Bucle Canciones Kotlin

A continuación, explicamos la implementación y creación de un código simple en KOTLIN, que permite utilizar las listas circulares en un programa que simula la reproducción en bucle de 5 canciones en una playlist (lista circular). Se utilizan dos clases: **ListaCircular** y **Nodo**

Optimización

Hemos adaptado el código para Kotlin, manteniendo la misma lógica que en Java. La clase `Nodo` ahora usa propiedades de Kotlin, y la clase `ListaCircular` utiliza las funciones de repetición (`repeat`) para simplificar el bucle de reproducción

Resultado Esperado

Al ejecutar el programa, deberías ver algo similar a esto en la consola:

```
Iniciando reproducción:
Reproduciendo: Nice Guy - BOYNEXTDOOR
Reproduciendo: 0X1=LOVESONG (I Know I Love You) feat. Seori -
Reproduciendo: Style (Taylor´s Version) - Taylor Swift
Reproduciendo: Rock with you - SEVENTEEN
Reproduciendo: Espresso - Sabrina Carpenter
Reproduciendo: Nice Guy - BOYNEXTDOOR
Reproduciendo: 0X1=LOVESONG (I Know I Love You) feat. Seori -
Reproduciendo: Style (Taylor´s Version) - Taylor Swift
Reproduciendo: Rock with you - SEVENTEEN
Reproduciendo: Espresso - Sabrina Carpenter
...
```

Clase `Nodo`

```
class Nodo(val cancion: String) {
    var siguiente: Nodo? = null
```

```
}
```

Explicación

1. **Definición de la Clase**:

   ```
   class Nodo(val cancion: String) {
   ```

   - `class Nodo(val cancion: String)`: Declara una clase llamada `Nodo` con un constructor primario que toma un parámetro `cancion` de tipo `String`. La palabra clave `val` indica que `cancion` es una propiedad de solo lectura.

2. **Propiedad `siguiente`**:

   ```
   var siguiente: Nodo? = null
   ```

   - `var siguiente: Nodo? = null`: Declara una propiedad mutable `siguiente` de tipo `Nodo?` (puede ser `Nodo` o `null`). Inicialmente, se establece en `null`, indicando que este nodo no apunta a ningún otro nodo por defecto.

La clase `Nodo` en Kotlin es similar a su contraparte en Java, pero aprovecha las características de Kotlin para una sintaxis más concisa. Cada instancia de `Nodo` representa un elemento de la lista, almacenando una canción y una referencia al siguiente nodo. Esto permite que los nodos se conecten entre sí, formando una estructura circular.

Clase `ListaCircular`

```
class ListaCircular {
    private var inicio: Nodo? = null
    private var ultimo: Nodo? = null

    fun agregarCancion(cancion: String){
        val nuevoNodo = Nodo(cancion)
        if (inicio == null) {
            inicio = nuevoNodo
```

```
            ultimo = nuevoNodo
            ultimo?.siguiente = inicio
        } else {
            ultimo?.siguiente = nuevoNodo
            ultimo = nuevoNodo
            ultimo?.siguiente = inicio
        }
    }

    fun reproducir(veces: Int){
        if (inicio != null){
            var actual = inicio
            repeat(veces){
                do {
                    println("Reproduciendo: ${actual?.canci
on}")
                    actual = actual?.siguiente
                } while (actual != inicio)
            }
        }
    }
}
```

Explicación

1. **Definición de la Clase**:

```
class ListaCircular {
```

   - `class ListaCircular`: Declara una clase llamada `ListaCircular`.

2. **Atributos de la Clase**:

```
private var inicio: Nodo? = null
private var ultimo: Nodo? = null
```

   - `private var inicio: Nodo? = null`: Declara una propiedad mutable `inicio` de tipo `Nodo?` (puede ser `Nodo` o `null`). Inicialmente, se establece en `null`, indicando que la lista está vacía al principio.

- `private var ultimo: Nodo? = null`: Declara una propiedad mutable `ultimo` de tipo `Nodo?`, también inicializada a `null`.

3. **Método `agregarCancion`**:

```
fun agregarCancion(cancion: String){
    val nuevoNodo = Nodo(cancion)
    if (inicio == null) {
        inicio = nuevoNodo
        ultimo = nuevoNodo
        ultimo?.siguiente = inicio
    } else {
        ultimo?.siguiente = nuevoNodo
        ultimo = nuevoNodo
        ultimo?.siguiente = inicio
    }
}
```

- `fun agregarCancion(cancion: String)`: Define una función que agrega una nueva canción a la lista.
- `val nuevoNodo = Nodo(cancion)`: Crea un nuevo nodo con la canción proporcionada.
- `if (inicio == null)`: Verifica si la lista está vacía.
  - Si está vacía, el nuevo nodo se convierte en `inicio` y `ultimo`, y se cierra el ciclo apuntando `ultimo?.siguiente` a `inicio`.
- `else`: Si la lista no está vacía, el nuevo nodo se agrega al final y se actualiza `ultimo` para cerrar el ciclo nuevamente.

4. **Método `reproducir`**:

```
fun reproducir(veces: Int){
    if (inicio != null){
        var actual = inicio
        repeat(veces){
            do {
                println("Reproduciendo: ${actual?.cancio
n}")
                actual = actual?.siguiente
```

```
            } while (actual != inicio)
        }
    }
}
```

- `fun reproducir(veces: Int)`: Define una función que reproduce las canciones en la lista un número específico de veces.
- `if (inicio != null)`: Verifica si la lista no está vacía.
- `var actual = inicio`: Inicializa una variable `actual` que comienza en el `inicio` de la lista.
- `repeat(veces)`: Un bucle que se ejecuta el número de veces especificado.
    - `do { ... } while (actual != inicio)`: Un bucle `do-while` que recorre la lista circular, imprimiendo cada canción y avanzando al siguiente nodo hasta que vuelve al `inicio`.

La clase `ListaCircular` en Kotlin maneja la lógica para agregar nodos y reproducir las canciones en un ciclo continuo. Los métodos `agregarCancion` y `reproducir` permiten gestionar y recorrer la lista circular, respectivamente.

Función `main`

```
fun main() {
    val listaCanciones = ListaCircular()

    // Agregar canciones a la lista circular
    listaCanciones.agregarCancion("Nice Guy - BOYNEXTDOOR")
    listaCanciones.agregarCancion("0X1=LOVESONG (I Know I L
ove You) feat. Seori - TOMORROW X TOGETHER")
    listaCanciones.agregarCancion("Style (Taylor´s Version)
- Taylor Swift")
    listaCanciones.agregarCancion("Rock with you - SEVENTEE
N")
    listaCanciones.agregarCancion("Espresso - Sabrina Carpe
```

```
nter")

    // Simular la reproducción en bucle
    println("Iniciando reproducción:")
    listaCanciones.reproducir(5)
}
```

Explicación

1. **Definición de la Función `main`**:

   ```
   fun main() {
   ```

   - `fun main()`: Define la función `main`, que es el punto de entrada de cualquier aplicación Kotlin.

2. **Creación de la Lista Circular**:

   ```
   val listaCanciones = ListaCircular()
   ```

   - `val listaCanciones = ListaCircular()`: Crea una nueva instancia de `ListaCircular` llamada `listaCanciones`. La palabra clave `val` indica que `listaCanciones` es una referencia inmutable a la instancia de `ListaCircular`.

3. **Agregar Canciones a la Lista**:

   ```
   listaCanciones.agregarCancion("Nice Guy - BOYNEXTDOOR")
   listaCanciones.agregarCancion("0X1=LOVESONG (I Know I Lo
   ve You) feat. Seori - TOMORROW X TOGETHER")
   listaCanciones.agregarCancion("Style (Taylor´s Version)
   - Taylor Swift")
   listaCanciones.agregarCancion("Rock with you - SEVENTEE
   N")
   listaCanciones.agregarCancion("Espresso - Sabrina Carpen
   ter")
   ```

   - `listaCanciones.agregarCancion(...)`: Llama a la función `agregarCancion` de `ListaCircular` para agregar varias canciones a la lista. Cada llamada crea

un nuevo nodo en la lista circular con la canción especificada.

4. **Simular la Reproducción en Bucle**:

```
println("Iniciando reproducción:")
listaCanciones.reproducir(5)
```

- `println("Iniciando reproducción:")`: Imprime un mensaje en la consola indicando que la reproducción está comenzando.
- `listaCanciones.reproducir(5)`: Llama a la función `reproducir` de `ListaCircular` para reproducir las canciones en la lista 5 veces. Esta función recorre la lista circular y imprime cada canción en un bucle.

La función `main` en Kotlin crea una instancia de `ListaCircular`, agrega varias canciones a la lista y luego simula la reproducción de estas canciones en un bucle continuo. Este enfoque permite ver cómo se comporta la lista circular en una situación práctica, como la reproducción de música en un servicio de streaming.
