

# **TRABAJO FINAL INTEGRADOR PROGRAMACIÓN II**

**Coordinador: Carlos Martinez**

**Grupo 167**

**Integrantes:**

Rodrigo Aguirre - DNI 39.549.930 - Comisión 16

Sol Yoon - DNI 92.906.418 - Comisión 6

Magdalena Darchez - DNI 47.402.878 - Comisión 7

Celeste Monsalbe - DNI 45.028.610 - Comisión 4

**Fecha de entrega: 17/11/2025**

**Link Video:**

[https://drive.google.com/file/d/1LbIPYBHysrLO3ux1eTZD7J7LxQqorlVy/view?usp=drive\\_link](https://drive.google.com/file/d/1LbIPYBHysrLO3ux1eTZD7J7LxQqorlVy/view?usp=drive_link)



# Trabajo Final Integrador (TFI)

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

### Integrantes y Roles

Integrante	Rol
Aguirre	Diseño del UML, entidades y configuración de la conexión a
Rodrigo	BD.
Sol Yoon	Implementación de la capa DAO y consultas SQL
Celeste Monsalbe	Implementación de la capa Service y manejo de transacciones
Magdalena Darchez	Desarrollo del menú principal, pruebas funcionales y documentación.

### Elección del dominio y justificación

Se seleccionó el dominio **Pedido–Envío** por ser un caso representativo en sistemas de logística. Permite modelar una relación **uno a uno**, sencilla pero funcional, donde cada pedido posee exactamente un envío asociado. Este dominio también es adecuado para aplicar transacciones, validaciones y operaciones CRUD típicas. Su estructura clara facilita la demostración de la arquitectura en capas solicitada en el trabajo.

### Justificación del diseño 1→1 con FK única

La relación 1→1 se implementó mediante una **foreign key única** en la tabla **pedido**.

Se eligió este enfoque porque:

- es simple de implementar y mantener;
- permite manejar Pedido y Envío con ciclos de vida independientes;
- evita las restricciones rígidas de una clave primaria compartida;
- asegura la unicidad mediante **UNIQUE**, garantizando la relación 1→1.

# Arquitectura por capas

El proyecto está estructurado en cinco capas, cada una con responsabilidades claras:

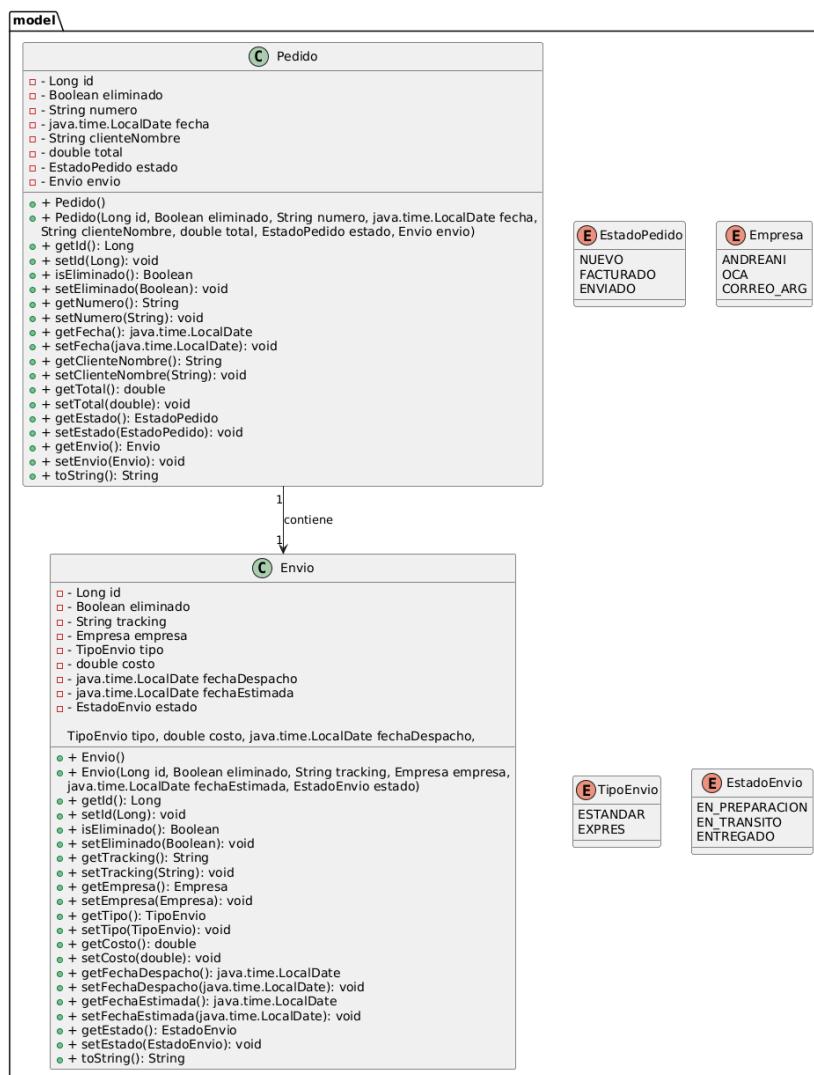
## Capa entities.

Contiene las clases que representan el modelo del dominio:

- Pedido
- Envío

Responsabilidad:

- ✓ Mantener únicamente atributos y getters/setters.
- ✓ No contiene lógica de acceso a datos ni reglas de negocio.



## Diagrama UML de entidades

Representa a las clases definidas para cada entidad y su relación. Cada clase contiene atributos privados relacionados a la instancia, un método constructor por defecto vacío, otro completo que inicializa las variables del objeto, y métodos públicos setter y getter para acceder a los atributos privados. Por otro lado, se establece una relación de asociación unidireccional donde Pedido contiene a Envio, sin embargo Envio no tiene referencia y no conoce a la clase Pedido.

## Capa config.

Clases:

- DatabaseConnection
- RunScriptsSQL

Responsabilidad:

- ✓ Establecer conexiones JDBC con el servidor MySQL y con la base de datos.
- ✓ Manejar errores de conexión.
- ✓ Correr scripts SQL para la creación de la base de datos, el esquema y la población masiva con datos de prueba.

### DatabaseConnection

La clase DatabaseConnection se encarga de gestionar la conexión con la base de datos MySQL de forma centralizada. Su función principal es proporcionar métodos estáticos para obtener conexiones JDBC sin que el resto del sistema tenga que preocuparse por detalles como la URL, el usuario o la contraseña.

La clase utiliza la librería Dotenv para cargar automáticamente los parámetros necesarios para la conexión:

La clase ofrece dos métodos públicos y estáticos:

- getConnection()  
Devuelve una conexión JDBC a la base de datos seleccionada.  
Se usa en la aplicación para operaciones CRUD.
- getServerConnection()  
Conecta únicamente al servidor MySQL sin base específica.  
Se usa, por ejemplo, para crear la base en la primera ejecución.

## **RunScriptsSQL**

La clase RunScriptsSQL es un utilitario cuya función principal es automatizar la creación de la base de datos, la generación del esquema y la carga inicial de datos, ejecutando scripts SQL directamente desde Java. Se utiliza una sola vez, al iniciar el proyecto.

## **Capa DAO.**

Clases:

- GenericDAO
- PedidoDAO
- EnvioDAO

Responsabilidad:

- ✓ Ejecutar operaciones CRUD mediante consultas SQL.
- ✓ Recibir una conexión ya abierta desde los Services.
- ✓ No manejar transacciones.

La capa DAO es responsable de ejecutar las operaciones CRUD sobre la base de datos mediante JDBC. En nuestro proyecto, esta capa está compuesta por PedidoDAO y EnvioDAO, ambos implementando la interfaz GenericDAO. Su función es exclusivamente acceder a los datos, sin aplicar reglas de negocio ni manejar transacciones.

### **Recepción de la conexión desde la capa Service**

En esta arquitectura, los DAO no crean su propia conexión; en cambio, reciben una conexión abierta que es gestionada por la capa Service. La conexión proviene de la clase DatabaseConnection, pero es la capa Service quien la solicita, la valida y controla su ciclo de vida dentro de una transacción.

Esto permite que la capa Service administre la transacción completa, aplicando:

- conn.setAutoCommit(false)
- conn.commit()
- conn.rollback()

De esta manera, varias operaciones DAO pueden ejecutarse dentro de la misma transacción (por ejemplo, insertar un Pedido y su Envío), asegurando atomicidad y consistencia en la base de datos.

## Uso de PreparedStatement

Todas las consultas SQL del DAO utilizan PreparedStatement:

- Previene inyecciones SQL.
- Maneja parámetros correctamente (setString, setDate, setDouble, etc.).
- Permite ejecutar sentencias precompiladas en el motor.

## Mapeo de datos con ResultSet

En las operaciones de lectura, el DAO utiliza ResultSet para recorrer y transformar las filas obtenidas desde MySQL en objetos del dominio:

```
ResultSet rs = ps.executeQuery();
if (rs.next()) {
    Pedido p = mapearPedido(rs);
}
```

La función mapearPedido encapsula la lógica de conversión de una fila del ResultSet en un objeto Pedido, manteniendo limpio el código.

## Obtención de claves generadas (getGeneratedKeys)

Cuando se crea un nuevo Pedido, el ID autoincremental asignado por MySQL se obtiene mediante asegurando que el objeto Java quede sincronizado con la base de datos.:

```
try (ResultSet rs = ps.getGeneratedKeys()) {
    if (rs.next()) {
        pedido.setId(rs.getLong(1));
    }
}
```

## Relación 1→1 y colaboración entre DAOs

La relación unidireccional entre Pedido → Envío está implementada directamente en el DAO:

- PedidoDAO contiene una instancia de EnvioDAO.
- Al leer un Pedido, también se consulta su Envío asociado.
- Al actualizar un Pedido, primero se actualiza el Envío relacionado.

## **Capa services.**

Clases:

- GenericService
- PedidoService
- EnvioService

Responsabilidad:

- ✓ Manejar reglas de negocio.
- ✓ Validaciones.
- ✓ Manejo de transacciones: commit y rollback.
- ✓ Coordinación entre DAOs.

### **Reglas de Negocio y Pruebas de Rollback**

Las reglas de negocio aplicadas garantizan la coherencia del dominio: cada Pedido debe tener un único Envío, no se permite crear o actualizar registros marcados como eliminados y los datos clave, como el tracking del Envío, deben ser únicos y válidos. Además, no se admite registrar pedidos sin envío asociado ni cargar valores inválidos (por ejemplo, fechas inconsistentes o totales negativos).

Para comprobar el manejo de transacciones, se realizaron pruebas que forzaron errores, como repetir un tracking existente, ingresar fechas incorrectas o totales negativos. En todos los casos, la operación fue revertida correctamente mediante rollback, asegurando que ningún dato quede registrado cuando ocurre una falla.

### **Manejo de Transacciones**

Las operaciones que afectan a más de una tabla se ejecutan dentro de una transacción para asegurar atomicidad. En este proyecto, crear o actualizar un Pedido implica también operar sobre su Envío asociado, por lo que ambas acciones deben completarse juntas. El proceso general consiste en desactivar el auto-commit, ejecutar las operaciones en los DAO y confirmar los cambios con commit() si todo sale bien. Si ocurre alguna falla o validación incorrecta, se ejecuta rollback() para mantener la base de datos consistente. De esta forma, nunca queda registrado un Pedido sin su Envío correspondiente.

### **Validaciones Aplicadas**

Antes de guardar o actualizar datos, el Service aplica validaciones para evitar errores y asegurar integridad. Se verifica que el tracking del Envío sea único y no nulo, que las fechas sean coherentes y que el costo no sea negativo. En el caso de Pedido, se controla que el total sea positivo, que el estado sea válido y que el nombre del cliente no esté vacío.

Si alguna validación falla, se lanza una excepción y la transacción se revierte automáticamente.

## **Capa app (interfaz / menú).**

Clase:

- AppMenu
- Main

Responsabilidad:

- ✓ Punto de entrada a la aplicación.
- ✓ Interactuar con el usuario.
- ✓ Solicitar datos.
- ✓ Llamar métodos de los Services.
- ✓ Mostrar listados y resultados.

### **Implementación del Menú Principal (Capa App)**

Se desarrolló la capa de interfaz de usuario (app), formada por las clases Main y AppMenu, con el objetivo de permitir la interacción del usuario con todas las operaciones CRUD definidas en la aplicación.

El menú funciona como punto de entrada al sistema y coordina la comunicación con la capa Service. No realiza lógica de negocio; únicamente solicita datos y muestra resultados, respetando la arquitectura en capas.

### **Pruebas realizadas sobre Pedido**

Alta de pedidos con validación de campos, modificación de pedidos utilizando IDs existentes, búsqueda por ID, listado completo de pedidos, eliminación con confirmación por parte del usuario.

### **Pruebas realizadas sobre Envío**

Creación de envíos asociados a pedidos válidos, edición de datos de un envío (domicilio, transportista, fechas), eliminación de envíos y verificación posterior de la relación 1→1.

### **Pruebas integradas Pedido–Envío**

Validación de que cada Pedido tenga un único Envío asociado, comprobación de errores controlados al intentar asociar más de un envío al mismo pedido, visualización conjunta de datos de Pedido y Envío.

## Pruebas de comportamiento general del menú

Manejo de excepciones ante entradas inválidas, flujo continuo incluso cuando el usuario proporciona datos incorrectos, verificación de mensajes informativos y consistencia en la interfaz textual.

*Ejemplo menú principal:*

```
===== MENU PRINCIPAL =====
1. CRUD de Pedidos
2. CRUD de Envíos
0. Salir
Elegi una opcion:
```

*Ejemplo eliminación de pedido:*

```
===== CRUD PEDIDO =====
1. Crear Pedido (con Envio opcional)
2. Ver Pedido por ID
3. Listar Pedidos
4. Actualizar Pedido
5. Eliminar Pedido (baja logica)
6. Buscar por numero
0. Volver
Elegi una opcion: 5
ID del pedido a eliminar: 1
Pedido marcado como eliminado.
```

## Conclusiones

El proyecto permitió integrar conceptos de POO, JDBC y persistencia mediante una arquitectura en capas clara y mantenible. La implementación del dominio Pedido–Envío permitió trabajar con transacciones, validaciones, DAO, PreparedStatement y manejo de excepciones. El sistema final demuestra un flujo completo de alta, consulta, actualización y eliminación lógica, aplicando buenas prácticas profesionales.

## Mejoras futuras

- Incorporar autenticación y permisos.
- Agregar más entidades (Producto, Cliente, DetallePedido).
- Implementar pruebas unitarias.
- Evolucionar a una API o aplicación web utilizando Spring Boot.

## Fuentes y herramientas utilizadas

**Herramientas:** Java 21, JDBC, MySQL, MySQL Workbench, Apache NetBeans, GitHub.

**Fuentes:** Documentación oficial de JDBC y MySQL, material de cátedra, apuntes teóricos, ChatGPT como apoyo.