Primer proyecto: “Carrito de compras”

Alumnos:

- **Alexandra María Henríquez Miranda HM232507**
- **Kallahan Andrea Salas Bojorquez SB210537**
- **Fernando Josué Montano González MG210111**
- **David Gerardo Menjívar Ramírez MR210455**
- **Jairo Rafael Colocho Díaz CD210488**
- **Bryan Josué Alberto Elena AE210567**

Desarrollo de Software para Móviles DSM941 G01T

Ing. Alexander Alberto Sigüenza Campos

29 de septiembre de 2024

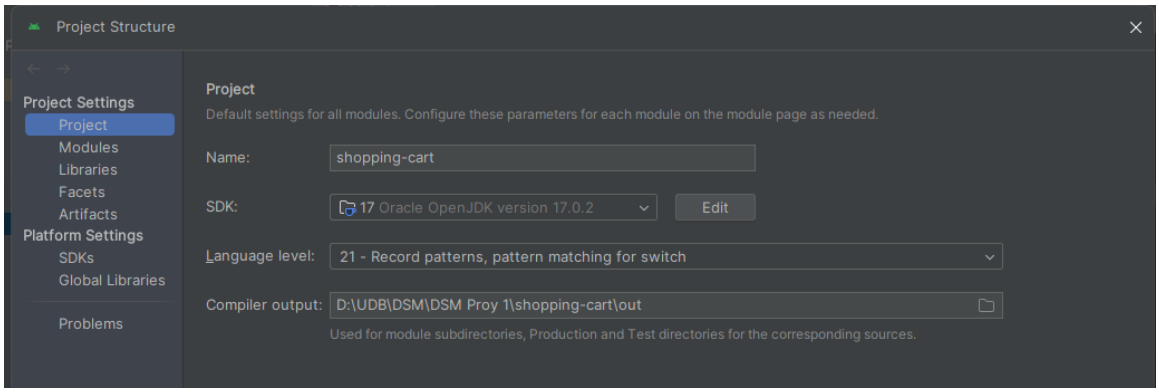
Contenido

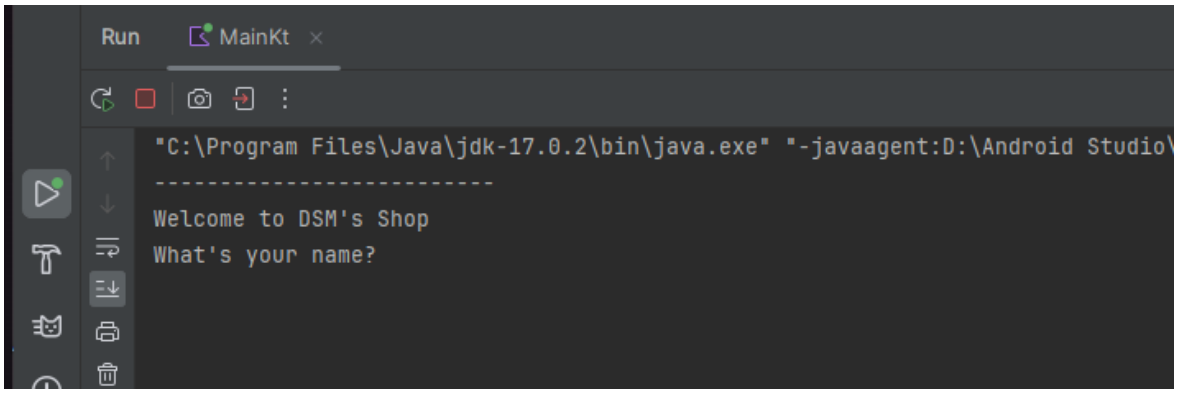
Introducción

La finalidad de este documento es explicar de manera detallada el código fuente de nuestro proyecto de carrito de compras. Este lo hemos dividido en varias partes como lo son los controladores, las excepciones, los modelos y vistas y cada una de estas organizadas de manera en la cual podamos seguir una modelo vista controlador. Lo que haremos es presentar una descripción detallada de cada archivo del proyecto, acompañados de los fragmentos de código que nos ayudaran a comprender mejor cómo estos funcionan

Ejecutar aplicación

1. **Instalación de Java JDK:** Para poder ejecutar el programa debemos asegurarnos de tener un JDK instalado en el sistema. Para este proyecto utilizamos el amazon coretto JDK, versión 22. Una vez descargado e instalado, verificamos sea la versión correcta ejecutando java -version en una terminal.
2. **Instrucciones de ejecución:** Debemos abrir el proyecto en un entorno compatible con Kotlin y Java. El proyecto tiene que estar utilizando un jdk correcto como mencionábamos anteriormente. Para compilar y ejecutar, únicamente iniciamos la clase Main.kt
3. **Interacción con la aplicación:** Una vez la hayamos iniciado, el sistema nos va a pedir un nombre y a continuación nos va a mostrar un menú con las siguientes opciones: Ver productos, añadir productos al carrito, ver carrito, proceder al pago, y eliminar productos del carrito. Este flujo va a continuar hasta que el usuario decida salir de la aplicación.





Main.kt

Este archivo es el punto inicial de nuestro sistema. Su función es iniciar con la interfaz de usuario e iniciar el flujo del programa. Cuando ejecutamos la app, lo primero que se hace es crear una instancia de la clase MainView, representando la vista principal del programa y luego invocamos el método init(), el cual es el encargado de iniciar la interacción con el usuario.

```
import View.MainView


fun main() {
  val mainView = MainView();
  mainView.init();
}
```

MainView.kt

Este archivo es importante porque tiene la lógica relacionada con la interacción del usuario con el sistema. En esta clase gestionamos la interfaz principal del sistema y le permitimos al usuario ver los productos, agregar o eliminar artículos del carrito y luego de eso, proceder con el pago. El método init() solicita el nombre de del usuario y luego llama al método mainMenu(), que lo que hace es mostrar las opciones disponibles como ver los productos, ver el carrito, realizar la compra o eliminar productos del carrito.

Con el uso del bloque “when”, el sistema responde según la selección del usuario, si el usuario decide ver los productos, entonces invoca al método viewProducts() y si este selecciona realizar la compra, entonces se gestiona el proceso a través de cartController, que es el que maneja la lógica del pago.

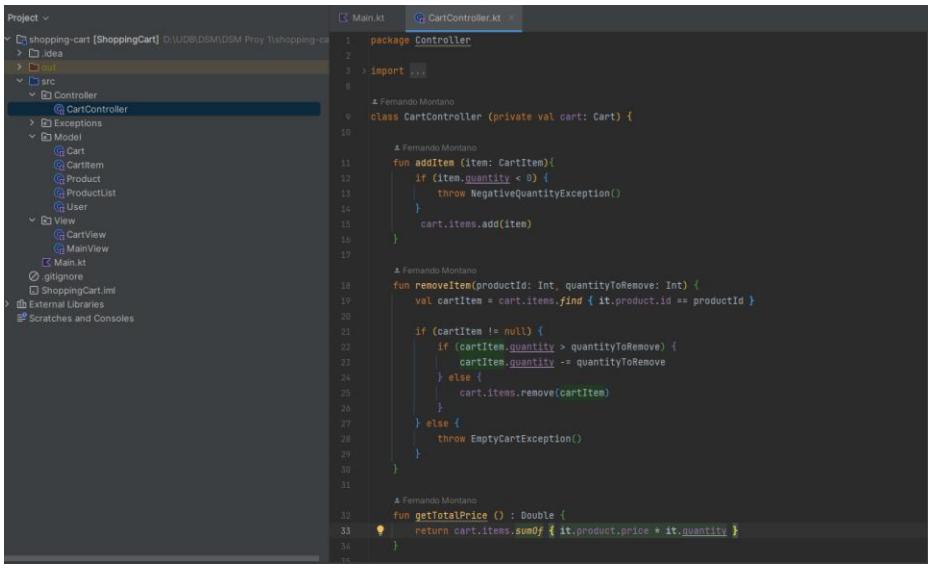
```
class MainView {
   private lateinit var user: User
   private val cart = Cart()  // Inicializa el carrito
   private val cartController = CartController(cart)
   private val productList = ProductList()  // Lista de productos

   fun init() {
      println("Bienvenido a la tienda")
      println("¿Cuál es tu nombre?")
      user = User(readln())
      mainMenu()
   }
```

CartController.kt

Cuando agregamos un artículo, el método addItem() es el que se encarga de ver que la cantidad no sea negativa y luego añade el producto al carro. Para eliminar un articulo, tenemos el método removeItem() que reduce la cantidad o en su defecto elimina el producto del carrito.

Al agregar un artículo, el método addItem() verifica que la cantidad no sea negativa y luego añade el producto al carrito. Para eliminar un artículo, el método removeItem() reduce la cantidad o elimina el producto del carrito si la cantidad es insuficiente.

```
class CartController (private val cart: Cart) {
    fun addItem(item: CartItem){
        if (item.quantity < 0) {
            throw NegativeQuantityException()
        }
        cart.items.add(item)
    }
    fun removeItem(productId: Int, quantityToRemove: Int) {
        val cartItem = cart.items.find { it.product.id == productId }
        if (cartItem != null) {
            if (cartItem.quantity > quantityToRemove) {
                cartItem.quantity -= quantityToRemove
            } else {
                cart.items.remove(cartItem)
            }
        } else {
            throw EmptyCartException()
        }
    }
```

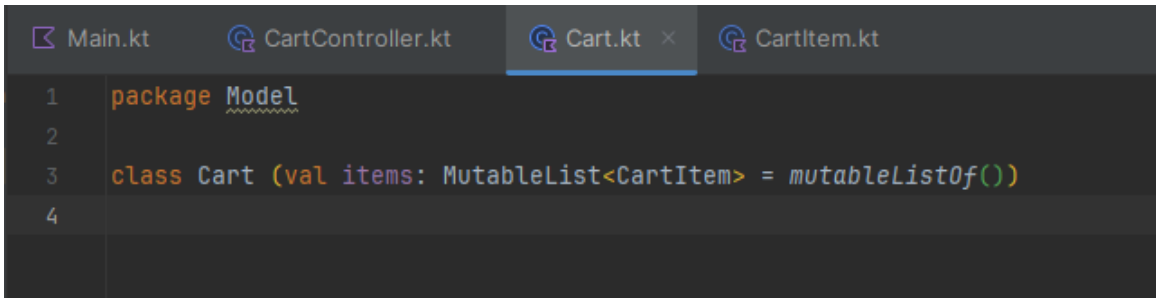


Cart.kt y CartItem.kt

Los archivos Cart.kt y CartItem.kt son los que definen las estructuras de los datos que se utilizan en el carrito. El archivo cart.kt es el que contiene la clase cart, que es básicamente una lista mutable de artículos (CartItem) añadidos por el usuario. Cada artículo está compuesto por un producto y una cantidad y esta estructura permite gestionar varias veces un mismo producto dentro del carrito

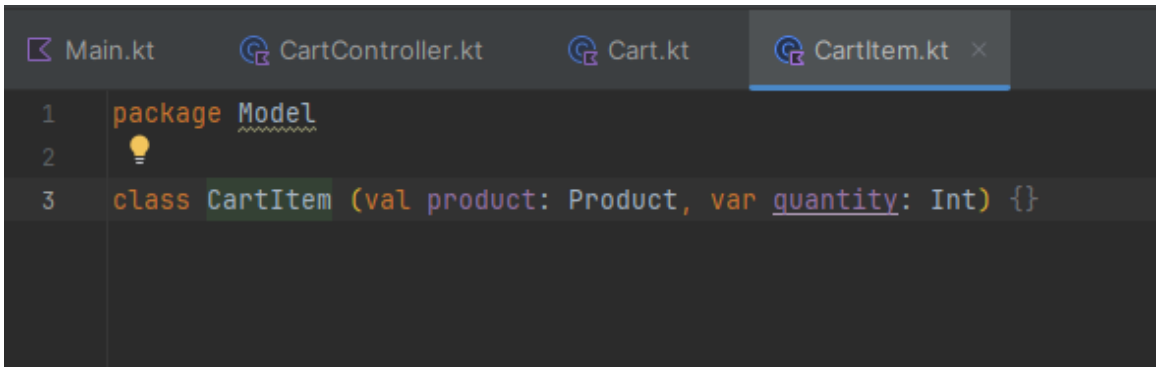
// Cart.kt

class Cart(val items: MutableList<CartItem> = mutableListOf())



// CartItem.kt

class CartItem(val product: Product, var quantity: Int)



Product.kt y ProductList.kt

El archivo Product.kt es en el que estamos definiendo la estructura de un producto, definimos su id, nombre y el precio. Esta clase la utilizamos para representar y mostrar los productos que se pueden agregar al carrito.

También tenemos el archivo ProductList.kt en el cual gestionamos una lista de productos que están disponibles y así el usuario puede buscarlos por su nombre o por su id.

El método getProductById() devuelve el producto basándose en su id, y el método showProducts() es el encargado de mostrar la lista completa de los productos y claro, su respectiva información.

```
class ProductList {
    val products: List<Product> = listOf(
        Product(1, "Laptop", 999.99),
        Product(2, "Smartphone", 699.99),
    )

    fun getProductById(productId: Int): Product? {
        return products.find { it.id == productId }
    }
    fun showProducts() {
        println("Productos disponibles:")
        products.forEach { println("${it.id}. ${it.name} - $${it.price}") }
    }
}
```

kotlinc.xml

Aquí básicamente podemos encontrar las configuraciones del compilador. Definimos que estamos trabajando con la versión 1.8 de la JVM y utilizamos la versión 2.0 para la API y para el lenguaje de kotlin

```
<?xml version="1.0" encoding="UTF-8"?>

<project version="4">

  <component name="Kotlin2JvmCompilerArguments">

    <option name="jvmTarget" value="1.8" />

  </component>

  <component name="KotlinCommonCompilerArguments">

    <option name="apiVersion" value="2.0" />

    <option name="languageVersion" value="2.0" />

  </component>

</project>
```

Main.kt CartController.kt Cart.kt CartItem.kt kotlinc.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project version="4">
  <component name="Kotlin2JvmCompilerArguments">
    <option name="jvmTarget" value="1.8" />
  </component>
  <component name="KotlinCommonCompilerArguments">
    <option name="apiVersion" value="2.0" />
    <option name="languageVersion" value="2.0" />
  </component>
</project>
```

misc.xml

Este archivo define el SDK que utiliza el proyecto. En este caso, se usa "corretto-22" como el JDK para compilar el proyecto Java, y se especifica la ubicación de los archivos de salida.

<?xml version="1.0" encoding="UTF-8"?>

<project version="4">

<component name="ProjectRootManager" version="2" project-jdk-name="corretto-22" project-jdk-type="JavaSDK">

<output url="file://$PROJECT_DIR$/out" />

</component>

</project>

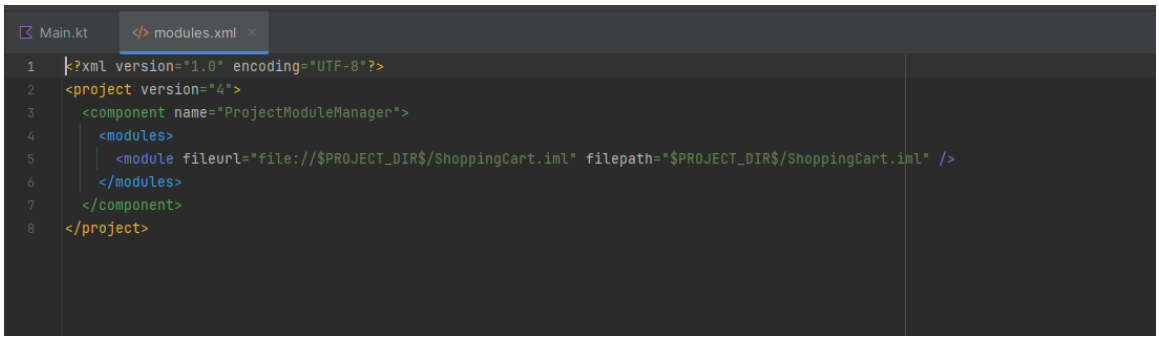
Main.kt kotlinc.xml misc.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project version="4">
  <component name="ProjectRootManager" version="2" languageLevel="JDK_21" project-jdk-name="17" project-jdk-type="JavaSDK">
    <output url="file://$PROJECT_DIR$/out" />
  </component>
</project>
```

modules.xml

Este archivo gestiona los módulos dentro del proyecto. Hace referencia al archivo de configuración del módulo principal, en este caso, ShoppingCart.iml.

```
<?xml version="1.0" encoding="UTF-8"?>

<project version="4">

  <component name="ProjectModuleManager">

    <modules>

      <module fileurl="file://$PROJECT_DIR$/ShoppingCart.iml"
filepath="$PROJECT_DIR$/ShoppingCart.iml" />

    </modules>

  </component>

</project>
```



vcs.xml

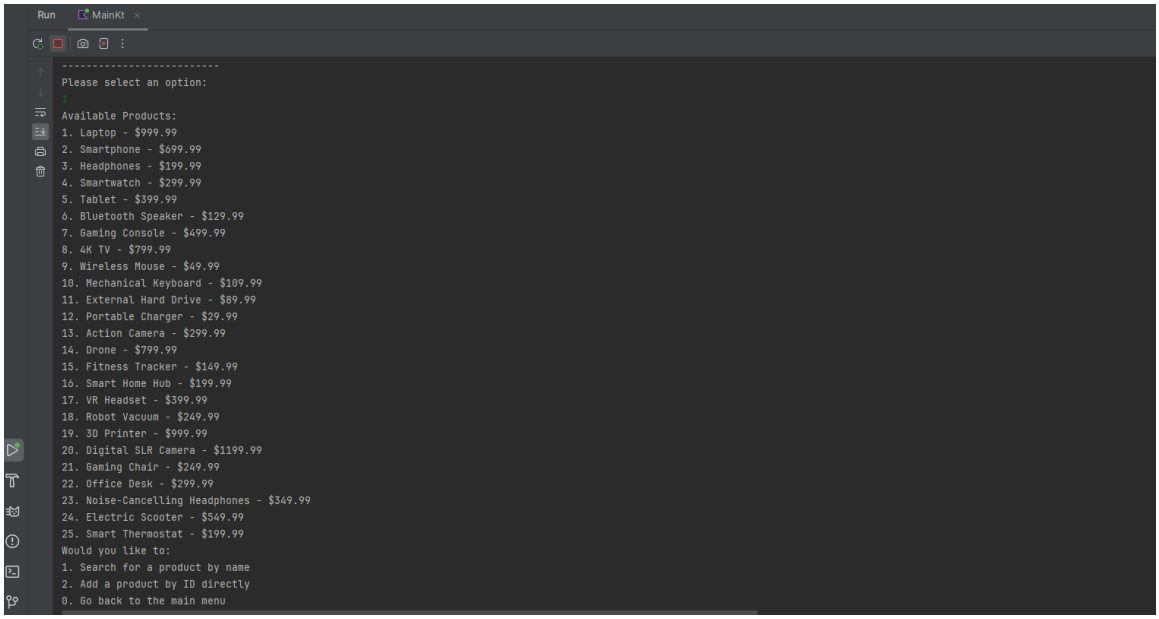
Este archivo está relacionado con el sistema de control de versiones que se utiliza en el proyecto. En este caso, el sistema de control de versiones es Git y está mapeado al directorio raíz del proyecto.

```
<?xml version="1.0" encoding="UTF-8"?>

<project version="4">

  <component name="VcsDirectoryMappings">

    <mapping directory="$PROJECT_DIR$" vcs="Git" />

  </component>

</project>
```

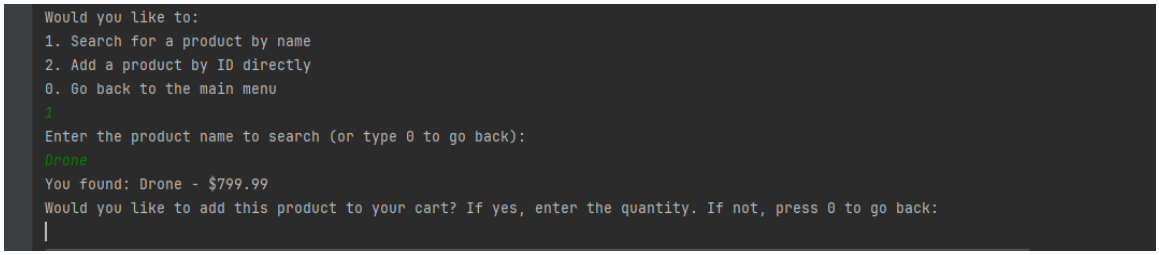
Funcionalidad

Menú desplegado:

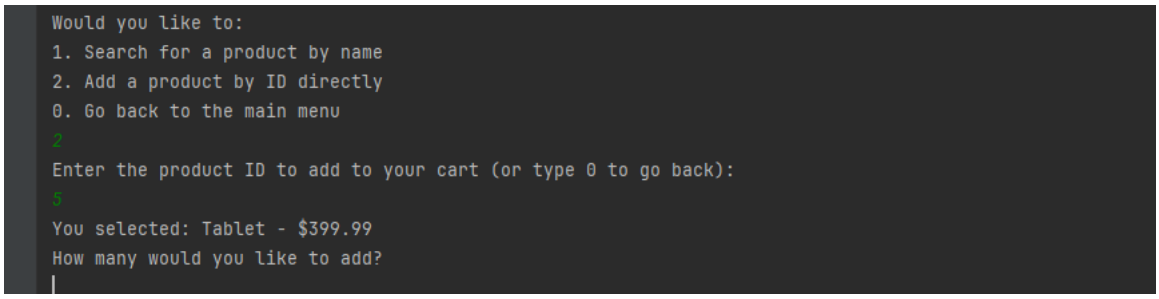
Opción 1 – Ver productos:



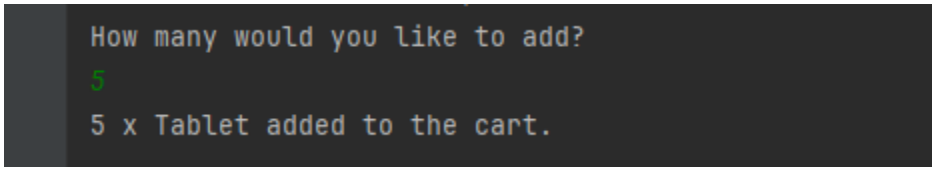
Opción 1.1 - Buscar producto por nombre:



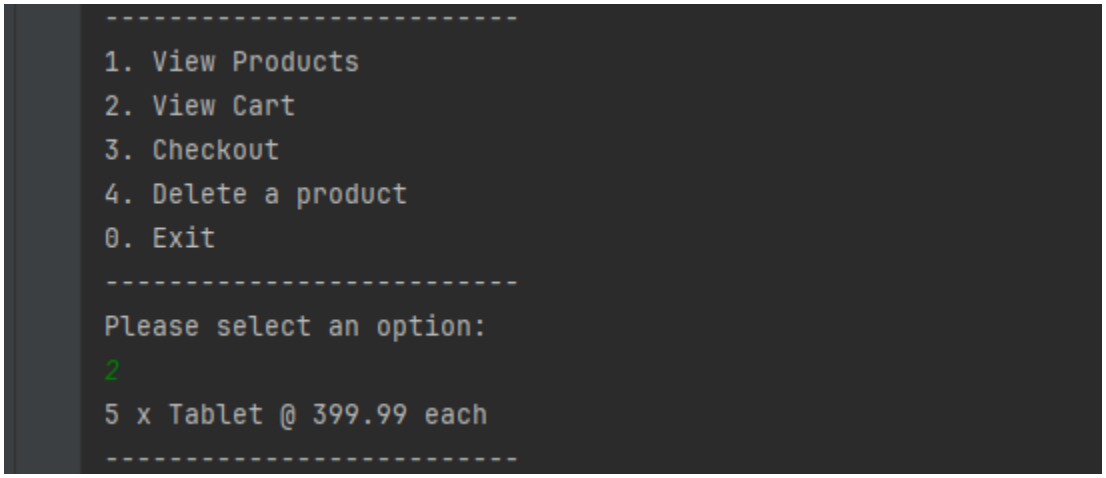
Opción 1.2 - Añadir producto por ID directamente:



Añadir cantidad de producto previamente seleccionado:



Opción 2 – Ver carrito de compras:



Opción 3 – Checkout del carrito:

```
--------------------------
1. View Products
2. View Cart
3. Checkout
4. Delete a product
0. Exit
--------------------------
Please select an option:
3
The total price is: $1999.95
Proceed to payment? (y/n)
```

Proceder al pago:

```
Please select an option:
3
The total price is: $1999.95
Proceed to payment? (y/n)
y
Payment successful. Thank you for shopping!
--------------------------
```

Opción 4 – Borrar producto de carrito:

Con el carrito:

```
--------------------------
1. View Products
2. View Cart
3. Checkout
4. Delete a product
0. Exit
--------------------------
Please select an option:
2
4 x Office Desk @ 299.99 each
3 x Bluetooth Speaker @ 129.99 each
6 x Tablet @ 399.99 each
--------------------------
```

Seleccionando producto con ID 5 y removiendo cantidad de 5:

```
Select a product by ID to remove from your cart (or type 0 to go back):
5
How many would you like to remove?
5
5x Tablet removed from your cart.
```

Carrito después del borrar producto:

```
--------------------------
Please select an option:
2
4 x Office Desk @ 299.99 each
3 x Bluetooth Speaker @ 129.99 each
1 x Tablet @ 399.99 each
--------------------------
```

Opción 0 – Salir:

```
--------------------------
1. View Products
2. View Cart
3. Checkout
4. Delete a product
0. Exit
--------------------------
Please select an option:
0
Thank you for visiting DSM's Shop. Goodbye!
--------------------------
```

Enlace a Repositorio

https://github.com/montanoo/shopping-cart.git