Trabajo Final Integrador (TFI)

Aplicación Java con relación 1→1 unidireccional + DAO + MySQL

Integrantes

Andres Meshler (Comisión 17): Desarrollador.

Martin Molina (Comisión 17: Analista y Tester.

Agustin Martinez (Comisión 17: Desarrollador.

Link del video: https://youtu.be/7QRvEbFJupk

Elección del dominio

El proyecto implementa un sistema de gestión de Pedidos y Envíos.

Elegimos este dominio porque permite trabajar un escenario realista y aplicable a situaciones del mundo laboral, pero a la vez mantiene un nivel de complejidad adecuado para un Trabajo Práctico Integrador.

- Modelo de datos con entidades relacionadas.
- Persistencia mediante DAO.
- Arquitectura por capas.
- Lógica de negocio con validaciones.
- Interacción mediante un menú por consola.

Este dominio nos permitió:

Aplicar relaciones entre entidades de forma clara
El modelo incluye entidades que naturalmente se relacionan (un Pedido genera un Envío), lo que facilita implementar claves primarias, claves foráneas, relaciones 1→1, validaciones y dependencia entre operaciones.

Trabajar con un flujo de negocio concreto
El proceso de registrar un pedido, generar su envío y consultar su estado refleja un circuito que existe en comercios, mensajerías y sistemas logísticos.
Esto asegura que las decisiones de diseño no sean artificiales, sino tomadas sobre un contexto con sentido.

Implementar una arquitectura por capas sin que sea forzada
El dominio permite separar naturalmente:

Capa de entidades,capa de persistencia (DAO),capa de servicios con reglas de negocio,y capa de presentación (menú).
Esta separación es clave para demostrar buenas prácticas de diseño usadas en la industria.

Permite validar reglas de negocio reales
Tales como no crear un envío sin un pedido previo,no eliminar pedidos que ya tienen envíos asociados,verificar estados válidos y datos obligatorios.
Estas reglas surgen de manera lógica del dominio, lo que demuestra una correcta interpretación del problema.

Es suficientemente amplio para crecer
Elegimos este dominio también porque soporta extensiones naturales, como integrar pagos, agregar repartidores, implementar estados avanzados del envío, calcular costos, o integrar notificaciones al cliente.
Esto permite mostrar posibles mejoras futuras sin cambiar el diseño principal.

Mantiene un equilibrio ideal entre complejidad y alcance

El dominio no es tan simple como para que no tenga dificultad, ni tan complejo como para volverse imposible de terminar dentro de los tiempos de la materia.
Esto nos permitió enfocarnos en aplicar correctamente los contenidos vistos: UML, persistencia, transacciones, validaciones y pruebas.

Es un dominio simple, pero suficientemente completo para mostrar el uso de relaciones entre entidades como Pedido, Envío, y operaciones típicas de CRUD.

Diseño

El proyecto está organizado siguiendo una arquitectura por capas, lo que permite separar responsabilidades, facilitar el mantenimiento y mejorar la comprensión del sistema. Las capas principales encontradas son: entities, dao, service, config y main.

Paquete entities/ – Modelo del dominio

Contiene las clases que representan los datos del sistema:

-Pedido

-Envio

-GenericEntity

Estas clases funcionan como “tablas” del modelo. Solo contienen atributos y métodos simples (getters/setters). No tienen lógica de negocio.

Representan las entidades que se guardan en la base de datos.

Relación entre entidades

El modelo presenta la relación central del dominio:

Un Pedido puede tener un Envío asociado.

Esta es una relación 1 → 1, donde Envio referencia al Pedido mediante una clave foránea (FK).

Decisión de PK/FK

Utilizamos un diseño mas flexible:

Pedido: PK propia (id autoincremental).

Envio: PK propia y una FK (pedido_id) que apunta a Pedido.

Este enfoque es más simple de mantener que compartir una misma PK entre ambas tablas.

Paquete dao/ – Acceso a Datos (persistencia)

Incluye las clases responsables de ejecutar consultas SQL y comunicarse directamente con la base:

-GenericDao

Clase base con operaciones comunes (insert, update, delete, find).

Algunos DAOs extienden de esta clase para evitar duplicar código.

-PedidoDao

Maneja toda la persistencia de la entidad Pedido.

-EnvioDao

DAO específico para la entidad *Envio*. Maneja la persistencia de los envíos, consultas y operaciones asociadas a la tabla correspondiente.

Responsabilidad del DAO

Los DAO ejecutan queries SQL,mapean los resultados a objetos Java,manejan excepciones SQL,y realizan commit/rollback según corresponda.

Son la única capa que interactúa directamente con la base de datos.

Paquete service/ – Reglas de negocio

-PedidoService

-EnvioService

-GenericService

La capa service actúa como intermediaria entre el menú principal y los DAO.

Responsabilidades:

Validar reglas de negocio antes de llamar al DAO.

Coordinar operaciones entre varias entidades.

Decidir cuándo aplicar una transacción completa (por ejemplo: crear pedido + crear envío).

Manejar mensajes de error para el usuario.

Ejemplo: un Envío solo puede crearse si existe un Pedido válido.

Paquete config/ – Configuración del sistema

-DataBaseConnection

Gestiona la conexión global a la base de datos mediante JDBC.

Esta clase centraliza la URL de la BD,usuario/contraseña,apertura y cierre de conexiones,control de autocommit,política de transacciones

Esta separación facilita cambiar de base de datos sin tocar el resto del programa.

Paquete main/ – Interfaz con el usuario y flujo general

Incluye las clases que componen el menú de interacción por consola:

-Main (punto de entrada)

-AppMenu (estructura del menú)

-MenuDisplay (muestra opciones y mensajes)

-MenuHandler (procesa qué hace cada opción del menú)

-TestConexion (verifica que la BD esté correctamente conectada)

Esta capa recibe la entrada del usuario y llama a los servicios correspondientes.

La separación por capas fue elegida porque:

-Facilita localizar errores

-Permite crecer el proyecto sin romper otras partes

-Mejora la legibilidad

-Sigue buenas prácticas usadas en la industria

Cada capa cumple una función muy concreta, evitando mezclar lógica de negocio con persistencia o con la UI.

Persistencia – Base de datos, operaciones y transacciones

-Tabla Pedido

id (BIGINT NOT NULL AUTO_INCREMENT)

eliminado (BOOLEAN DEFAULT FALSE)

numero (VARCHAR NOT NULL UNIQUE)

fecha (DATE NOT NULL)

cliente_nombre (VARCHAR NOT NULL)

total (DECIMAL NOT NULL)

estado (VARCHAR NOT NULL)

envio_id (BIGINT, PRIMARY KEY (id)

Tabla Envio

id (BIGINT NOT NULL AUTO_INCREMENT)

eliminado (BOOLEAN DEFAULT FALSE)

tracking (VARCHAR UNIQUE)

empresa (VARCHAR NOT NULL)

tipo (VARCHAR NOT NULL)

costo (DECIMAL)

fecha_despacho (DATE)

fecha_estimada (DATE)

estado (VARCHAR NOT NULL PRIMARY KEY (id))

Orden de operaciones

-PedidoService valida si los datos son correctos.

-Llama a PedidoDao.insert().

-Si corresponde crear Envío, llama a EnvioDao.insert().

-Si todo salió bien → commit.

-Si algo falla → rollback.

Transacciones

Los commits y rollbacks deben ejecutarse en la capa DAO o dentro de un Service que coordina varias operaciones.

Si un pedido depende de un envío o viceversa, la operación completa debe hacerse en una sola transacción.

Validaciones y reglas de negocio

Estas validaciones se encuentran dentro de los Service antes de llamar a los DAO.

-No se permite crear un pedido sin datos obligatorios (fecha, cliente, detalle).

-No se puede crear un envío sin tener un pedido asociado previamente.

-No se permite eliminar un pedido que ya tiene un envío registrado.

-Campos que deben cumplir formatos (fecha válida, importe numérico, etc).

-IDs deben existir antes de asociarlos.

Pruebas realizadas

En el método insertar() de PedidoService se implementa una transacción manual controlando el ciclo completo:

inicio de la transacción, ejecución de las operaciones (insert del pedido, insert del envío y asociación), commit en caso de éxito y rollback si ocurre cualquier error. Esto asegura la atomicidad, es decir, que todas las operaciones se realizan juntas o ninguna, evitando inconsistencias en la base de datos.

```
¿Confirmar creación del pedido? (S/N): s
Error al crear pedido: Error al crear el pedido: Error al insertar envío: Duplicate entry 't1' for
 key 'envio.tracking'
```

Conclusiones y mejoras futuras

-El sistema cumple con los requisitos básicos del proyecto.

-La arquitectura por capas está bien separada y facilita entender el flujo.

-El uso de DAO y Services permite mantener la lógica ordenada.

-Sería útil documentar mejor el esquema SQL y agregar validaciones más estrictas.

Mejoras futuras

-Implementar pruebas automáticas (JUnit).

-Agregar logs.

-Incluir manejo de excepciones más detallado.

-Agregar comprobación de estado del envío.

-Crear una interfaz gráfica simple en vez del menú por consola.

Fuentes y herramientas utilizadas

-Java

-JDBC / SQL

-IDE: NetBeans / IntelliJ IDEA

-GitHub para control de versiones

-IA (ChatGPT / Copilot) para asistencia puntual en:

    -Ayuda con errores

    -Documentación