COLECCIONES KOTLIN

1. Colecciones en Kotlin

Las colecciones en Kotlin son estructuras de datos que te permiten almacenar elementos de manera organizada. Existen diferentes tipos de colecciones como listas, conjuntos y mapas. Las colecciones se dividen en dos tipos principales:

- **Colecciones inmutables**: No pueden ser modificadas después de ser creadas (como `listOf()`, `setOf()`, `mapOf()`).
- **Colecciones mutables**: Puedes agregar, eliminar o modificar elementos (como `mutableListOf()`, `mutableSetOf()`, `mutableMapOf()`).

Ejemplo básico de colección:

```
val list = listOf(1, 2, 3, 4)
val set = setOf(1, 2, 3, 4)
val map = mapOf(1 to "one", 2 to "two")
```

2. Funciones de Orden Superior

Las funciones de orden superior son funciones que toman otras funciones como argumentos o devuelven funciones. Algunas de las funciones más comunes en Kotlin son `map`, `filter`, `groupBy`, `sortedBy`, y `forEach`.

3. `forEach`

`forEach` es una función de orden superior que permite iterar sobre una colección y realizar una acción con cada elemento. Es una forma concisa de recorrer elementos sin usar un bucle clásico como `for`.

Sintaxis:

```
collection.forEach { item ->
// Acción a realizar con cada elemento (item)
}
```

Ejemplo de uso:

```
val numbers = listOf(1, 2, 3, 4, 5)
numbers.forEach { number ->
    println(number))
}
```

Esto imprimirá:

```
1
2
3
4
5
```

En los ejemplos de los ejercicios, `forEach` se usa para recorrer listas de libros o productos y hacer algo con cada elemento.

4. Funciones más utilizadas con colecciones

- `groupBy`: Agrupa los elementos de una colección por una clave especificada.

  Ejemplo:

  ```
  kotlin
  Copiar código
  val booksByCategory = books.groupBy { it.category }
  booksByCategory.forEach { (category, booksInCategory) ->
      println("$category: ${booksInCategory.size} libros")
  }
  ```

- `filter`: Filtra los elementos de la colección que cumplen con una condición.

  Ejemplo:

  ```
  kotlin
  Copiar código
  ```

```
val affordableBooks = books.filter { it.isAffordable(15.
00) }
affordableBooks.forEach { book ->
    println("- ${book.title} por ${book.author} - $${boo
k.price}")
}
```

- `sortedBy` y `sortedWith`: Ordena los elementos de la colección de acuerdo a una propiedad.

  Ejemplo con `sortedBy`:

  ```
  kotlin
  Copiar código
  val sortedBooks = books.sortedBy { it.price }
  sortedBooks.forEach { book ->
      println("${book.title} - $${book.price}")
  }
  ```

  Ejemplo con `sortedWith`:

  ```
  kotlin
  Copiar código
  val booksOrderedByPrice = books.sortedWith(compareBy { i
  t.price })
  booksOrderedByPrice.forEach { book ->
      println("${book.title} - $${book.price}")
  }
  ```

- `maxByOrNull`: Encuentra el elemento de la colección con el valor máximo de acuerdo a una propiedad.

  Ejemplo:

  ```
  kotlin
  Copiar código
  val mostExpensiveBook = books.maxByOrNull { it.price }
  ```

```
mostExpensiveBook?.let { book ->
    println("El libro más caro es: ${book.title} - $${bo
ok.price}")
}
```

5. Uso de `let`, `apply`, `run`, `also` y `with`

Estas funciones son conocidas como funciones de alcance (scope functions) y permiten manipular objetos de manera concisa y eficiente. Son útiles para aplicar cambios dentro de un bloque de código específico.

- `let`: Se utiliza para ejecutar un bloque de código con el objeto actual y devuelve el resultado del bloque.

  Ejemplo:

  ```
  kotlin
  Copiar código
  mostExpensiveBook?.let { book ->
      println("El libro más caro es: ${book.title} - $${bo
  ok.price}")
  }
  ```

- `apply`, `run`, `also` y `with` funcionan de manera similar, pero tienen diferencias en la manera en que retornan el objeto o el resultado.

6. Ejemplos de ejercicios completos

Ejercicio 1: Libros

```
// Enumeración para las categorías de libros
enum class Category {
    FICTION, NON_FICTION, SCIENCE, HISTORY
}

// Clase Book con un método de extensión
data class Book(
```

```
    val title: String,
    val category: Category,
    val author: String,
    val price: Double
)

// Método de extensión para comprobar si el libro está dent
ro del presupuesto
fun Book.isAffordable(budget: Double): Boolean {
    return this.price <= budget
}

fun main() {
    val books = listOf(
        Book("El Gran Gatsby", Category.FICTION, "F. Scott
Fitzgerald", 10.99),
        Book("Breve Historia del Tiempo", Category.SCIENCE,
"Stephen Hawking", 15.99),
        Book("El Arte de la Guerra", Category.HISTORY, "Sun
Tzu", 8.99),
        Book("Sapiens", Category.NON_FICTION, "Yuval Noah H
arari", 20.00),
        Book("Matar a un Ruiseñor", Category.FICTION, "Harp
er Lee", 12.50),
        Book("El Gen Egoísta", Category.SCIENCE, "Richard D
awkins", 14.99)
    )

    // 1. Agrupar libros por categoría y mostrar cuántos ha
y en cada una
    val booksByCategory = books.groupBy { it.category }
    booksByCategory.forEach { (category, booksInCategory) -
>
        println("$category: ${booksInCategory.size} libro
s")
    }

    // 2. Filtrar libros dentro de un presupuesto específic
```

```
o
    val affordableBooks = books.filter { it.isAffordable(1
5.00) }
    affordableBooks.forEach { book ->
        println("- ${book.title} por ${book.author} - $${bo
ok.price}")
    }

    // 3. Ordenar libros por precio y mostrar el más caro
    val booksOrderedByPrice = books.sortedWith(compareBy {
it.price })
    val mostExpensiveBook = booksOrderedByPrice.last()
    println("El libro más caro es: ${mostExpensiveBook.titl
e} por ${mostExpensiveBook.author} - $${mostExpensiveBook.p
rice}")
}
```

Ejercicio 2: Productos

```
//Enumeración para los tipos de productos
enum class ProductType {
    ELECTRONICS, GROCERIES, CLOTHING, BEAUTY
}

// Clase Product con un método de extensión
data class Product(
    val name: String,
    val type: ProductType,
    val stock: Int,
    val price: Double
)

// Método de extensión para comprobar si el producto está e
n stock
fun Product.isInStock(): Boolean {
    return this.stock > 0
}
```

```
fun main() {
    val products = listOf(
        Product("Laptop", ProductType.ELECTRONICS, 5, 1000.
0),
        Product("Shampoo", ProductType.BEAUTY, 20, 4.99),
        Product("T-shirt", ProductType.CLOTHING, 15, 19.9
9),
        Product("Bananas", ProductType.GROCERIES, 50, 0.9
9),
        Product("Headphones", ProductType.ELECTRONICS, 10,
99.99),
        Product("Jeans", ProductType.CLOTHING, 0, 39.99)
    )

    // 1. Mostrar productos ordenados por precio
    val productsSortedByPrice = products.sortedBy { it.pric
e }
    productsSortedByPrice.forEach { product ->
        println("- ${product.name} - $${product.price}")
    }

    // 2. Agrupar productos por tipo
    val productsByType = products.groupBy { it.type }
    productsByType.forEach { (type, productsList) ->
        println("$type: ${productsList.size} productos")
    }

    // 3. Filtrar productos en stock
    val inStockProducts = products.filter { it.isInStock()
}
    inStockProducts.forEach { product ->
        println("- ${product.name} (Stock: ${product.stoc
k})")
    }

    // 4. Encontrar el producto más caro
    val mostExpensiveProduct = products.maxByOrNull { it.pr
```

```
ice }
    mostExpensiveProduct?.let { product ->
        println("El producto más caro es: ${product.name} -
Tipo: ${product.type} - Precio: $${product.price}")
    }
}
```

7. Resumen de Funciones importantes

- `groupBy`: Agrupa elementos por una propiedad.
- `filter`: Filtra elementos que cumplen una condición.
- `sortedBy`: Ordena elementos según una propiedad.
- `forEach`: Itera sobre una colección y realiza una acción por cada elemento.