



Trabajo Práctico Integrador - Programación II

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

Universidad Tecnológica Nacional

Tecnicatura Universitaria en Programación

Programación II

Profesor:

Alumnos:

Eyo Bartl Florencia

Fredes Lucas

Ferreyra Matias



Informe - Gestión de Pedidos y Envíos

1. Integrantes y Roles

Integrantes:

Eyo Bartl Florencia

Fredes Lucas

Ferreya Matias

Roles y Metodología del trabajo: El presente trabajo fue llevado a cabo de forma conjunta, por los tres integrantes del grupo.

La metodología de trabajo consistió en reuniones sincrónicas a través de Google Meet, donde se abordaron todas las etapas del ciclo de vida del desarrollo de software de manera colaborativa.

Descripción del proceso en conjunto:

Diseño de arquitectura del proyecto.

Realización del UML.

Diseño e implementación de la base de datos.

Definición de interfaces genéricas (GenericDao y GenericService).

Desarrollo de las clases de la capa de acceso a datos (PedidoDao y EnvioDao).

Desarrollo de las clases de lógica de negocio (PedidoService y EnvioService).



Implementación de validaciones de negocio (validateNumeroUnique, etc.).

Manejo de las transacciones (Commit/Rollback) usando TransactionManager.

Desarrollo de la interfaz de usuario por consola (AppMenu, MenuHandler y MenuDisplay).

Creación de datos de prueba (insert_datos.sql).

Ejecución de pruebas funcionales (CRUD) y pruebas de transacciones (Rollback).

2. Elección del Dominio y Justificación

Para este trabajo, se seleccionó el dominio **Pedido** → **Envío**.

Justificación:

Esta relación representa un escenario de negocio (e-commerce, logística) muy común en el mundo real. Un **Pedido** (Entidad A) debe estar asociado a un único **Envío** (Entidad B) para ser despachado, y un **Envío** específico pertenece a un solo **Pedido**.

Esta elección nos pareció la correcta por las siguientes razones:

1. Relación 1 a 1 clara: Modela perfectamente la restricción 1 a 1 unidireccional solicitada.
2. Lógica: Permite implementar reglas de negocio complejas, como la gestión de estados (por ej. un `Pedido` no puede pasar a "ENVIADO" si no tiene un Envío asociado) y el uso de Enums (Estados, EstadosEnvios, Empresas).
3. Transacciones Significativas: La creación de un `Pedido` con su `Envío` asociado es un caso de uso que nos permite demostrar la atomicidad (transacciones). Si falla la creación del Envío (por ej. tracking duplicado), debe fallar la creación del Pedido, y viceversa.



3. Diseño y Decisiones Clave

Decisión de Diseño 1 a 1: Clave Foránea Única (FK Única)

La consigna permitía dos enfoques para la relación 1-a-1: Clave Primaria Compartida o Clave Foránea Única.

Se optó por la Clave Foránea Única (FK Única).

Implementación: La tabla **pedidos** (Entidad A) contiene la columna **envio_id**, la cual es una Clave Foránea que apunta al id de la tabla envíos (Entidad B). Para garantizar la relación 1 a 1 (un pedido sólo puede tener un envío y un envío solo puede estar en un pedido), se aplicó una restricción ``UNIQUE`` a la columna ``envio_id`` en la tabla ``pedidos``.

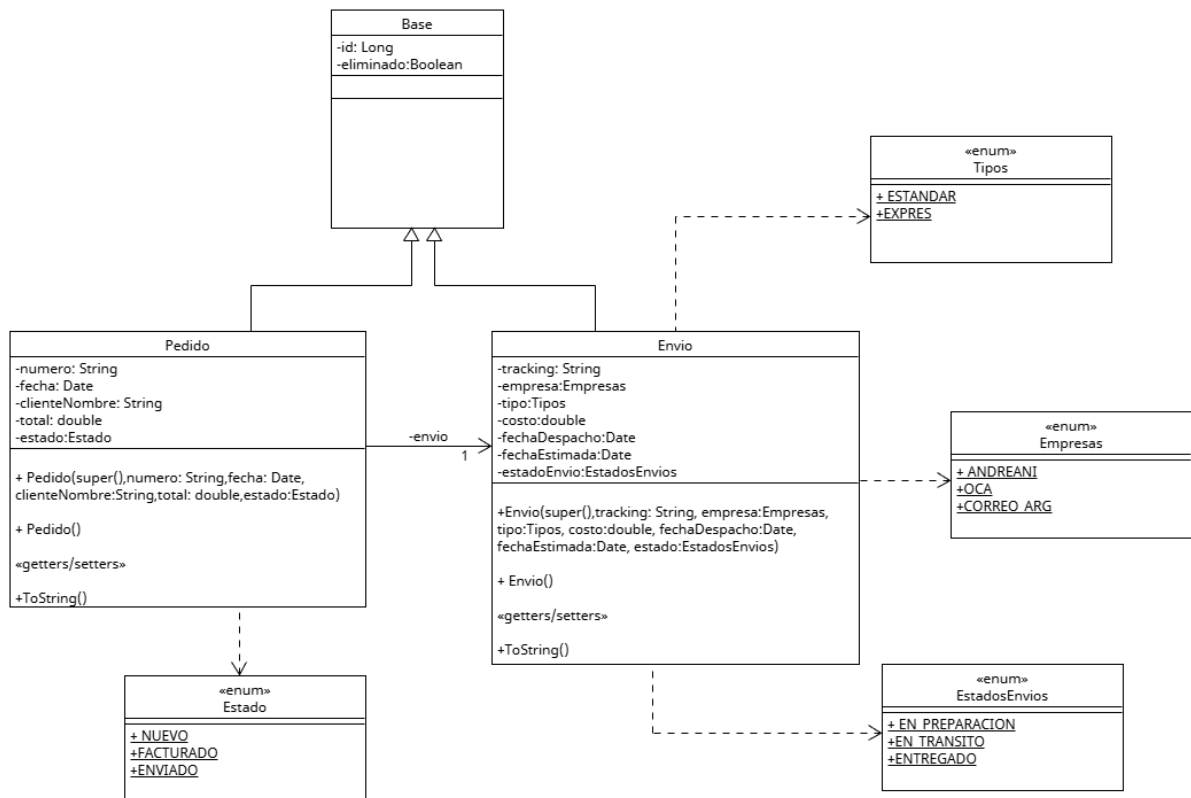
Justificación: Esta estrategia nos pareció más óptima que la PK compartida ya que nos da mayor flexibilidad, necesaria para nuestro dominio:

Ciclos de vida independientes: Permite que un **Envío** exista en la base de datos antes de ser asignado a un **Pedido**. Esto aplica con nuestra funcionalidad de "Crear Envío" y "Asignar Envío a Pedido", donde un envío puede generarse primero (por ej. en el depósito) y luego asociarse al pedido correspondiente.

Claridad de PK: Mantiene las claves primarias (id) de ambas tablas como números auto-incrementables, facilitando su gestión.

Diagrama UML

A continuación, se presenta el Diagrama de Clases UML que modela nuestra solución. En el mismo se puede observar la herencia de la clase ``Base`` (que implementa ``id`` y ``eliminado``), la relación unidireccional 1 a 1 y el uso de Enums.



4. Arquitectura por Capas

El proyecto sigue una arquitectura de 4 capas para asegurar la separación de responsabilidades, basándose en la estructura de paquetes solicitada:

Paquete Model (Entidades):

Contiene las clases de dominio (Pedido, Envio) que reflejan los datos.

Incluye la clase abstracta Base que provee los campos id y eliminado (este último para la baja lógica).

Define todas las enumeraciones (Estados, Empresas, Tipos, EstadosEnvios) que restringen los valores del dominio.

Paquete `Config` (Configuración):



DatabaseConnection: Clase utilitaria que gestiona la conexión JDBC con la base de datos MySQL. Carga el driver y provee un método estático **getConnection()**.

TransactionManager: Clase que encapsula la lógica de transacciones JDBC. Implementa `AutoCloseable` para ser usada con “try-with-resources”, garantizando que **commit()** o **rollback()** se ejecuten siempre.

Paquete Dao (Data Access Object):

Responsable de la persistencia y el acceso a datos (CRUD).

GenericDao: Interfaz que define el contrato CRUD estándar.

PedidoDao y EnvioDao: Implementaciones concretas que ejecutan el SQL usando `PreparedStatement` (para prevenir SQL Injection) y manejan `ResultSet`.

Exponen métodos transaccionales (por ej. `insertTx(T entidad, Connection conn)`) que aceptan una conexión externa, permitiéndoles participar en una transacción manejada por la capa de servicio.

Paquete Service (Lógica de Negocio):

Maneja las operaciones y contiene las reglas de negocio.

GenericService: Interfaz que define el contrato para los servicios.

EnvioService: Valida los datos de un envío (ej. `validateEnvio`).

PedidoService: Contiene la lógica de negocio más compleja, como `validateNumeroUnique`` (para asegurar que no haya números de pedido duplicados) y gestiona las transacciones al insertar o actualizar un Pedido y su Envío asociado.

Paquete trabajointegrador2 (UI / Main):

Punto de entrada de la aplicación (`AppMenu` y `Main`).



MenuDisplay: Responsable únicamente de imprimir los menús en consola.

MenuHandler: Es el "controlador" de la UI. Recibe las entradas del usuario (a través de `Scanner`), las procesa, invoca a la capa Service y maneja las excepciones de usuario (por ej. `NumberFormatException`).

5. Persistencia y Transacciones

Estructura de la Base

La persistencia se logró en MySQL (script `Prog2_TPI.sql`). La decisión más importante en este apartado fue la restricción **UNIQUE** en la columna **envio_id** de la tabla pedidos, que fuerza la relación 1 a 1 en la base de datos.

Transacciones: Orden de Operaciones (Atomicidad)

El requisito más crítico del TPI era asegurar la atomicidad ("todo o nada") en operaciones compuestas. El caso de uso principal es "Crear un Pedido con un nuevo Envío".

Si el usuario crea un Pedido y un Envío en la misma operación, ambas entidades deben insertarse correctamente, o ninguna debe hacerlo.

Este control se implementó en **PedidoService.insertar(Pedido pedido)**:

Inicio: El `Service` obtiene una conexión de `DatabaseConnection` y la pasa al `TransactionManager`, que se inicializa en un bloque `try-with-resources`.

tx.startTransaction(): El manager deshabilita el autoCommit (conn.setAutoCommit(false)).

Operación 1 (Hijo): Se invoca a `envioService.getEnvioDao().insertTx(pedido.getEnvio(), tx.getConnection())`. Se inserta el Envío usando la conexión compartida. Si el tracking está duplicado, esto lanza una `SQLException`.



Operación 2 (Padre): Se invoca a `pedidoDao.insertTx(pedido, tx.getConnection())`. Se inserta el Pedido usando la misma conexión. Si el número está duplicado, esto lanza una `SQLException`.

Confirmación: Si ambas operaciones (3 y 4) tienen éxito, se alcanza la línea **`tx.commit()`**. La transacción se confirma y los datos se guardan permanentemente.

Manejo de Fallos (Rollback)

El **rollback** se gestiona de dos maneras:

Explícito (Catch): Si se captura una **Exception** (por ej. `SQLException` por clave duplicada) dentro de `PedidoService`, se llamaría a **`tx.rollback()`**.

Implícito (AutoCloseable): Gracias a `AutoCloseable`, si una excepción ocurre y no es capturada inmediatamente, el bloque “try-with-resources” finaliza. El método **`TransactionManager.close()`** se ejecuta automáticamente. Este método comprueba si la transacción sigue activa y, si lo está, ejecuta un `rollback()` antes de cerrar la conexión.

Esto garantiza que la base de datos nunca quede en un estado inconsistente.

6. Validaciones y Reglas de Negocio

Las validaciones se implementaron en la capa `Service`, antes de intentar cualquier operación en la base de datos.

Principales validaciones implementadas:

Validación de Nulidad/Vacío:

En `PedidoService.validatePedido`: Se verifica que “numero” y “clienteNombre” no sean nulos o vacíos.

En ``EnvioService.validateEnvio``: Se verifica que ``tracking``, ``empresa``, ``tipo`` y ``estadoEnvio`` no sean nulos.



Validación de Formato (Datos):

En PedidoService.validatePedido: Se verifica que “total” no sea negativo.

En EnvioService.validateEnvio: Se verifica que “costo” no sea negativo.

En MenuHandler: Se maneja NumberFormatException y DateTimeParseException si el usuario ingresa formatos de número o fecha incorrectos.

Regla de Negocio (Unicidad):

PedidoService.validateNumeroUnique(String numero, Long pedidold): Esta es la regla de negocio más importante. Antes de insertar o actualizar, el servicio consulta al DAO (buscarPorNumero) para asegurar que el número de pedido no esté ya en uso por otro pedido.

7. Pruebas Realizadas

Se realizaron múltiples pruebas funcionales y de integridad desde el menú de consola:

Prueba 1: Creación Exitosa (Transacción Commit)

Acción: Se usó la Opción 1 para crear un Pedido nuevo con un Envío nuevo.

Resultado: El menú informó "Pedido creado exitosamente".

Verificación SQL: Se constató que ambas filas (en `pedidos` y `envios`) fueron creadas y el `envio_id` en el pedido coincidía con el `id` del envío.



***** GESTION DE PEDIDOS Y ENVIOS *****

```
1. Crear pedido
2. Listar pedidos
3. Actualizar pedido
4. Eliminar pedido
5. Crear envio
6. Listar envios
7. Actualizar envio
8. Eliminar envio
9. Asignar envio a pedido
10. Eliminar envio de pedido
0. Salir
Ingrese una opcion: 1
Numero de pedido: PED-006
Nombre del cliente: Alejandro Martinez
Total: 95000
```

----- Estados de Pedido -----

```
1. NUEVO
2. FACTURADO
3. ENVIADO
Seleccione estado: 1
Desea crear un envio para este pedido? (s/n): s
Numero de tracking: TRK-RKG-4566
Costo: 1500
```

----- Empresas de Envio -----

```
1. ANDREANI
2. OCA
3. CORREO_ARG
Seleccione empresa: 1
```

----- Tipos de Envio -----

```
1. ESTANDAR
2. EXPRES
Seleccione tipo: 2
```

----- Estados de Envio -----

```
1. EN_PREPARACION
2. EN_TRANSITO
3. ENTREGADO
Seleccione estado: 1
Desea agregar fecha de despacho? (s/n): n
Desea agregar fecha estimada? (s/n): n
Pedido guardado exitosamente con su envio asociado
Pedido creado exitosamente con ID: 6
```

	id	numero	fecha	cliente_nombre	total	estado	envio_id	eliminado
▶	1	PED-001	2025-10-25	Juan Perez	150000.00	ENVIADO	1	0
	2	PED-002	2025-11-08	Maria Garcia	45600.50	ENVIADO	2	0
	3	PED-003	2025-11-11	Tecno Sur SA	980000.00	ENVIADO	3	0
	4	PED-004	2025-11-12	Carlos Lopez	12500.00	FACTURADO	NULL	0
	5	PED-005	2025-11-13	Ana Martinez	32000.00	NUEVO	NULL	0
	6	PED-006	2025-11-14	Alejandro Martinez	95000.00	NUEVO	5	0
✱	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL



Result Grid								
		Filter Rows:		Edit:		Export/Import:		Wrap Cell Content: <input type="checkbox"/>
id	tracking	empresa	tipo	costo	fecha_despacho	fecha_estimada	estado_envio	eliminado
1	TRK-1001	ANDREANI	ESTANDAR	5500.50	2025-11-01	2025-11-05	ENTREGADO	0
2	TRK-1002	OCA	EXPRES	8200.00	2025-11-10	2025-11-12	EN_TRANSITO	0
3	TRK-1003	CORREO_ARG	ESTANDAR	4100.00	NULL	NULL	EN_PREPARACION	0
4	TRK-1004	ANDREANI	EXPRES	9000.00	NULL	NULL	EN_PREPARACION	0
5	TRK-1005	ANDREANI	EXPRES	1500.00	NULL	NULL	EN_PREPARACION	0
6	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Prueba 2: Fallo de Transacción (Rollback por Duplicado)

Acción: Se intentó crear un Pedido nuevo, pero se utilizó un número de pedido (ej. PED-001) que ya existía en la base de datos (cargado desde insert_datos.sql).

Resultado: PedidoService lanzó una excepción (IllegalArgumentException: "Ya existe un pedido con el número: PED-001"), que fue capturada por el MenuHandler y mostrada al usuario.

Verificación SQL: Se verificó que ningún dato fue insertado. Ni el Pedido ni el Envío asociado (que sí era válido) se guardaron, demostrando el rollback exitoso de la transacción.



```
***** GESTION DE PEDIDOS Y ENVIOS *****
1. Crear pedido
2. Listar pedidos
3. Actualizar pedido
4. Eliminar pedido
5. Crear envio
6. Listar envios
7. Actualizar envio
8. Eliminar envio
9. Asignar envio a pedido
10. Eliminar envio de pedido
0. Salir
Ingrese una opcion: 1
Numero de pedido: PED-001
Nombre del cliente: Lopez Mabel
Total: 60000

----- Estados de Pedido -----
1. NUEVO
2. FACTURADO
3. ENVIADO
Seleccione estado: 1
❖Desea crear un envio para este pedido? (s/n): s
Numero de tracking: TRK-1006
Costo: 3000

----- Empresas de Envio -----
1. ANDREANI
2. OCA
3. CORREO_ARG
Seleccione empresa: 2

----- Tipos de Envio -----
1. ESTANDAR
2. EXPRES
Seleccione tipo: 2

----- Estados de Envio -----
1. EN_PREPARACION
2. EN_TRANSITO
3. ENTREGADO
Seleccione estado: 1
❖Desea agregar fecha de despacho? (s/n): n
❖Desea agregar fecha estimada? (s/n): n
Error al crear pedido: Ya existe un pedido con el numero: PED-001
```



	id	numero	fecha	cliente_nombre	total	estado	envio_id	eliminado
▶	1	PED-001	2025-10-25	Juan Perez	150000.00	ENVIADO	1	0
	2	PED-002	2025-11-08	Maria Garcia	45600.50	ENVIADO	2	0
	3	PED-003	2025-11-11	Tecno Sur SA	980000.00	ENVIADO	3	0
	4	PED-004	2025-11-12	Carlos Lopez	12500.00	FACTURADO	NULL	0
	5	PED-005	2025-11-13	Ana Martinez	32000.00	NUEVO	NULL	0
	6	PED-006	2025-11-14	Alejandro Martinez	95000.00	NUEVO	5	0
★	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

	id	tracking	empresa	tipo	costo	fecha_despacho	fecha_estimada	estado_envio	eliminado
▶	1	TRK-1001	ANDREANI	ESTANDAR	5500.50	2025-11-01	2025-11-05	ENTREGADO	0
	2	TRK-1002	OCA	EXPRES	8200.00	2025-11-10	2025-11-12	EN_TRANSITO	0
	3	TRK-1003	CORREO_ARG	ESTANDAR	4100.00	NULL	NULL	EN_PREPARACION	0
	4	TRK-1004	ANDREANI	EXPRES	9000.00	NULL	NULL	EN_PREPARACION	0
	5	TRK-1005	ANDREANI	EXPRES	1500.00	NULL	NULL	EN_PREPARACION	0
★	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Prueba 3: Baja Lógica (Soft Delete)

Acción: Se usó la Opción 4 para eliminar un pedido existente.

Resultado: El menú informó "Pedido eliminado exitosamente".

Verificación SQL: Se ejecutó `SELECT * FROM pedidos WHERE id = ?` y se constató que la fila no fue borrada sino que el campo "eliminado" cambió de FALSE a TRUE.

```
***** GESTION DE PEDIDOS Y ENVIOS *****
1. Crear pedido
2. Listar pedidos
3. Actualizar pedido
4. Eliminar pedido
5. Crear envio
6. Listar envios
7. Actualizar envio
8. Eliminar envio
9. Asignar envio a pedido
10. Eliminar envio de pedido
0. Salir
Ingrese una opcion: 4
ID del pedido a eliminar: 1
Pedido eliminado exitosamente.
```



	id	numero	fecha	cliente_nombre	total	estado	envio_id	eliminado
▶	1	PED-001	2025-10-25	Juan Perez	150000.00	ENVIADO	1	1
	2	PED-002	2025-11-08	Maria Garcia	45600.50	ENVIADO	2	0
	3	PED-003	2025-11-11	Tecno Sur SA	980000.00	ENVIADO	3	0
	4	PED-004	2025-11-12	Carlos Lopez	12500.00	FACTURADO	NULL	0
	5	PED-005	2025-11-13	Ana Martinez	32000.00	NUEVO	NULL	0
	6	PED-006	2025-11-14	Alejandro Martinez	95000.00	NUEVO	5	0
★	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

8. Conclusiones y Mejoras Futuras

Conclusiones

Durante el desarrollo de este trabajo integrador se logró implementar una aplicación funcional que cumple con todos los requisitos técnicos solicitados. Los objetivos clave alcanzados fueron:

El dominio de la persistencia manual con JDBC puro, sin la asistencia de un ORM.

La correcta implementación del patrón DAO y la Arquitectura por Capas, logrando una alta cohesión y bajo acoplamiento.

El manejo robusto de Transacciones con commit y rollback, asegurando la integridad de los datos (atomicidad) mediante una clase **TransactionManager** reutilizable.

Mejoras Futuras

Si bien el proyecto es funcional, existen varias áreas de mejora que podrían implementarse en futuras versiones como por ejemplo:

Mejora de la UI (MenuHandler): Implementar bucles de re-intento (while) para la validación de entradas en **MenuHandler**. Actualmente, si el usuario ingresa una fecha en formato incorrecto, la operación se cancela; lo ideal sería volver a solicitar solo ese dato, manteniendo una lógica de ejecución perpetua hasta que se desee voluntariamente terminarla.



Revertir Baja Lógica: Implementar la funcionalidad recuperar(long id) (Opción 11 del menú, actualmente no implementada) para revertir la baja lógica (poner eliminado = FALSE)

Inyección de Dependencias (DI): Aunque la DI se usó manualmente (pasando DAOs a los constructores de los Services).

Otra mejora que podría ser pertinente es la creación de una interfaz de usuario para controlar el menú. De esta forma el usuario no requiere interacción por consola con una API como por ejemplo Java Swing.

Generación automática y secuencial tanto del número de pedido como del número de tracking, con un formato preestablecido.

Pruebas Unitarias: Implementar pruebas unitarias (ej. con JUnit) para la capa de Servicio, la capa DAO para probar la lógica de negocio y las validaciones de forma aislada.

Otra mejora que se podría implementar es la utilización de índices para las consultas SQL.

9. Fuentes y Herramientas Utilizadas

IDE: Apache NetBeans.

SGBD: MySQL Workbench 8.0. Se utilizó parte la base de datos de Mysql que se desarrolló para el trabajo integrador de base de datos

Control de versiones: Git / GitHub.

Documentación: Documentación oficial de Java (Oracle) para implementar el uso del JDK. Y la implementación de Mysql

Asistencia de IA: Se utilizó un asistente de Inteligencia Artificial (Gemini) para las siguientes tareas:



Auditoría de código y revisión de cumplimiento contra las consignas del TFI.

Sugerencias de mejora (ej. manejo de AutoCloseable en TransactionManager).

Generación de los archivos SQL de prueba (datos_prueba.sql).

Revisión y corrección del diagrama UML.

Asistencia en la redacción y estructuración de este informe.

9.1 Referencias Bibliográficas

Bloch, J. 2018 *Effective Java* (3rd ed.). Addison-Wesley.

Fowler, M. 2003 *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional.

Oracle. (s.f.). *Java Platform Standard Edition & Java Development Kit Version 24 API Specification*. Recuperado de <https://docs.oracle.com/en/java/javase/24/docs/api/index.html>

Oracle. s.f. *MySQL 8.0 Reference Manual* [Documentación oficial de MySQL]. Recuperado de <https://dev.mysql.com/doc/refman/8.0/en/>