Universidad Don Bosco.

Campus Virtual

Asignatura:

Desarrollo de Software para Móviles DSM941

Grupo Teórico:

01

Proyecto 1

Integrantes:

Alas Linares Alejandro Antonio AL192188

De Paz Velásquez Vicente Daniel DV192307

Duran Meléndez Gilberto Emmanuel DM192201

Gutiérrez Borja Rafael Armando GB192205

Docente:

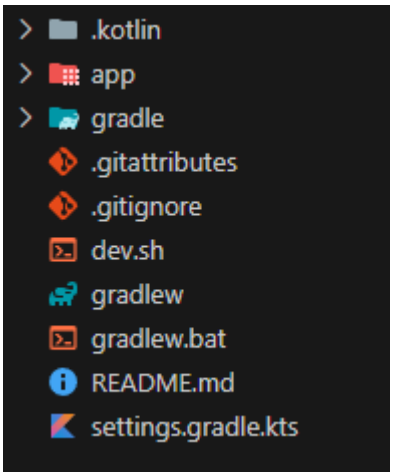
Alexander Alberto Sigüenza Campos

Fecha: lunes 30 de septiembre del 2024

Contenido

Estructura del código

El proyecto ha sido creado usando gradle así que tiene la estructura normal de un proyecto hecho con gradle:



- **.kotlin:** carpeta generada automáticamente por el compilador, guarda información interna relacionada con la configuración del proyecto como los caches de compilación.
- **app:** En esta carpeta se guardan los archivos principales de la aplicación.
    - **src/main:** dentro de esta carpeta se encuentra el código fuente, como el proyecto está hecho en **kotlin**, entonces todos los archivos fuente se encuentran dentro de una carpeta llamada *kotlin*, seguido de la estructura del nombre de nuestro paquete, en nuestro caso sería: *org/example*. Nuestra *main class* se llama **App.kt**
    - **src/test**: dentro de esta carpeta de encuentran los test unitarios, para el caso de este proyecto no han sido implementados ya que no era un requerimiento.
    - **src/build.gradle.kts:** Este archivo es el script de configuración para la compilación del módulo de la aplicación. Aquí se definen las dependencias, configuraciones específicas para Android, versiones del SDK, configuraciones de compilación, etc.

- **gradle:** Contiene archivos y configuraciones necesarias para la correcta ejecución de Gradle en el proyecto.
- **.gitattributes**: Archivo de configuración de git que indica a git cómo guardar los finales de línea en archivos de Linux, Windows o jar.
- **.gitignore**: Archivo de configuración de git que indica cuáles archivos no se deben subir al repo.
- **dev.sh**: Script de bash que facilita la ejecución local del proyecto para un desarrollo más rápido.
- **gradlew**: Script de inicio de gradle diseñado para consolas POSIX
- **gradlew.bat**: Script de inicio de gradle diseñado para Windows
- **README.md**: Archivo markdown que contiene información básica del proyecto.
- **settings.gradle.kts**: Archivo de configuración generado automáticamente por gradle que define los ajustes globales del proyecto.

¿Cómo ejecutar la aplicación?

Para poder ejecutar el proyecto se debe tener instaladas las siguientes herramientas: **java**, **kotlin** y **gradle**.

Estas herramientas pueden ser instaladas fácilmente en sistemas **UNIX** utilizando SDKMan.

Si se está usando un sistema UNIX se puede ejecutar el script bash que se encuentra en el *root* del proyecto para ejecutar el proyecto de manera sencilla:

```
1. sh dev.sh
2.
```

Si no, también se puede ejecutar de la siguiente manera:

1. Compilar el archivo jar con el comando:

```
1. ./gradlew clean shadowJar
2.
```

O también:

```
1. gradlew.bat clean shadowJar
2.
```

2. Se creará un archivo fatjar dentro del directorio *build* que se puede ejecutar con java:

```
1. java -jar ./app/build/libs/app-all.jar
2.
```

De esta manera el aplicativo ya estará corriendo en nuestra consola.

Menu.kt

La clase Menu es la clase que controla cómo se comporta la aplicación, cuenta con dos variables que sirven para controlar las opciones del usuario, así como los textos a desplegarse dentro del flujo principal, estas variables son:

```
1.  val menuOpt = mutableMapOf<Int, (List<ProductoInventario>, Carrito) -> Unit>()
2.  val menuList = listOf(
        "1. Agregar producto al carrito",
        "2. Eliminar producto del carrito",
        "3. Ver carrito",
        "4. Mostrar factura",
        "5. salir"
    )
```

La variable “menuOpt” es un mapa mutable que contiene un mapeo de enteros y funciones, el entero que funciona como llave del mapa hace referencia al índice de la opción desplegada en “menuList”, las funciones que se almacenan dentro del mapa son funciones que reciben una lista de la clase “ProductoInventario” y un objeto de la clase Carrito estas funciones no devuelven ningún valor y funcionan exclusivamente como los puntos de entrada para las acciones de cada una de las opciones de la lista de opciones. Haciendo uso de funciones de primer orden las funciones junto con las opciones y sus encabezados estarán disponibles para ser utilizados en cualquier clase ejecutable lo que facilita su reutilización. La clase cuenta con un constructor que inicializa el mapa de funciones.

Las funciones disponibles dentro de la clase son:

1. “addToCarrito”

Está función agrega elementos al carrito que se envía como parámetro a la función, hace uso de la lista de inventario con el propósito de mostrar los elementos disponibles en el inventario y seleccionar el id del elemento correspondiente y agregarlo al carrito.

2. “removeFromCarrito”

Está función solicita el id del producto y envía ese id como parámetro al método “getProductoEnCarrito”, esté método devuelve un producto que se utiliza para dar feedback al usuario del elemento seleccionado y luego solicita la cantidad de elementos a eliminar, en caso de ingresar una cantidad superior a la registrada en el carrito da un error y regresa a la página de inicio, en caso de una entrada correcta la clase usará el método “eliminarProducto” de la clase carrito, el cual eliminará la cantidad y el producto seleccionado.

3. “printFactura”

Esta función genera y muestra una factura con los productos en el carrito. Si el carrito está vacío, informa al usuario; de lo contrario, crea una tabla con detalles

de cada producto (ID, nombre, precio unitario, cantidad y subtotal). Calcula el subtotal, el impuesto según el IVA del 13%, y el total con impuestos. Luego, muestra la factura con todos estos detalles y agradece al usuario por la compra. Una vez mostrada esa información la clase limpia el carrito y elimina la cantidad de productos seleccionados del inventario y devuelve al usuario a la página principal.

App.kt

La clase principal de la aplicación encargada de instanciar los objetos necesarios para el funcionamiento de la aplicación, es la encargada de instanciar el ciclo principal de la aplicación y controlar el flujo de decisiones de los usuarios. La clase cuenta con 3 funciones, las encargadas de mostar headers y el inventario, y la función main que contiene el flujo de la aplicación.

1. "imprimirHeader"

Como su nombre indida está función es la encargada de mostrar el mensaje de bienvenida de la aplicación. Su funcinamiento es simple y tiene un unico objetivo, mantiene una filosofia de función pura pues no genera ningún tipo de side effect.

2. "imprimirInventario"

Una función que recibe como parametro el inventario de la aplicación y hace uso de la librería de AsciiTable para generar una tabla que ayude a visualizar los productos disponibles al usuario.

```
1.  val menuOpt = mutableMapOf<Int, (List<ProductoInventario>, Carrito) -> Unit>()
2.  println("Presentamos nuestro catálogo de productos:")
3.
4.  println()
5.
6.  val at = AsciiTable()
7.
8.  // Añadir encabezados de columna
9.  at.addRule()
10. at.addRow("ID - Producto", "Precio", "Cantidad disponible")
11. at.addRule()
12.
13. // Añadir filas
14. inventario.forEach { producto ->
15.      at.addRow(
16.            "${producto.producto.id} - ${producto.producto.nombre}",
17.            "$${producto.producto.precio}",
18.            "${producto.cantidadDisponible}"
19.      )
20. }
21.
22. at.addRule()
23.
24. // Imprimir la tabla
25. println(at.render())
```

3. "main"

Instancia los objetos necesarios para el funcionamiento de la aplicación.

```
1.  val menuOpt = mutableMapOf<Int, (List<ProductoInventario>, Carrito) -> Unit>()
2.  val inventario = crearInventario()
3.  val carrito = Carrito()
4.  var opt = 0
5.  val menu = Menu()
```

Posteriormente hace uso de un ciclo do while para el funcionamiento de un ciclo inicial que controla todo el flujo de la aplicación.

```
1.  do {
2.     //application flow
3.  } while(opt != 5)
```

Dentro del flujo se imprimen el header y el inventario, y se hace uso de la clase menu, que cuenta con un parámetro llamado menuOpt y otro llamado menuList, haciendo uso de un foreach se muestran las opciones de flujo disponibles y se asocian a las funciones para ser invocadas. Se pide al usuario seleccionar una opción y en caso de un error el ciclo se reinicia, de ser exitoso se ejecuta la función y al terminar esa ejecución se reinicia el ciclo. La aplicación se detiene cuando se selecciona la opción de salir, correspondiente al número 5.

```
1.  menu.cleanScreen()
2.
3.  imprimirHeader()
4.
5.  imprimirInventario(inventario)
6.
7.  println("")
8.
9.  for (option in menu.menuList) {
10.      println(option)
11. }
12.
13. println("")
14. print("Ingrese la opción que desea realizar: ")
15.
16. opt = readLine()?.toIntOrNull() ?: continue
17.
18. if(!menu.menuOpt.containsKey(opt)) {
19.      menu.cleanScreen()
20.      continue
21. }
22.
23. menu.menuOpt[opt]?.let { it(inventario, carrito) };
24. menu.cleanScreen()
```