TECNICATURA UNIVERSITARIA
EN PROGRAMACIÓN
A DISTANCIA

Trabajo Final Integrador (TFI)

Programación 2

Fecha de entrega: 20/11/2025
Dominio del Proyecto: Pedido → Envío
Grupo: 168

Integrantes y Roles

- **Franco Luna - francomoon99@gmail.com - comisión: 17**
  Se encargó del paquete de Entidades, clase de dominio (A y B) con id y eliminado (baja lógica), de crear y modificar el UML y del ordenamiento del informe.

- **Fernando Aguillón - fernando.j.aguillon.b@gmail.com - comisión: 18**
  Se encargó de crear el paquete de Service, reglas del negocio, validaciones, transacciones y de la confección del Video.

- **Gabriel Ledesma - gledesma10@gmail.com - comisión: 16**
  Se encargó de crear los paquetes de Config, con la lectura de propiedades externas; de la creación de la base de datos y del Main, con el arranque de app y el AppMenu.

- **Manuel Williams - manu_williams@hotmail.com - comisión: 17**
  Se encargó de crear el paquete de Dao con sus interfaces genéricas, JDBC y PreparedStatement

Elección del dominio y justificación

Nuestra elección fue la de elegir el dominio "**Pedido → Envío**" para la realización del trabajo práctico integrador, esto fue debido a que varias aplicaciones ligadas al comercio electrónico emplean un sistema de transporte de mercancías para la notificación del estado del pedido y la recepción de este, razón por la cual nos vimos inclinados a elegir este dominio.

Diseño: decisiones clave (1→1, FK única vs PK compartida) + UML.

El modelo del sistema se basa en dos entidades principales: **Pedido** y **Envío**. Entre ambas se definió una **relación 1→1 unidireccional**, donde **un Pedido posee exactamente un Envío asociado**, mientras que un Envío no mantiene referencias de vuelta hacia Pedido.

Esta decisión se tomó para mantener un diseño simple y orientado a la navegación desde la capa de negocio: el Pedido es el elemento central del flujo y el Envío funciona como un **detalle complementario**.

Tipo de relación 1→1

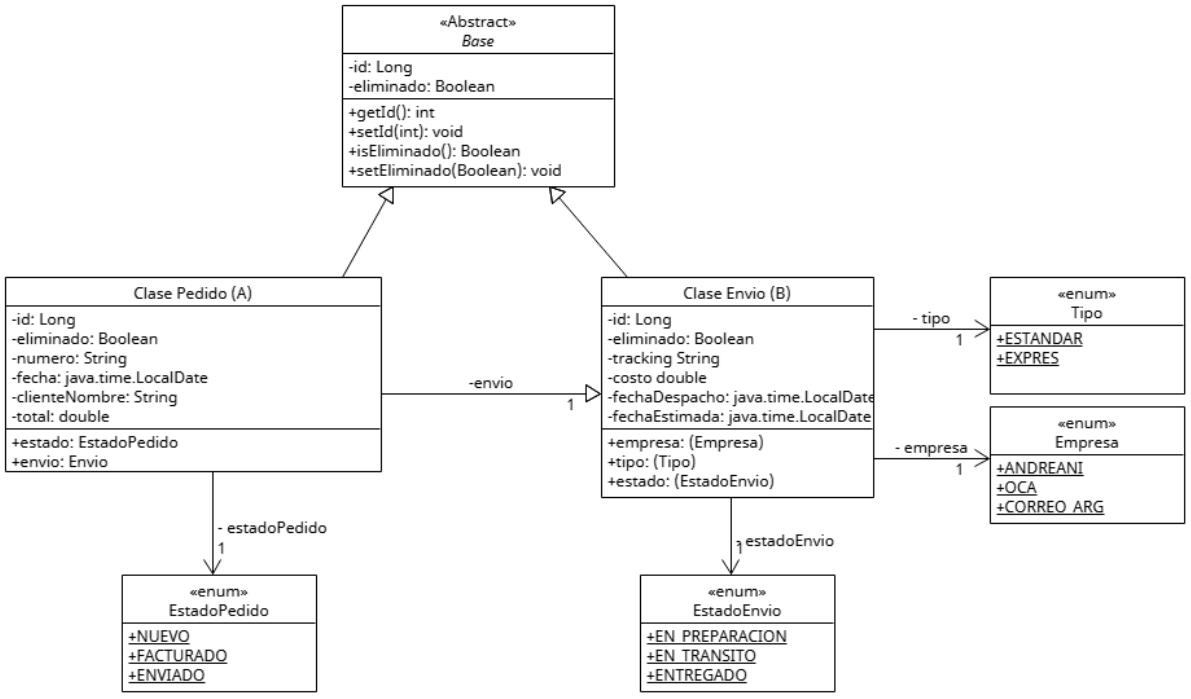
La relación entre **Pedido (A)** y **Envio (B)** es **unidireccional** por lo cual solo la clase **Pedido** contiene el atributo **envio**. La clase **Envio** no tiene un atributo que tenga una referencia hacia Pedido.
Dado el tipo de programa, la Cardinalidad es real debido a que cada Pedido tiene un único Envío, y cada Envío pertenece a exactamente un Pedido.

Implementación: FK única vs PK compartida

En este caso el Envío tiene su propio id independiente, y el Pedido referencia a ese Envío mediante un campo envio_id, por lo cual hay una independencia entre las entidades permitiendo crear primero un pedido para luego crear el envio.

UML

En el UML se pueden apreciar las 2 entidades, sus atributos y su relación unidireccional con su cardinalidad y sus enums correctamente separados.



Arquitectura por capas

El programa fue dividido en 5 capas,las cuales son: **Entities, Service, Dao, Config y Main**. La descripción de estas capas son:

- **Entities**: Este paquete conforma las Entidades del programa, con sus atributos, sus modificadores de acceso, métodos y relaciones. Este está compuesto de 3 clases, las cuales son:
    - Base: Es una clase abstracta que contiene los atributos de Id y eliminado, que se repiten en las 2 clases principales de “Pedidos” y “Envios”.
    - Envio: Clase que contiene los atributos que comprende al envío de un producto.
        - Atributos:
            - private String tracking
            - private Empresa empresa
            - private Tipo tipo
            - private double costo
            - private LocalDate fechaDespacho
            - private LocalDate fechaEstimada
            - private Estado estado
        - Enum:
            - public enum Empresa { ANDREANI, OCA, CORREO_ARG }
              // Contiene las constantes de empresas utilizadas en los atributos de Envios.
            - public enum Tipo { ESTANDAR, EXPRES }
              // Contiene las constantes de tipos utilizados en los atributos de Envios.
            - public enum Estado { EN_PREPARACION, EN_TRANSITO, ENTREGADO }
              // Contiene las constantes de estados utilizados en los atributos de Envios.

    - Pedidos:Clase que contiene los atributos que comprenden al pedido de un producto.
        - Atributos:
            - private String numero
            - private LocalDate fecha
            - private String clienteNombre
            - private double total
            - private Estado estado

        - private Envio envio
    - Enum:
        - public enum Estado { NUEVO, FACTURADO, ENVIADO } //Contiene las constantes de estados utilizados en los atributos de Pedidos.

- **Service**: La Capa Service actúa como intermediaria entre el menú (interacción con el usuario) y la Capa DAO (persistencia). Su función principal es:
  -Validar datos antes de acceder a la base.
  -Garantizar reglas de negocio.
  -Orquestar operaciones complejas que involucran múltiples DAOs.
  -Controlar las transacciones JDBC con commit y rollback.

  EnvioService: Es un servicio simple asociado a la entidad B, encargado de ejecutar las operaciones CRUD sobre Envio de manera aislada.
  -Cada operación abre y cierra su propia conexión.
  -Realiza validaciones básicas (empresa, tipo, longitud del tracking).
  -No aplica transacciones compuestas porque la creación del Envío forma parte de la transacción ejecutada por el PedidoService.

  PedidoService: Es el servicio principal del ítem 5. Su responsabilidad es ejecutar operaciones atómicas que involucran **Pedido (A)** y **Envio (B)** dentro de una misma transacción.
  insertar(Pedido), Crea primero el Envío, luego el Pedido. Ambas operaciones se ejecutan dentro de una misma transacción.
  Si alguna falla, se ejecuta rollback() para mantener la consistencia.
  actualizar(Pedido), Actualiza primero el Envío y después el Pedido, garantizando consistencia en cambios vinculados.
  eliminar(long id), Realiza la baja lógica del Pedido y su Envío asociado dentro de la misma transacción.

- **Dao**: La Capa DAO es la responsable de gestionar el acceso a la base de datos y actuar como puente entre el modelo de objetos y la persistencia en SQL.
  Su función principal es:

  - Ejecutar operaciones CRUD (crear, leer, actualizar, eliminar) para cada entidad.

  - Convertir objetos Java a registros SQL y viceversa.

  - Manejar conexiones, sentencias preparadas y ResultSet.

- Garantizar que la lógica de acceso a datos esté completamente separada de la lógica de negocio.

- Proveer métodos con conexión propia y métodos que aceptan una conexión externa para operaciones transaccionales coordinadas desde la capa Service.

GenericDao: Es una interfaz genérica que define la estructura que deben seguir todos los DAO del sistema.
Incluye:

- Métodos estándar CRUD (crear, leer, leerTodos, actualizar, eliminar).
- Sobrecargas de esos métodos que aceptan una Connection externa para permitir que la capa Service controle transacciones complejas.

EnvioDao: DAO dedicado exclusivamente a la entidad *Envio*.
Sus responsabilidades son:

- Ejecución de las operaciones CRUD sobre la tabla envio.
- Manejo de soft delete por medio del campo eliminado.
- Mapear cada fila del ResultSet a un objeto Envio utilizando un método privado mapEnvio.
- Ejecutar cada operación con su propia conexión cuando se lo invoca directamente desde la capa Service.
- Permitir operaciones dentro de una transacción cuando recibe una conexión externa desde PedidoService.

Este diseño permite que los Envíos se gestionen de manera independiente, pero también puedan integrarse a operaciones compuestas vinculadas a los Pedidos.

PedidoDao: Es el DAO principal del sistema, ya que administra la entidad *Pedido* y su relación 1 → 1 con *Envio*.
Sus responsabilidades incluyen:

- Insertar el Pedido y su Envío asociado dentro de la misma conexión.
- Leer pedidos incluyendo los datos del Envío mediante JOIN.
- Actualizar ambos objetos de forma coordinada para mantener consistencia entre Pedido y Envío.
- Realizar soft delete del Pedido y del Envío asociado dentro de una única operación.
- Proveer métodos con conexión externa para permitir transacciones controladas desde PedidoService.

Esta clase encapsula toda la lógica de persistencia de la operación A→B, asegurando integridad y atomicidad.

- **Config**: El paquete **config** tiene como responsabilidad central administrar la conexión a la base de datos y garantizar que toda la aplicación trabaje sobre una misma instancia de conexión, evitando errores y duplicación de recursos.

  Su pieza principal es la clase **Conexion**, que implementa un patrón Singleton. Esto significa que solo se crea una única conexión a la base de datos durante toda la ejecución del programa. Esta conexión se abre cuando la aplicación la necesita y se mantiene disponible para las diferentes capas: DAO, servicios y cualquier otra parte que requiera acceder a la base de datos.
  La clase se encarga de:

  -Cargar los datos de configuración como URL, usuario y contraseña.
  -Registrar el driver JDBC necesario.
  -Crear y devolver la conexión activa.
  -Manejar errores de conexión y exponer mensajes claros en caso de fallos.

  Gracias a este paquete, el resto de la aplicación no necesita preocuparse por cómo conectarse a la base de datos: simplemente solicita la conexión y la usa. Esto mejora la organización del código, reduce la duplicación y asegura un funcionamiento estable y consistente.

- **Main**:

  AppMenu: Esta funciona como el punto central donde se configura y se inicia el manejo del menú principal de la aplicación. Su tarea principal es inicializar los servicios necesarios (por ejemplo, PedidoService y EnvioService) y pasárselos a la clase que realmente gestiona la lógica del menú (MenuHandler).
  Preparando así el entorno para que el usuario pueda interactuar con el sistema y delega la ejecución del menú en el componente encargado de mostrar opciones y procesar acciones. Por ende esta clase mantiene la estructura del programa ordenada y separa la inicialización del flujo interactivo de la aplicación.

  Main: Esta clase es la puerta de entrada del programa. Contiene el método main, que se ejecuta automáticamente cuando se inicia la aplicación.
  El objetivo de esta es crear una instancia de AppMenu y llamar al método que pone en marcha el sistema. Es debido a esto que el programa queda correctamente encapsulado, ya que el método main no contiene lógica del negocio ni lógica de presentación, solamente da inicio a la aplicación.
  Esto ayuda a mantener el código limpio y facilita el mantenimiento y la ejecución desde cualquier entorno.

MenuHandler: Esta clase es la encargada de manejar toda la interacción con el usuario. Se ocupa de mostrar las opciones disponibles, leer las elecciones del usuario y ejecutar las acciones correspondientes usando los servicios del sistema.
Dentro de esta clase se concentra la lógica de flujo del menú mediante métodos que permiten hacer las siguientes acciones:

- crear pedidos
- crear envíos
- listar, buscar
- actualizar o eliminar registros

MenuHandler actúa como puente entre el usuario y los servicios, asegurando que las operaciones se ejecuten correctamente y que el programa responda de forma clara. Al centralizar esta lógica, MenuHandler evita duplicación de código y mantiene la interfaz del programa simple, ordenada y fácil de navegar.

Persistencia: Estructura de la base, orden de operaciones y transacciones

En la capa service se establecen los métodos utilizados para la generación de las transacciones y de las excepciones que provocan el rollback de los cambios de las transacciones.

Validaciones y reglas de negocio

La Capa Service aplica validaciones antes de invocar a la Capa DAO

Para Pedido:
- Número obligatorio y único.
- Cliente obligatorio.
- Fecha válida.
- Total positivo.
- Estado obligatorio.
- Envío obligatorio.

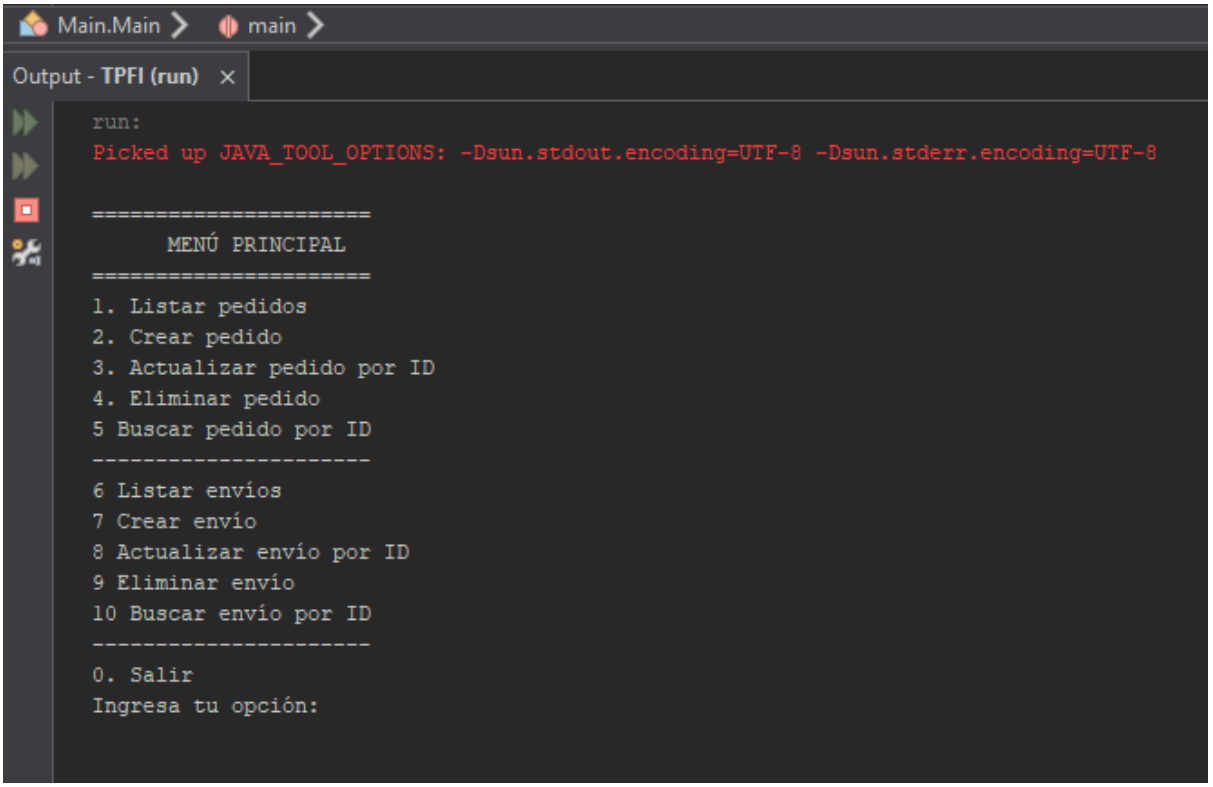
Para Envío:
- Empresa y tipo obligatorios.
- Tracking con longitud máxima.
- Validación de unicidad delegada a la BD.

Además, se garantiza la regla:
“Un Pedido sólo puede tener un Envío y un Envío solo puede pertenecer a un Pedido”

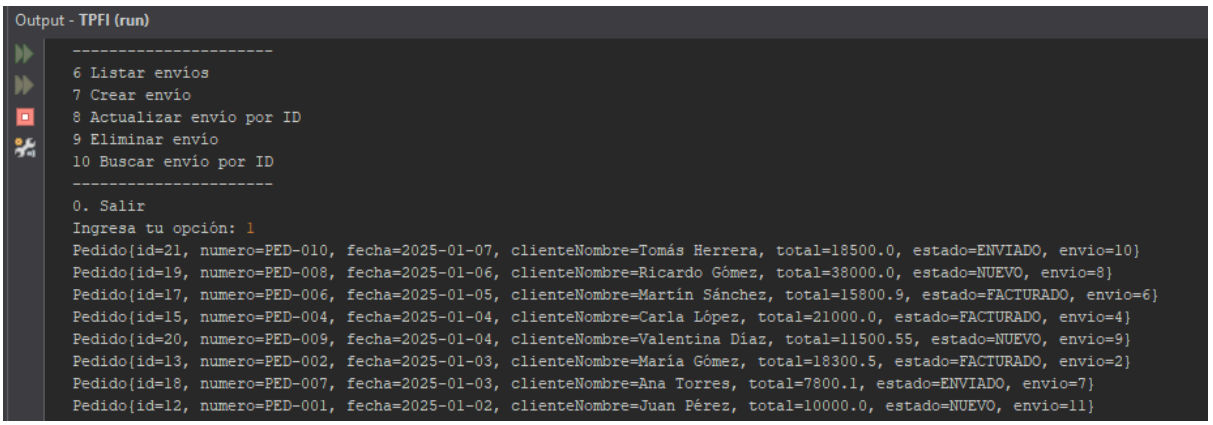
Esto se refuerza mediante: validaciones, la lógica del servicio, restricciones únicas en la base de datos.

Pruebas realizadas

Captura de menú principal del programa:

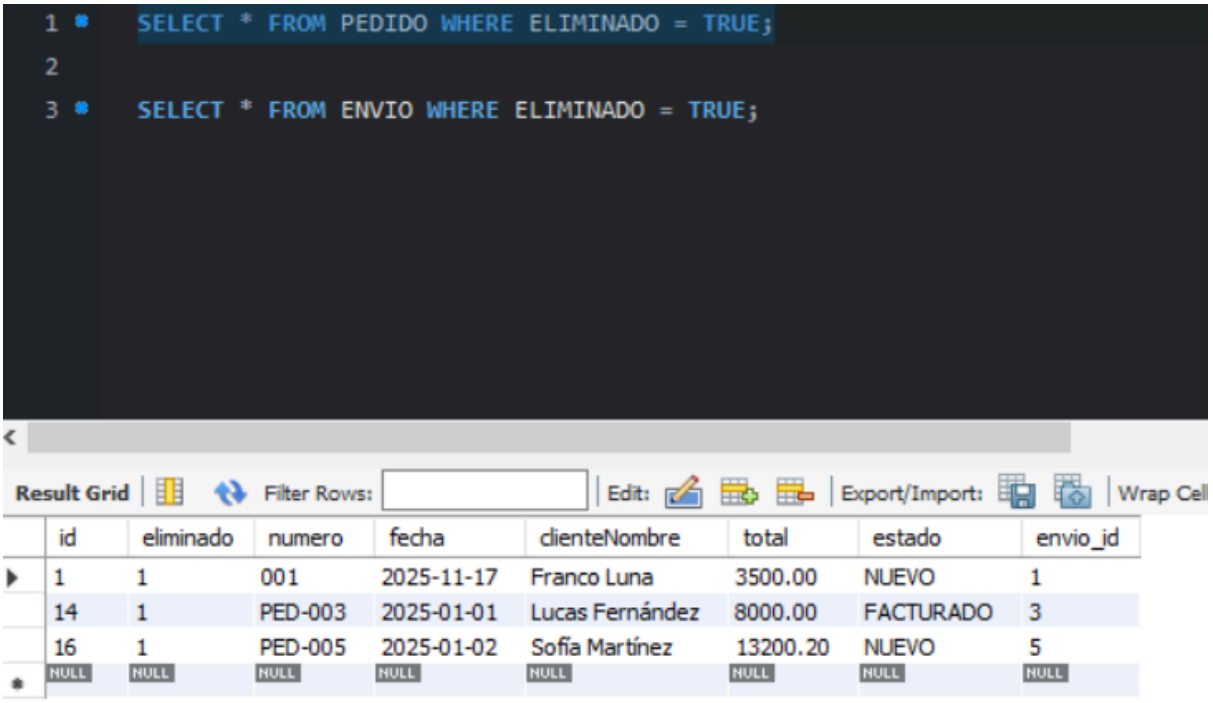


Captura eligiendo la opción “1. Listar Pedidos”:



Se puede ver que en la captura previa se listan los pedidos “NO eliminados”.

A continuación, una captura de la consulta en en MySQL Workbench para demostrar cómo quedan los pedidos “eliminados”:



id	eliminado	numero	fecha	clienteNombre	total	estado	envio_id
1	1	001	2025-11-17	Franco Luna	3500.00	NUEVO	1
14	1	PED-003	2025-01-01	Lucas Fernández	8000.00	FACTURADO	3
16	1	PED-005	2025-01-02	Sofía Martínez	13200.20	NUEVO	5
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Vemos aquí que los pedidos de ID 1, 14 y 16 no se muestran en el programa, sin embargo sí lo hacen cuando realizamos una consulta buscando aquellos pedidos que cumplan con la condición de Eliminado = True(1).
Esto funciona de la misma manera para los envíos.

Conclusiones

El proyecto implementa una arquitectura por capas simple pero funcional, permitiendo administrar pedidos y envíos mediante un esquema claro de entidades, DAOs, servicios y manejo a través de un menú interactivo.
La estructura final logra cumplir con los objetivos principales: persistir datos, respetar la relación 1→1 entre Pedido y Envío, aplicar baja lógica y ofrecer operaciones CRUD completas.
El diseño mantiene una separación adecuada entre la lógica de acceso a datos, la lógica de negocio y la interacción con el usuario, lo que facilita el mantenimiento y la comprensión general del sistema.
A pesar de su complejidad inicial, el desarrollo permitió entender mejor cómo se conectan y trabajan juntas las distintas capas de una aplicación basada en JDBC.

Fuentes y herramientas utilizadas

Herramientas utilizadas:

- Xampp mariaDB 10.4.32
- JDBC /J 8.4.0
- JDK versión 24
- SQL
- MySQL server

Fuentes utilizadas

- Album de videos para la conexión JDBC https://www.youtube.com/playlist?list=PLTd5ehIj0goOO7pOMV77a_nach9JbQ3rH
- OpenAI. (2024). ChatGPT (versión GPT-5.1) [Modelo de lenguaje grande]. https://chat.openai.com/