

Trabajo Final Integrador (TFI)

Grupo 205 – Pedido → Envío

Alumnos:

- Reynoso Franco.
- Ríos Jonathan.
- Rodríguez Santiago.
- Rojas Maximiliano.

Introducción.

El presente informe tiene como objetivo documentar el desarrollo del **Trabajo Final Integrador** correspondiente a la asignatura *Programación II*. El proyecto se centra en la implementación de un sistema orientado al dominio **Pedido → Envío**, aplicando los principios de la programación orientada a objetos y las buenas prácticas de diseño por capas.

La solución propuesta se estructura en diferentes módulos que representan las capas fundamentales de una arquitectura lógica: **entidades, acceso a datos (DAO), servicios y presentación**. Cada una de estas capas cumple un rol específico en la separación de responsabilidades, favoreciendo la mantenibilidad y escalabilidad del sistema.

El desarrollo se realizó de manera colaborativa por un equipo conformado por cuatro integrantes: **Franco, Jonathan, Maximiliano y Santiago**. La metodología adoptada se basó en la división de tareas según las competencias técnicas y la lógica del proyecto, permitiendo una integración eficiente de las distintas partes. Este enfoque fomentó el trabajo en equipo, la comunicación constante y la aplicación de conceptos aprendidos durante la cursada.

A continuación, se detallan los objetivos del proyecto, la descripción del dominio, el diseño de la solución, la implementación técnica y la distribución de roles, así como las conclusiones obtenidas tras la finalización del trabajo.

Elección del dominio y justificación.

Para el desarrollo del trabajo final integrador se seleccionó el dominio Pedido -> envío, correspondiente a una relación unidireccional 1 -> 1 donde cada Pedido puede generar, como máximo, un único envío asociado. La elección de este dominio se fundamenta en varios motivos:

En primer lugar, representa un caso realista y ampliamente utilizado en sistemas comerciales, especialmente en plataformas de venta online, logística y gestión de órdenes. La relación entre un pedido y su envío es clara, natural y permite modelar

procesos completos de negocio que involucran estados, costos, fechas relevantes y trazabilidad (tracking).

Además, este dominio permite aplicar adecuadamente los conceptos solicitados en el trabajo:

- Relación 1 -> 1 estricta, donde el Envío depende del Pedido y no puede existir sin el.
- Reglas de negocio claras, como impedir la creación de más de un Envío para un mismo Pedido.
- Campos relevantes para validación, incluyendo estados (ENUM), tracking único y fechas obligatorios
- Operaciones transaccionales reales, como crear primero el Envío y luego confirmar la creación del Pedido, garantizando atomicidad.

Asimismo, el modelado Pedido -> Envío facilita la implementación del patrón DAO y Service, ya que ambas entidades tienen comportamientos diferenciados pero complementarios. El dominio también potencia el uso del menú de consola para CRUD, búsquedas y operaciones con rollback, permitiendo demostrar de forma clara el funcionamiento de la capa de persistencia.

En síntesis, la elección del dominio se justifica porque combina simplicidad conceptual con una estructura rica en reglas de negocio, validaciones y consistencia transaccional.

Diseño.

La relación entre las entidades Pedido y Envío se definió como uno a uno, dado que cada pedido puede tener un único envío asociado. Para implementar esta relación, se optó por utilizar una clave foránea única en la tabla envío (pedido_id), en lugar de compartir la clave primaria. Esta decisión permite que la entidad Envío mantenga su propia clave primaria independiente, otorgando mayor flexibilidad para futuras extensiones del modelo. La restricción UNIQUE sobre la FK asegura la cardinalidad 1->1, mientras que la cláusula ON DELETE CASCADE preserva la integridad referencial al eliminar automáticamente el envío cuando se elimina el pedido correspondiente. El diagrama UML refleja esta decisión mostrando la asociación 1..1 entre ambas clases y la ubicación de la FK en la clase Envío.

Arquitectura del Sistema y Funcionalidad de las Capas.

El sistema desarrollado se basa en una arquitectura por capas, que permite separar responsabilidades y garantizar la modularidad del código. Cada capa cumple un rol específico dentro del flujo de ejecución, interactuando con las demás para ofrecer una solución completa y escalable. A continuación, se describen las capas implementadas:

1. Capa de Entidades (Entities).

Esta capa define el **modelo de datos** del sistema, representando los objetos del dominio **Pedido → Envío**. Las clases como Pedido, Envío, EmpresaEnvío, EstadoPedido, EstadoEnvío y TipoEnvío encapsulan atributos y comportamientos propios de cada

entidad.

Función principal: Proporcionar una estructura clara para los datos que se manipulan en el sistema, asegurando la coherencia y el uso de principios de POO como encapsulamiento.

2. Capa de Acceso a Datos (DAO).

El paquete **DAO** contiene las clases responsables de la **interacción con la base de datos**. A través de clases como EnvioDao, PedidoDao y GenericDao, se implementan operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre las entidades.

Función principal: Abstraer la lógica de persistencia, evitando que las capas superiores dependan directamente de la base de datos.

3. Capa de Servicios (Service).

La capa de servicios implementa la **lógica de negocio** del sistema. Clases como EnvioService, PedidoService y GenericService coordinan las operaciones entre la capa DAO y la capa de presentación, aplicando reglas específicas del dominio.

Función principal: Garantizar que las operaciones sobre pedidos y envíos se realicen de acuerdo con las reglas del negocio, centralizando la lógica y evitando duplicación de código.

4. Capa de Presentación (Main).

Esta capa incluye las clases AppMenu y TPINTEGRADOR, que permiten la **interacción con el usuario**. A través de menús y opciones, se gestionan las solicitudes del usuario y se invocan los métodos de la capa de servicios.

Función principal: Ofrecer una interfaz sencilla para ejecutar las funcionalidades del sistema, integrando todas las capas anteriores.

Funcionamiento en Conjunto.

El flujo del sistema sigue el principio de **responsabilidad única y acoplamiento débil**:

- El **usuario** interactúa con la **capa de presentación**, que recibe las solicitudes.
- La **capa de servicios** procesa la lógica y delega las operaciones de persistencia a la **capa DAO**.
- La **capa DAO** accede a la base de datos y devuelve los resultados.
- Las **entidades** actúan como contenedores de datos que viajan entre las capas.
-

Este diseño favorece la **mantenibilidad**, la **escalabilidad** y la **reutilización del código**, cumpliendo con los principios de la arquitectura en capas.

Persistencia: estructura de la base, orden de operaciones y transacciones (commit/rollback)

La persistencia se implementó utilizando MySQL, JDBC y el patrón DAO. Se diseño una estructura relacional que respeta estrictamente la relación 1 -> 1 entre Pedido (A) y Envío (B), garantizando integridad referencial mediante una clave foránea única en la tabla envío.

Estructura de la base de datos

Tabla *pedido*

- *Id* BIGINT PK
- *numero* VARCHAR UNIQUE
- *fecha* DATE
- *cliente_nombre* VARCHAR
- *total* DECIMAL
- *estado* ENUM
- *eliminado* BOOLEAN

Tabla *envio*

- *id* BIGINT PK
- *tracking* VARCHAR UNIQUE
- *empresa* ENUM
- *tipo* ENUM
- *costo* DECIMAL
- *fecha_despacho* DATE
- *fecha_estimada* DATE
- *estado* ENUM
- *eliminado* BOOLEAN
- *pedido_id* BIGINT UNIQUE, FK -> pedido(id)

La presencia de *pedido_id* como clave foránea única garantiza que:

- Un Pedido puede tener 0 o 1 Envíos.
- Un Envío solo puede pertenecer a un Pedido.
- No pueden asociarse múltiples envíos al mismo pedido (unicidad + FK combinadas).

Implementación de los DAO

Los DAO (PedidoDAO y EnvioDAO) usan:

- PreparedStatement en todas las operaciones
- Métodos CRUD completos
- Conexiones externas para operaciones transaccionales

El mapeo ResultSet -> Entidad se realiza manualmente, cumpliendo con el paradigma JDBC sin ORM.

Orden de operaciones en una transacción típica

Las transacciones se gestionan en la capa Service (PedidoService o EnvioService), aunque el patrón queda claro a partir del uso en DAO y el diseño del proyecto.

Un flujo transaccional típico para crear un Pedido con su Envío sigue este orden:

1. Abrir la conexión desde la capa Service.

2. Desactivar el auto-commit
`conn.setAutoCommit(false);`
3. Crear primero el Envío usando `EnvioDao.create(envio, conn)`
 - Esto es necesario ya que el Envío puede requerir validación de unicidad del tracking.
4. Crear el Pedido usando `PedidoDao.create(pedido, conn)`.
5. Asociar el Envío al Pedido actualizando `pedido_id` en la tabla `envio` si corresponde.
6. Si todas las operaciones finalizan sin errores:
`conn.commit();`
7. En caso de cualquier excepción SQL o validación fallida:
`conn.rollback();`
8. Finalmente, restaurar el modo por defecto:
`conn.setAutoCommit(true);`

Este flujo asegura atomicidad: si falla la creación del Envío o del Pedido, ninguno de los dos queda registrado en la base, evitando incosistencias o relaciones incompletas.

Casos donde se aplica rollback

Se realiza rollback cuando ocurre una de las siguientes situaciones:

- Intento de asignar un Envío a un Pedido que ya tiene uno (violación 1→1).
- Violación de la unicidad del número de pedido o del tracking.
- Fallo en la inserción de cualquiera de las dos entidades.
- Error de validación (fechas inválidas, estados incompatibles, etc.).
- Excepción en la capa DAO durante cualquier operación SQL.

Esto garantiza que el estado de la base de datos siempre se mantenga consistente.

Ventajas del enfoque implementado

1. Asegura **integridad referencial** fuerte.
2. Evita Envíos huérfanos.
3. Simplifica el CRUD porque cada entidad tiene responsabilidad aislada.
4. El manejo transaccional evita registros incompletos.
5. El uso de Connection externa hace que DAO y Service trabajen como una única unidad atómica.

Validaciones y reglas de negocio.

En esta sección se detallan las principales validaciones y reglas de negocio implementadas para las entidades pedido y envío dentro del sistema.

El objetivo es garantizar la consistencia de los datos, evitar operaciones inválidas y respetar la relación 1→1 definida entre ambas tablas. Para ello se combinan restricciones a nivel de base de datos (tipos, claves foráneas, unicidad), controles adicionales en la lógica de aplicación (coherencia de estados, fechas e importes) y el uso de baja lógica mediante el campo eliminado, asegurando un comportamiento predecible y seguro en todas las operaciones del modelo.

Validaciones sobre la tabla pedido a nivel BD

- **numero** es obligatorio (NOT NULL), único (UNIQUE), por lo que no puede haber dos pedidos con el mismo número.
- **fecha** es obligatoria.
- **cliente_nombre** es obligatorio, con longitud máxima 120 caracteres.
- **total** es obligatorio y se almacena con precisión de 2 decimales.
- **estado** es obligatorio y sólo admite los valores del enum ('NUEVO', 'FACTURADO', 'ENVIADO').
-

Reglas de negocio sobre Pedido (nivel aplicación)

- **total** debe ser **mayor a 0** antes de insertar o actualizar el registro.
- La fecha del pedido no debe ser posterior a la fecha actual del sistema.
- **cliente_nombre** no debe contener sólo espacios en blanco.
- No se permite cambiar el estado hacia atrás (por ejemplo, de ENVIADO a NUEVO).
- El campo **eliminado** implementa baja lógica:
- **Pedidos con eliminado = TRUE** no se muestran en listados “activos”.
- No se permite modificarlos ni asociarles nuevos envíos.

Validaciones sobre la tabla envío a nivel BD

- **tracking** es: Obligatorio (NOT NULL), único (UNIQUE), no se repiten códigos de seguimiento.
- **empresa** es obligatorio y sólo admite ('ANDREANI', 'OCA', 'CORREO_ARG').
- **tipo** es obligatorio y sólo admite ('ESTANDAR', 'EXPRES').
- **costo** es obligatorio y se almacena con 2 decimales.
- **estado** sólo admite los valores del enum ('EN_PREPARACION', 'EN_TRANSITO', 'ENTREGADO').
- **pedido_id**:
- Es único (UNIQUE), garantizando **como máximo un envío por pedido**.
- Es clave foránea a pedido(id) con ON DELETE CASCADE, de modo que, si un pedido se borra físicamente, su envío asociado también se elimina.
-

Reglas de negocio sobre Envío a nivel Aplicación.

- **costo** debe ser mayor o igual a 0.
- **estado** se trata como obligatorio en la lógica de negocio, aunque en la tabla pueda ser NULL; la aplicación no permite guardar un envío sin estado.
- Coherencia de fechas:
- **fecha_despacho** no puede ser anterior a la fecha del pedido asociado.
- Si se informa **fecha_estimada**, debe ser igual o posterior a **fecha_despacho**.
- Un envío con estado = 'ENTREGADO' no puede volver a EN_TRANSITO ni EN_PREPARACION.
- **eliminado** funciona como baja lógica:

- **Envíos con eliminado = TRUE** no se listan como activos.
- No se permite seguir modificando su estado.
-

Reglas de negocio de la relación Pedido → Envío (1 → 1)

- La restricción UNIQUE sobre envio_pedido_id garantiza que **cada pedido tenga a lo sumo un envío**.
- Antes de crear un nuevo envío, la capa de servicio verifica que el **pedido_id** elegido exista en la tabla pedido y no tenga ya un envío asociado.
- Para poder marcar un **pedido.estado** = 'ENVIADO' se exige que el pedido tenga un **envío** asociado y que el **envío.estado** sea al menos 'EN_TRANSITO' (o 'ENTREGADO', según la política definida).
- No se permite asociar un envío a un pedido con eliminado = TRUE.

Coherencia de los datos de ejemplo

Los datos de prueba insertados respetan las reglas anteriores:

```
INSERT INTO pedido (...)  
VALUES  
(FALSE, 'PED001', '2025-11-01', 'Encarnación Ezcurra', 15000.00, 'NUEVO'),  
(FALSE, 'PED002', '2025-11-02', 'Juan Manuel de Rosas', 23000.50, 'FACTURADO'),  
(FALSE, 'PED003', '2025-11-03', 'Carlos Moreno', 8700.75, 'ENVIADO');
```

```
INSERT INTO envío (...)  
VALUES  
(FALSE, 'TRK001', 'ANDREANI', 'ESTANDAR', 500.00, '2025-11-02', '2025-11-05',  
'EN_PREPARACION', 1),  
(FALSE, 'TRK002', 'OCA', 'EXPRES', 750.00, '2025-11-03', '2025-11-04', 'EN_TRANSITO', 2),  
(FALSE, 'TRK003', 'CORREO_ARG', 'ESTANDAR', 600.00, '2025-11-04', '2025-11-07',  
'ENTREGADO', 3);
```

- Cada pedido tiene a lo sumo un envío (relación 1 → 1).
- Las fechas de despacho son posteriores a las fechas de los pedidos.
- Los estados de envío son coherentes con los estados de pedido (por ejemplo, PED003 está ENVIADO y su envío TRK003 se encuentra ENTREGADO).

Pruebas realizadas.

Menú.

```
Output - TP-INTEGRADOR (run)
run:

    === MENÚ PRINCIPAL ===
    1. Crear Pedido + Envío
    2. Listar Pedidos
    3. Listar Envíos
    4. Eliminar Pedido
    5. Eliminar Envío
    0. Salir

    Seleccione una opción: 2

    === LISTA DE PEDIDOS ===
    Pedido{ id= 1, Número = 'PED001', Fecha = 2025-11-01, Nombre Completo = 'Encarnación Ezcurra', Total = 15000.0, Estado = NUEVO, Envío = No tiene}
    Pedido{ id= 2, Número = 'PED002', Fecha = 2025-11-02, Nombre Completo = 'Juan Manuel de Rosas', Total = 23000.5, Estado = FACTURADO, Envío = No tiene}
    Pedido{ id= 3, Número = 'PED003', Fecha = 2025-11-03, Nombre Completo = 'Carlos Moreno', Total = 8700.75, Estado = ENVIADO, Envío = No tiene}
    Pedido{ id= 5, Número = '4', Fecha = 2025-11-13, Nombre Completo = 'Julio Sosa', Total = 1.23, Estado = NUEVO, Envío = No tiene}

    === MENÚ PRINCIPAL ===
    1. Crear Pedido + Envío
    2. Listar Pedidos
    3. Listar Envíos
    4. Eliminar Pedido
    5. Eliminar Envío
    0. Salir

    Seleccione una opción: 3

    === LISTA DE ENVÍOS ===
    Envío{ id=1, Tracking= 'TRK001', Empresa= ANDREANI, Tipo= ESTANDAR, Costo= 500.0, Fecha de despacho= 2025-11-02, Fecha estimada de entrega= 2025-11-05, Estado= EN_PREPARACION}
    Envío{ id=2, Tracking= 'TRK002', Empresa= OCA, Tipo= EXPRES, Costo= 750.0, Fecha de despacho= 2025-11-03, Fecha estimada de entrega= 2025-11-04, Estado= EN_TRANSITO}
    Envío{ id=3, Tracking= 'TRK003', Empresa= CORREO_ARG, Tipo= ESTANDAR, Costo= 600.0, Fecha de despacho= 2025-11-04, Fecha estimada de entrega= 2025-11-07, Estado= ENTREGADO}
    Envío{ id=4, Tracking= '6789', Empresa= OCA, Tipo= EXPRES, Costo= 345.0, Fecha de despacho= 2025-11-13, Fecha estimada de entrega= 2025-11-14, Estado= EN_PREPARACION}
    Envío{ id=5, Tracking= '002', Empresa= OCA, Tipo= EXPRES, Costo= 567.0, Fecha de despacho= 2025-11-13, Fecha estimada de entrega= 2025-11-15, Estado= EN_PREPARACION}

    === MENÚ PRINCIPAL ===
    1. Crear Pedido + Envío
    2. Listar Pedidos
    3. Listar Envíos
    4. Eliminar Pedido
    5. Eliminar Envío
    0. Salir

    Seleccione una opción: 0
    Saliendo...
BUILD SUCCESSFUL (total time: 1 minute 6 seconds)

```

En la captura podemos observar:

- _ Listado de los pedidos, insertados a través del SCRIPT y uno creado de forma manual.
(se ve la ausencia de **id=4**, debido a que fue creado y eliminado de prueba pero sin registro en captura).
- _ Listado de los envíos, insertados a través del SCRIPT

Consultas SQL útiles.

Algunas consultas que serán de utilidad en el funcionamiento del sistema.

Listar pedidos.

```
5  -- Listar pedidos
6  •  SELECT p.numero AS NumeroPedido, p.fecha AS FechaPedido, p.cliente_nombre AS Cliente,
7      p.total AS TotalPedido, p.estado AS EstadoPedido,
8      e.tracking AS TrackingEnvío, e.empresa AS EmpresaEnvío, e.tipo AS TipoEnvío, e.estado AS EstadoEnvío
9  FROM pedido p
10 LEFT JOIN envío e ON p.id = e.pedido_id;
```

Result Grid									
	NumeroPedido	FechaPedido	Cliente	TotalPedido	EstadoPedido	TrackingEnvío	EmpresaEnvío	TipoEnvío	EstadoEnvío
▶	PED001	2025-11-01	Encarnación Ezcurra	15000.00	NUEVO	TRK001	ANDREANI	ESTANDAR	EN_PREPARACION
	PED002	2025-11-02	Juan Manuel de Rosas	23000.50	FACTURADO	TRK002	OCA	EXPRES	EN_TRANSITO
	PED003	2025-11-03	Carlos Moreno	8700.75	ENVIADO	TRK003	CORREO_ARG	ESTANDAR	ENTREGADO
	1	2025-11-12	Maximus	12.35	NUEVO	6789	OCA	EXPRES	EN_PREPARACION
	4	2025-11-13	Julio Sosa	1.23	NUEVO	002	OCA	EXPRES	EN_PREPARACION

Costo de envío por empresa.

```
12      -- Costo de envío por empresa
13 •  SELECT empresa AS EmpresaEnvio, SUM(costo) AS TotalCostoEnvios
14    FROM envio
15   GROUP BY empresa;
```

Result Grid	
EmpresaEnvio	TotalCostoEnvios
ANDREANI	500.00
OCA	1662.00
CORREO_ARG	600.00

Total facturado del mes.

```
17      -- Total Facturado por mes
18 •  SELECT DATE_FORMAT(fecha, '%Y-%m') AS Mes, SUM(total) AS TotalFacturado
19    FROM pedido
20   GROUP BY DATE_FORMAT(fecha, '%Y-%m')
21  ORDER BY Mes;
22
```

Result Grid	
Mes	TotalFacturado
2025-11	46714.83

Conclusiones.

La aplicación desarrollada permite gestionar de forma integrada los pedidos y sus envíos, utilizando Java con acceso a una base de datos MySQL a través de JDBC. Se logró implementar un modelo de datos coherente y normalizado, con entidades diferenciadas para pedidos, envíos, empresas de envío, tipos y estados, además de un mecanismo de borrado lógico que evita la pérdida definitiva de información. La arquitectura en capas (entidades, DAOs, servicios y capa de presentación) favorece la separación de responsabilidades, facilita el mantenimiento del código y permite reutilizar lógica de negocio en distintos puntos del sistema. Asimismo, el uso de transacciones al momento de crear pedidos y envíos asociados garantiza la consistencia de los datos, ya que las operaciones se confirman o revierten de manera conjunta ante posibles errores. En conjunto, estas decisiones de diseño muestran una correcta aplicación de los conceptos trabajados en la materia: programación orientada a objetos, acceso a datos con JDBC, manejo de transacciones y diseño de aplicaciones por capas, cumpliendo satisfactoriamente con los objetivos planteados para el trabajo práctico integrador.

Mejoras futuras y posibles extensiones.

De cara al futuro, la aplicación podría ampliarse incorporando funcionalidades orientadas al usuario, como búsquedas y filtros avanzados de pedidos por fechas, estado o cliente, y de envíos por empresa, tipo o estado. También sería útil generar reportes y estadísticas simples (por ejemplo, total facturado por período, cantidad de pedidos por estado o costo total de envíos por empresa), así como reforzar ciertas reglas de negocio, impidiendo incoherencias como marcar un pedido como enviado sin un envío asociado o no actualizar el estado del pedido cuando el envío se entrega. En el plano técnico, una

línea de mejora importante sería extraer la configuración de la base de datos a archivos externos o variables de entorno, para evitar credenciales “hardcodeadas” y facilitar el despliegue en distintos entornos. También resultaría conveniente sumar validaciones previas a la persistencia de datos, mejorar el manejo de errores diferenciando mensajes para el usuario y para el log interno, e incorporar pruebas unitarias y de integración sobre la capa de servicios. Finalmente, a medida que la aplicación creciera, podría evaluarse una evolución hacia una interfaz gráfica de escritorio o una versión web, agregar índices y auditoría en la base de datos, e incluso implementar mecanismos de seguridad más avanzados (como roles de usuario y control de permisos), lo que permitiría escalar el sistema a contextos de uso más reales y exigentes.

Fuentes y herramientas.

Herramientas de desarrollo

- **Lenguaje:** Java (versión utilizada en la cursada).
- **IDE:** Apache NetBeans para la implementación de las clases Pedido, Envio y los enumerados (EstadoPedido, EstadoEnvio, TipoEnvio, EmpresaEnvio).
- **Base de datos:** MySQL, donde se crearon las tablas pedido y envio.
- **Entorno de ejecución de SQL:** consola de MySQL y/o herramienta gráfica (por ejemplo, MySQL Workbench) para ejecutar los scripts:
 - CREATE DATABASE tp_integrador;
 - Definición de tablas y claves foráneas.
 - Inserts de datos de prueba.

Material teórico y documentación

- Apuntes, guías y enunciado del **Trabajo Final Integrador – Programación 2** de la UTN.
- Documentación oficial de **Java** (Oracle) para manejo de:
 - <https://docs.oracle.com/en/java/>
 - Tipos de datos.
 - Enumeraciones.
 - Manejo de excepciones y validaciones en la capa de servicio.
- Documentación oficial de **MySQL** para:
 - <https://dev.mysql.com/doc/>
 - Syntaxis de CREATE TABLE, ENUM, FOREIGN KEY, UNIQUE.
 - Restricciones de integridad referencial y comportamiento ON DELETE CASCADE.

Herramientas de apoyo

- **ChatGPT/Copilot/Google Gemini:** utilizado como apoyo para:
 - Definir y redactar las **validaciones y reglas de negocio** del modelo Pedido–Envío en función del esquema SQL real.
 - Mejorar la redacción de la sección de “Fuentes y herramientas utilizadas”.