Fiction"
```

Además, puede agregar métodos a las clases para realizar operaciones en los datos. Por ejemplo, puede agregar un método a la clase Libro para buscar todos los libros que se publicaron en un año determinado:

Demo - Lista de libros

```
fun booksPublishedInYear(books: List<Book>, year: Int): List<Book>{
    return books.filter { it.year == year }
}

val booksPublishedIn1960 = booksPublishedInYear(books, 1960)
println("Books published in 1960:")
for (book in booksPublishedIn1960) {
    println("- ${book.title} by ${book.author}")
}
```

Al modelar los datos JSON como objetos, puede trabajar con los datos de una manera más intuitiva y estructurada, lo que hace que su código sea más fácil de leer, mantener y ampliar.

/* Reflexionando sobre el tema */

Ventajas del modelado de datos JSON

- Está claro que el modelado de datos JSON como objetos en lenguajes de programación ofrece varias ventajas. Al representar los datos JSON como objetos, puede trabajar con los datos de una manera más intuitiva y estructurada, lo que hace que su código sea más fácil de leer, mantener y ampliar.

- También puedes agregar métodos a las clases para realizar operaciones en los datos, lo que hace que su código sea más modular y reutilizable.

Ventajas del modelado de datos JSON

- Además, la programación orientada a objetos brinda la capacidad de encapsular datos y comportamientos en objetos, lo que le permite ocultar detalles de implementación y crear una interfaz clara y concisa para trabajar con los datos. Esto puede ayudar a reducir la cantidad de código repetitivo y hacer que su código sea más legible y fácil de mantener.

- Otra ventaja de modelar datos JSON como objetos es que proporciona un mapeo claro entre los datos y el código, lo que facilita el trabajo con datos JSON en un lenguaje con seguridad de tipos y tipado estático. Esto reduce el riesgo de errores en tiempo de ejecución y facilita la detección de problemas en tiempo de compilación.

Ventajas del modelado de datos JSON

- En general, modelar datos JSON como objetos es un enfoque poderoso y flexible que puede mejorar en gran medida la calidad y la capacidad de mantenimiento de tu código. Ya sea que estés trabajando en un lenguaje de tipo estático como Kotlin o en un lenguaje de tipo dinámico como Python, los conceptos de la programación orientada a objetos se pueden aplicar para hacer que su código sea más organizado y legible.

¿Cuáles son los beneficios de modelar datos JSON como objetos en lenguajes de programación?

Próxima sesión...

- *Actualizar la interfaz de usuario*

{desafío}
latam_

Academia de
talentos digitales