MANUAL DE CARRITO DE COMPRAS-DSM

Néstor Javier Artiaga Larin	**AL150682**
Enma Sofía López Rojas	**LR230079**
Kevin Enrique Martínez Martínez	**MM230084**
José Gerardo Marroquín Vásquez	**MV230090**
Alba Vanessa Urrutia Cruz	**UC151875**

LINK DE VIDEO:

https://github.com/KevinEnriqueMartinezMartinez/CarritoComprasUDB.git

Contenido

INTTRODUCCION

Se ha desarrollado una Proyecto de un carrito de compras, con esta aplicación se llevará un manejo de las compras de una tienda inicialmente se ha desarrollado para poder observar los detalles de este en la terminal del ide elegido.

Previo al desarrollo del proyecto se hizo un estudio los fundamentos del lenguaje a utilizar de las necesidades a solventar con la aplicación a crear, por lo que en este manual se basará en las configuraciones previas al inicio del desarrollo, así como la estructura actual del proyecto.

DESARROLLO

HERRAMIENTAS BASICAS

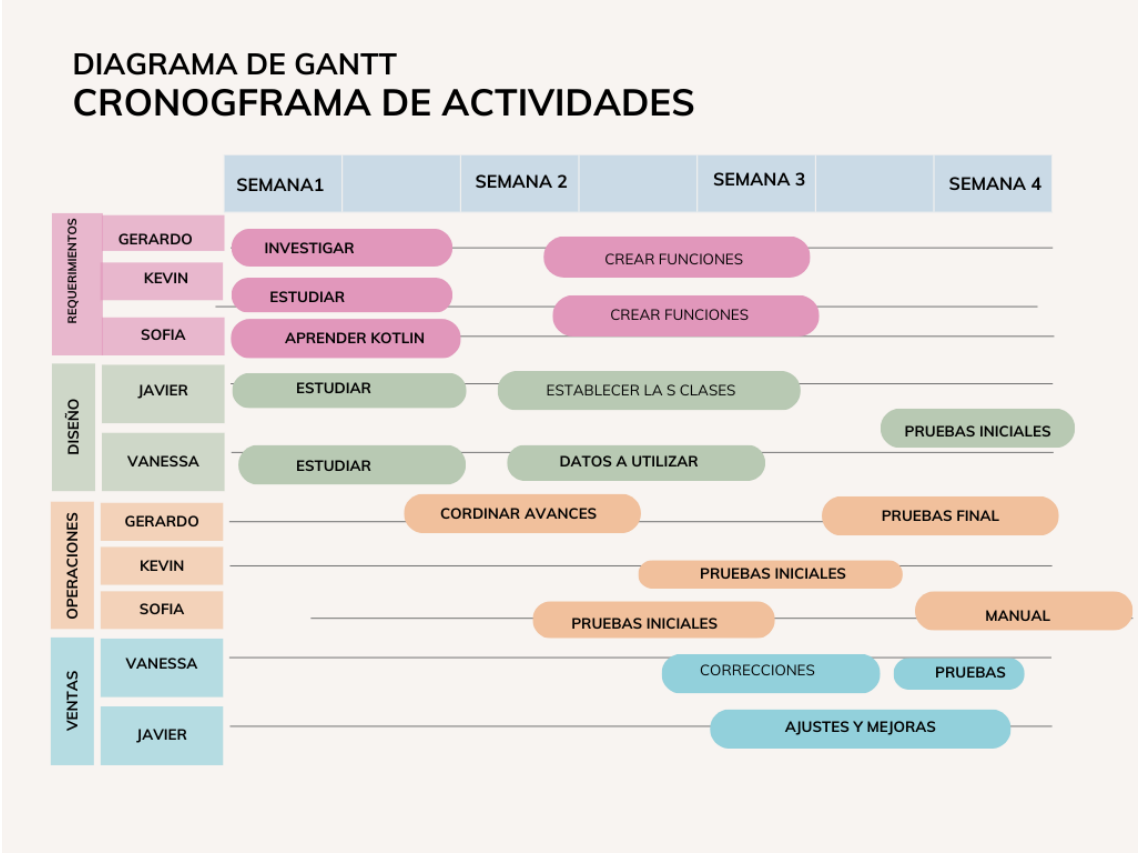
Para iniciar con el desarrollo del carrito de compras es necesario conocer las herramientas básicas para la realización del proyecto; por lo que detallaremos que herramientas son estas: como el lenguaje de programación se ha utilizado Kotlin y además se ha utilizado el manejo de colecciones como lo son el uso de listas para detallar los productos dentro del carrito lo que brinda una interfaz más amigable y detallada con el usuario, y como IDE de desarrollo se ha utilizado Intellij idea versión 2024 “x64”, una vez tengamos presente las herramientas básicas para la creación de una app procederemos a ir detallando algunas de las partes más importantes del proyecto desarrollado.

CONFIGURACIONES PREVIAS

Una vez se tenga claro el IDE a utilizar y el lenguaje mediante el cual se desarrollará es necesario configurar el entorno de desarrollo, para el desarrollo de este se ha utilizado la JDK 17.0.10 , la cual se ha configurado como variable de entorno en la máquina que se han creado el proyecto .

GRANTTPRO

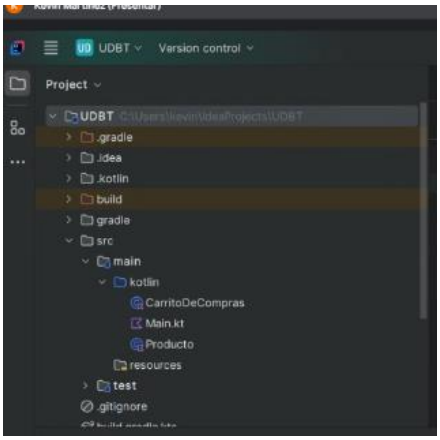
Para la realización de la app se organizó mediante un gestor de proyecto en el cual nos fuimos asignando las actividades a realizar como se muestra en la siguiente imagen



ESTRUCTURA DEL PROYECTO

CARPETAS

Para el desarrollo del proyecto se ha utilizado el ide de Intellij de JetBrains, una vez se hayan creado el proyecto se han creado la clase que se detallan dentro de la carpeta "src" dentro del proyecto estas clases llevan los nombres siguientes: Main, Producto, CarritoDeCompra.



A continuación, se detallará un poco más de las clases utilizadas

CLASE MAIN

Aquí se define una lista de productos disponibles usando la función listOf(). Cada producto es una instancia de la clase Producto, que tiene un **id**, **nombre**, **precio**, y **cantidad disponible**.

Se crea un objeto de la clase CarritoDeCompras, que será utilizado para almacenar los productos seleccionados por el usuario.

Esta variable almacenará la opción seleccionada por el usuario en el menú.

El bucle do-while garantiza que el menú se muestre repetidamente hasta que el usuario elija la opción de salida (opcion = 0).

```
fun main() {
    val productosDisponibles = listOf(
        Producto(1, "Laptop", 850.0, 10),
        Producto(2, "Mouse inalámbrico", 30.0, 25),
        Producto(3, "Teclado mecánico", 75.0, 15),
        Producto(4, "Monitor 24 pulgadas", 200.0, 8),
        Producto(5, "Disco Duro SSD 500GB", 120.0, 12),
        Producto(6, "Audífonos Bluetooth", 55.0, 20)
    )

    val carrito = CarritoDeCompras()




    var opcion: Int



    do {
        println("\n--- UDB TECHNOLOGY S.A de C.V ---")
        println("1. Ver productos")
        println("2. Agregar producto al carrito")
        println("3. Ver carrito")
        println("4. Eliminar producto del carrito")
```

Si el usuario selecciona la opción 1, se muestra la lista de productos disponibles, iterando sobre la lista con forEach.

Aquí se pide al usuario el **ID** del producto que quiere agregar al carrito. Se busca en la lista de productos mediante find(). Si el producto existe, se solicita la cantidad y se agrega al carrito usando la función agregarProducto de la clase CarritoDeCompras. Si no se encuentra el producto, se muestra un mensaje de error.

Llama a la función mostrarCarrito() de la clase CarritoDeCompras para mostrar los productos agregados al carrito.

Si el carrito no está vacío, se le pide al usuario que ingrese el índice del producto que desea eliminar. Luego se llama a la función eliminarProducto() de la clase CarritoDeCompras.

```
      println("5. Finalizar compra")
      println("0. Salir")
      print("Elija una opción: ")

 opcion = readLine()?.toIntOrNull() ?: 0

    when (opcion) {
      1 -> {
        println("\nProductos disponibles:")
        productosDisponibles.forEach { producto ->
           println("${producto.id}. ${producto.nombre}          -
$${producto.precio} (Cantidad disponible: ${producto.cantidadDisponible})")
          }
        }
       2 -> {
        print("Ingrese el ID del producto: ")
        val idProducto = readLine()?.toIntOrNull() ?: 0
        val productoSeleccionado =  productosDisponibles.find { it.id == idProducto
}

        if (productoSeleccionado != null) {
        print("Ingrese la cantidad: ")
        val cantidad = readLine()?.toIntOrNull() ?: 1
        carrito.agregarProducto(productoSeleccionado, cantidad)
        } else {
          println("Producto no encontrado.")
        }
       }
       3 -> carrito.mostrarCarrito()
       4 -> {
         carrito.mostrarCarrito()
         if (carrito.items.isNotEmpty()) {
         print("Ingrese el número del producto a eliminar: ")
          val indiceProducto = readLine()?.toIntOrNull() ?: 0
            carrito.eliminarProducto(indiceProducto)



}
        }
        5 -> carrito.finalizarCompra()
        0 -> println("Gracias por visitar UDB Technology.")
        else -> println("Opción no válida. Intente de nuevo.")
      }
    } while (opcion != 0)
}
```

CLASE PRODUCTO

data class: En Kotlin, una data class es una clase especial que se usa principalmente para contener datos. Proporciona automáticamente varias funciones útiles como toString(), equals(), hashCode(), y copy() sin necesidad de implementarlas manualmente. Es ideal para representar objetos con valores inmutables o ligeramente mutables.

La clase Producto tiene cuatro propiedades, que representan los atributos de un producto: id: Int, nombre: String, precio: Double, cantidadDisponible: Int

```
// Producto.kt
data class Producto(val id: Int, val nombre: String, val precio:
Double, var cantidadDisponible: Int)
```

CLASE CARRITO DE COMPRAS

items: Es una lista mutable (mutableListOf) que almacena pares (Pair<Producto, Int>). Cada par representa un producto y la cantidad que ha sido agregada al carrito. La estructura de los pares es: el producto y la cantidad de ese producto en el carrito.

Parámetros: Recibe un objeto Producto y una cantidad que se desea agregar.

Verificación de stock: Comprueba si la cantidad disponible del producto es suficiente.

Agregar al carrito: Si hay suficiente stock, se añade un par Producto-cantidad a la lista items y se reduce el stock del producto en el inventario (producto.cantidadDisponible -= cantidad).

Mensajes: Se imprime un mensaje indicando si el producto fue agregado o si no hay suficiente inventario

Carrito vacío: Si la lista items está vacía, se imprime un mensaje diciendo que el carrito está vacío.

Mostrar productos: Si hay productos en el carrito, se imprimen los detalles de cada producto usando un bucle forEachIndexed que muestra el índice del producto, su nombre, cantidad, precio unitario, y el total por ese producto.

Total del carrito: Al final, se calcula el total del carrito usando la función calcularTotal() y se imprime.

Calcular total: Esta función calcula y devuelve el costo total del carrito sumando el producto del precio y la cantidad de cada artículo en la lista items.

Utiliza sumByDouble para iterar sobre los pares de producto y cantidad, multiplicando el precio del producto por la cantidad.

```
class CarritoDeCompras {
    val items = mutableListOf<Pair<Producto, Int>>()

    fun agregarProducto(producto: Producto, cantidad: Int) {
        if (producto.cantidadDisponible >= cantidad) {
            items.add(Pair(producto, cantidad))
            producto.cantidadDisponible -= cantidad  // Reducir la cantidad disponible
            println("Se ha agregado $cantidad de '${producto.nombre}' al carrito.")
        } else {
            println("No hay Sufieciente Productos disponibles. Solo hay ${producto.cantidadDisponible} unidades en el inventario.")
        }
    }

    fun mostrarCarrito() {
        if (items.isEmpty()) {
            println("El carrito está vacío.")
        } else {
            println("Productos en el carrito:")
            items.forEachIndexed { index, (producto, cantidad) ->
                println("${index + 1}. ${producto.nombre} (x$cantidad) - Precio unitario: $${producto.precio} - Total: $${producto.precio * cantidad}")
            }

            // Mostrar el total al final
            val total = calcularTotal()
            println("**********************************")
            println("Total del carrito: $$total")
            println("**********************************")
        }
    }
```

Total de la compra: Calcula el total de la compra con calcularTotal().

```
fun calcularTotal(): Double {
    return items.sumByDouble { (producto, cantidad) ->
producto.precio * cantidad }
    println("hello word")
}
```

Finalización de compra: Si el carrito tiene productos (es decir, el total es mayor que 0), se imprime un resumen de la compra con los detalles de cada producto (nombre, cantidad, precio, y total por producto) y el total final de la compra.

Vaciar carrito: Una vez finalizada la compra, la lista items se limpia con items.clear() para que quede vacía.

Si el carrito está vacío, se muestra un mensaje indicando que no se puede finalizar la compra.

```
fun finalizarCompra() {
    val total = calcularTotal()
    if (total > 0) {
        println("*** UDB TECHNOLOGY S.A de C.V. ***")
        println("Ubicación: Frente a Plaza Salvador del Mundo")
        println("Contacto: 2413-7985 | udbtecnology@gmail.com")
        println("================================")
        println("Detalles de la compra:")

        items.forEachIndexed { index, (producto, cantidad) ->
            println("${index + 1}. ${producto.nombre} (x$cantidad) -
Precio unitario: $${producto.precio} - Total: $${producto.precio *
cantidad}")

        }

        println("================================")
        println("Total de la venta: $$total")
        println("================================")
        println("Gracias por su compra.")
        println("*** UDB TECHNOLOGY S.A de C.V***")

        items.clear()  // Limpia el carrito tras la compra
    } else {
        println("El carrito está vacío, no se puede finalizar la
compra.")
    }
}
```

Eliminar producto: Recibe un índice que indica qué producto se debe eliminar del carrito. El índice es 1-basado (es decir, el primer producto está en la posición 1, no en la 0).

```
fun eliminarProducto(indice: Int) {
    if (indice in 1..items.size) {
        val productoEliminado = items.removeAt(indice - 1)
```

Verificación del índice: Comprueba si el índice está dentro del rango válido (entre 1 y el tamaño de la lista items). Si es válido, elimina el producto en la posición correspondiente con removeAt().

Actualizar inventario: Cuando un producto es eliminado, su cantidad en el inventario se restaura, sumando la cantidad eliminada de vuelta al stock.

Manejo de errores: Si el índice es inválido, se muestra un mensaje de error.

```
            println("Se ha eliminado
'${productoEliminado.first.nombre}' del carrito.")
            productoEliminado.first.cantidadDisponible +=
productoEliminado.second  // Devolver stock al inventario
        } else {
            println("Índice inválido. Por favor, seleccione un número
correcto.")
        }
    }
}
```