Proyecto 02 - ODIN Punto de Venta

Integrantes

- 321671958 - *Cisneros Álvarez Danjiro*.
- 422083399 - *Teniente Ornelas Oscar Manuel*.
- 422109167 - *Tenorio Reyes Ihebel Luro*.

Descripción

Nota: Los elementos marcados como (no prioritarios) son porque es posible que no se implementen al final pero en nuestras ambiciones están considerados.

`ODIN` es principalmente el software donde se realiza la transacción comercial entre el cliente y la tienda, no obstante, puede cumplir funciones en la gestión de la misma.
Entre sus funciones se encuentran:

- Calcular el importe final.
  - Debe incluir los productos que se están añadiendo a la operación y su cantidad.
  - Deben poderse eliminar de la operación rápidamente.
  - Se debe preguntar la forma de pago del producto (en caso de ser con efectivo debe mostrar cuanto se debe entregar de cambio)
  - Si un producto requiere ser mayor de edad solo debe mandar un aviso al operador del `ODIN` de que no debe venderse a menores.
- Debe mostrar un ticket final que debe incluir:
  - El nombre de la tienda.
  - Los productos, su cantidad y su precio.
  - Monto total.
  - Si fue con efectivo, el monto recibido y cuanto se entregó de cambio
- Debe ser capaz de gestionar una base de datos de productos, esto es:
  - El operador puede agregar nuevos productos al catalogo.
  - El operador puede eliminar manualmente productos del catálogo (por mermas y así).
  - Tras cada transacción, se debe actualizar la base de datos.
  - Cada transacción también debe guardarse en un historial para la tienda.
  - Generar resúmenes.

En cuanto al **Frontend** nos centramos en una interfaz simple dividida en páginas navegables a través de botones grandes con letras legibles.

Debe ser rápido el acceso la sección de cobro en todo momento pues si el operador está haciendo otra cosa y llega un cliente debe poder atenderlo.

Manual

Compilación y Ejecución:

Maven (probado en Linux)

Compilar

`mvn clean compile`

Ejecutar

`mvn javafx:run`

Docker

Nota: Puede requerir `sudo`

Levantar imagen

`sudo docker build --no-cache -t odin-pos .`

Ejecutar

```
sudo docker run -it --rm
    -e DISPLAY=$DISPLAY
    -v /tmp/.X11-unix:/tmp/.X11-unix \
    odin-pos
```

Patrones de diseño

Patrón de Diseño	Propósito General	Uso Específico en tu Proyecto (PdV)
1. MVC (Modelo-Vista-Controlador)	Separa la aplicación en tres componentes para aislar la lógica de negocio de la interfaz de usuario.	Es la **arquitectura principal de tu aplicación JavaFX**. • **Vista (V):** Los archivos `*.fxml` que definen la interfaz gráfica. • **Controlador (C):** Las clases `*Controller.java` que responden a los clics del usuario. • **Modelo (M):** La `PuntoDeVentaFacade`

Patrón de Diseño	Propósito General	Uso Específico en tu Proyecto (PdV)
		(que es la puerta a toda tu lógica de backend) y los DTOs que se mueven entre el backend y el frontend.
2. Facade (Fachada)	Proporciona una interfaz unificada y simple a un conjunto de subsistemas complejos.	`PuntoDeVentaFacade` será la **única clase** con la que los `Controladores` (C) hablarán. Ocultará toda la lógica de servicios, builders y la BD en métodos simples.
3. Strategy (Estrategia)	Define una familia de algoritmos, los encapsula y los hace intercambiables.	Manejar las **formas de pago**. Habrá una `PagoEfectivoStrategy` (que sabe calcular el cambio) y una `PagoTarjetaStrategy` (que solo valida el monto). La venta usa la estrategia correcta sin saber sus detalles.
4. Observer (Observador)	Permite que un objeto (Sujeto) notifique a otros (Observadores) cuando su estado cambia, sin conocerlos directamente.	Para **acciones post-venta**. Cuando la venta se completa (`Sujeto`), notificará al `ServicioDeInventario` y al `ServicioDeHistorial` (`Observadores`) para que actualicen la base de datos (descontar stock y guardar la transacción).
5. Builder (Constructor) *(para la Venta)*	Separa la construcción de un objeto complejo de su representación final, permitiendo crearlo paso a paso.	Se usará un `VentaBuilder` para **gestionar la "cuenta" temporal** en memoria. Este objeto tendrá métodos como `agregarItem(...)` y `eliminarItem(...)`. Al final, su método `.build()` creará el objeto `Venta` final e inmutable.
6. Builder (Constructor) *(para el Ticket)*	(Mismo propósito, pero aplicado a una representación diferente).	Se usará un `TicketBuilder` para **generar el *string* del recibo**. Tomará el objeto `Venta` (ya finalizado) y construirá, paso a paso, el formato de texto (cabecera, items, totales) listo para imprimir.
7. Strategy (Estrategia)	Mismo propósito	Nos permite generar un resumen solo cambiando el algoritmo según sea por categoría, por producto o por tiempo, todo esto de forma intercambiable.

Nota, el código esta dividido por carpetas para que cada clase esté según el patrón al que pertenece.