INFORME

Descripción general del proyecto

Este proyecto implementa un sistema de consola para gestionar Pedidos y sus Envíos asociados. Está desarrollado en Java con una relación 1→1 unidireccional (Pedido referencia a Envío), persistencia mediante JDBC y patrón DAO sobre MySQL/MariaDB, y manejo de transacciones (commit/rollback). La aplicación ofrece operaciones CRUD, búsquedas y validaciones en capa de servicio.

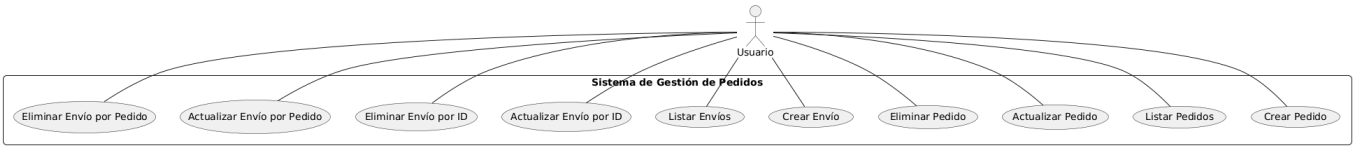
- Arquitectura en capas: UI (consola) → Service → DAO → Models → Base de datos
- Tecnologías: Java 17+, JDBC (PreparedStatement), MySQL/MariaDB
- Configuración: archivo `db.properties` y script SQL en `SQL BBD/Script.sql`

Para una guía ampliada de instalación y uso, revisar `README.md` en la raíz del proyecto.

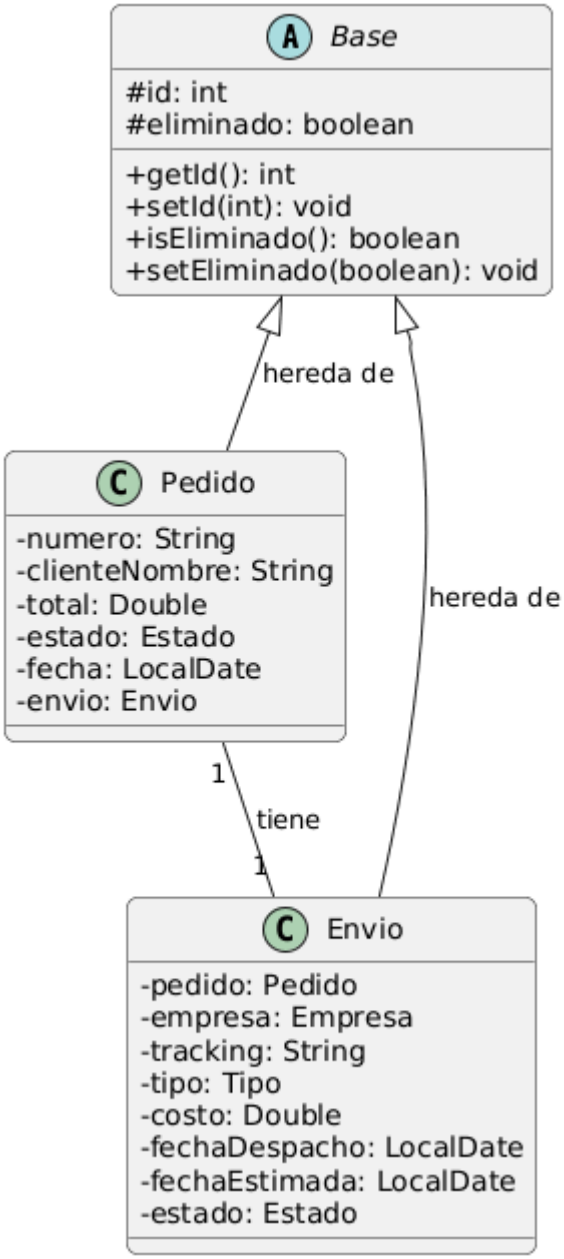
Archivos dentro de la carpeta UML

La carpeta `UML` contiene los diagramas/imágenes de apoyo al diseño del sistema:

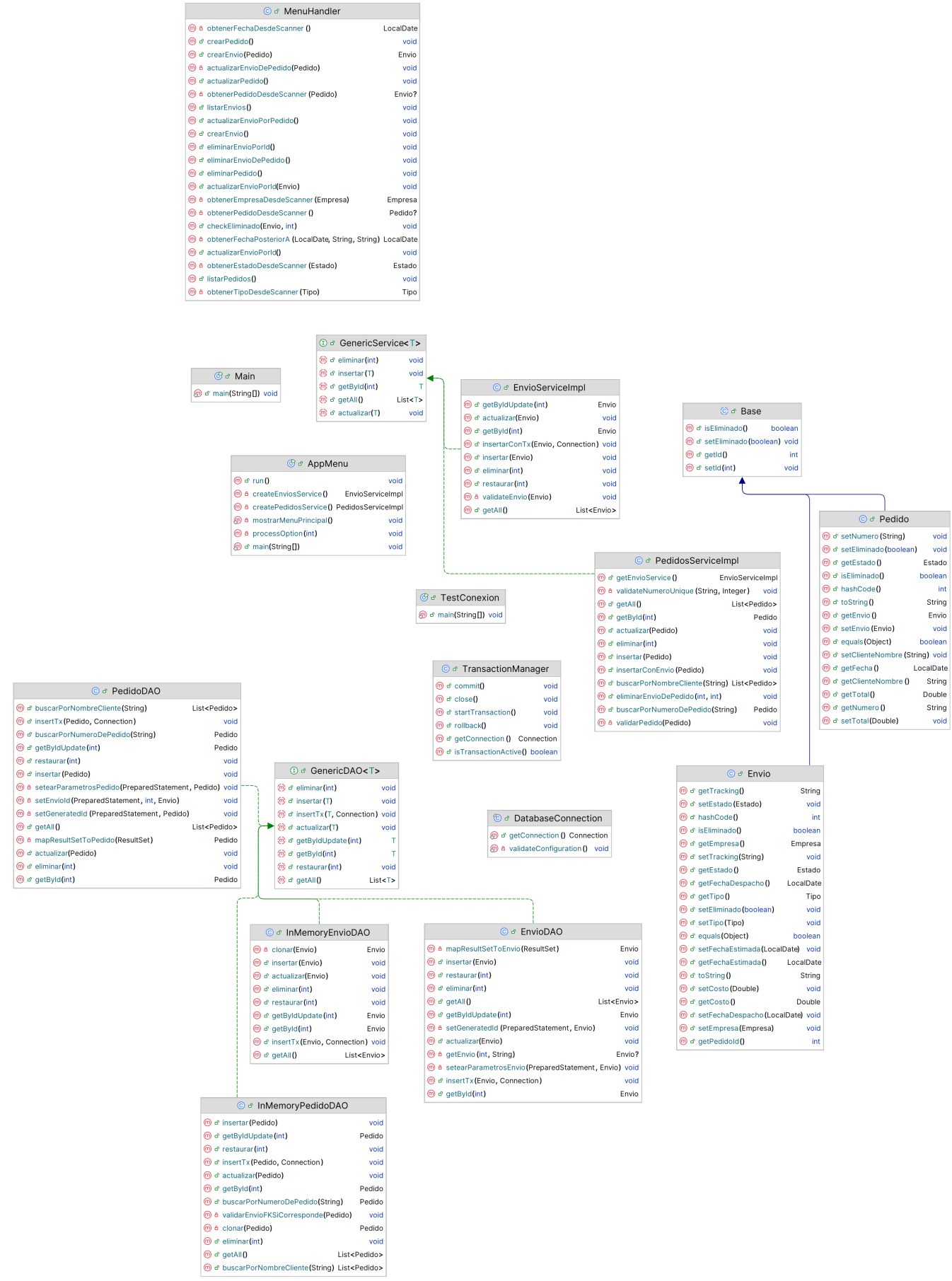
UML Casos de uso



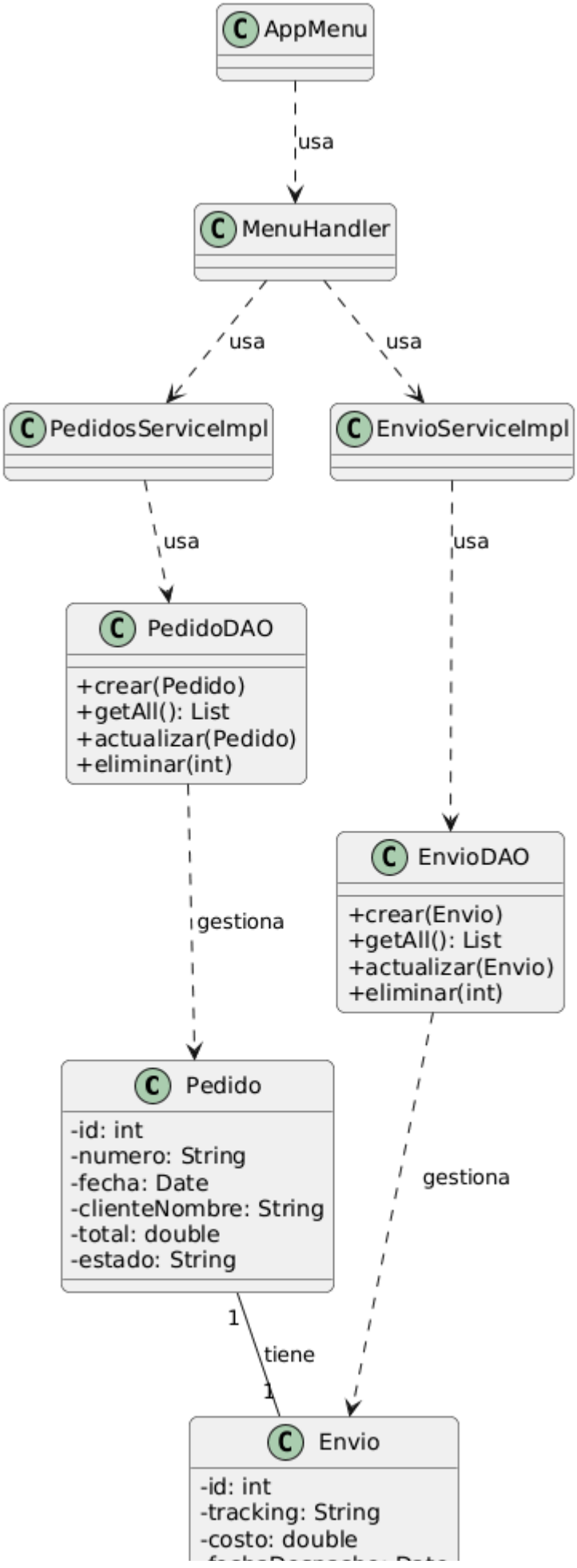
UML Modelos

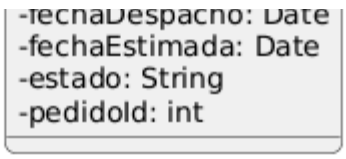


UML Completo, con métodos



UML de clases, simplificado


AppMenu
usa
MenuHandler
usa
usa
PedidosServiceImpl
EnvioServiceImpl
usa
usa
PedidoDAO
+crear(Pedido)
+getAll(): List
+actualizar(Pedido)
+eliminar(int)
gestiona
EnvioDAO
+crear(Envio)
+getAll(): List
+actualizar(Envio)
+eliminar(int)
Pedido
-id: int
-numero: String
-fecha: Date
-clienteNombre: String
-total: double
-estado: String
gestiona
1
tiene
1
Envio
-id: int
-tracking: String
-costo: double



Estos recursos ilustran la estructura de clases, la relación entre entidades y vistas generales del modelo.

Arquitectura en Capas

- Objetivo: separar responsabilidades, reducir acoplamiento y facilitar pruebas y mantenimiento.
- Capas utilizadas:
  - Presentación (UI por consola)
    - Qué hace: interacción con el usuario, lectura de opciones y datos de entrada.
    - Qué NO hace: no contiene reglas de negocio ni SQL.
    - Dónde está: `src\Main\AppMenu.java` , `src\Main\MenuHandler.java` , `src\Main\Main.java` .
  - Servicio (Service)
    - Qué hace: valida datos, orquesta operaciones compuestas, define límites transaccionales.
    - Qué NO hace: no conoce detalles de SQL ni de la conexión.
    - Dónde está: `src\Service\PedidosServiceImpl.java` , `src\Service\EnvioServiceImpl.java` .
  - Acceso a Datos (DAO)
    - Qué hace: CRUD con JDBC, mapea ResultSet ↔ modelos, usa `PreparedStatement` .
    - Qué NO hace: no valida reglas de negocio propias del dominio.
    - Dónde está: `src\Dao\PedidoDAO.java` , `src\Dao\EnvioDAO.java` , `src\Dao\GenericDAO.java` .
  - Modelo (Domain Models)
    - Qué hace: representa entidades y su estado; incluye enumeraciones y lógica mínima de consistencia.
    - Dónde está: `src\Models\Pedido.java` , `src\Models\Envio.java` , `src\Models\Base.java` .
  - Infraestructura
    - Qué hace: provee servicios técnicos (conexionado y transacciones).

        - Dónde está: `src\Config\DatabaseConnection.java`, `src\Config\TransactionManager.java`.
- Principios aplicados
    - Dependencias “hacia abajo”: UI → Service → DAO → Infraestructura. Los modelos son compartidos.
    - Sustitución de implementaciones: la capa Service depende de interfaces/DAOs concretos, pudiendo usarse DAOs en memoria (`InMemory*DAO`) para pruebas.
    - Beneficios: testeo aislado, menor acoplamiento, evolución independiente de UI/BD.
- Buenas prácticas adoptadas
    - Validaciones en Service para mantener los DAOs simples y reutilizables.
    - DAOs con `PreparedStatement` para evitar SQL Injection y parametrizar consultas.
    - Métodos cortos y específicos por responsabilidad (SRP).

Uso de Transacciones

- Conceptos clave
    - ACID:
        - Atomicidad: una operación compuesta se confirma toda o se revierte toda.
        - Consistencia: al finalizar, los datos respetan reglas y restricciones (FK, unicidad).
        - Aislamiento: transacciones concurrentes no se interfieren de manera incorrecta.
        - Durabilidad: una vez confirmado (commit), el cambio persiste ante fallas.
    - Autocommit: por defecto los drivers confirman cada sentencia; para operaciones compuestas se desactiva.
- Niveles de aislamiento (resumen)
    - READ COMMITTED: evita lecturas sucias; suele ser un buen equilibrio por defecto.
    - REPEATABLE READ: evita lecturas no repetibles; puede ser el default en MySQL/InnoDB.
    - SERIALIZABLE: máximo aislamiento con mayor costo de bloqueo.
    - Nota: el nivel puede ajustarse en la conexión si fuera necesario según la operación.
- Dónde y cómo se definen los límites transaccionales en este proyecto
    - La capa Service es la responsable de abrir/cerrar transacciones cuando coordina múltiples DAOs.
    - Clases relevantes:
        - `Config.DatabaseConnection`: fábrica de conexiones JDBC (lee `db.properties`).

    - `Config.TransactionManager` : encapsula `setAutoCommit(false)` , `commit()` y `rollback()` y asegura cierre seguro.
- Flujo típico de una operación compuesta
    i. Obtener conexión y crear `TransactionManager` .
    ii. `startTransaction()` para desactivar autocommit.
    iii. Ejecutar varios métodos DAO (insert/update/delete) coherentes entre sí.
    iv. Si todo sale bien, `commit()` ; si hay error, `rollback()` .
    v. En el `close()` se restaura `autoCommit=true` y se cierra la conexión.
- Ejemplo simplificado (pseudocódigo basado en `PedidosServiceImpl` / `EnvioServiceImpl`)

```
try (Connection conn = DatabaseConnection.getConnection();
     TransactionManager tx = new TransactionManager(conn)) {
    tx.startTransaction();

    // Validaciones de negocio previas
    // ...

    // Operaciones coordinadas
    pedidoDAO.insert(conn, pedido);
    if (pedido.tieneEnvio()) {
        envioDAO.insert(conn, pedido.getEnvio());
    }

    tx.commit();
} catch (Exception e) {
    // El TransactionManager hará rollback en close() si la transacción quedó activa
    throw e; // Propagar o transformar la excepción según la política de Service
}
```

- Errores comunes que se evitan con este enfoque
    - “Commit parcial”: confirmación de un DAO y fallo en otro → se evita al desactivar autocommit y confirmar al final.
    - Rollback olvidado: `TransactionManager.close()` hace `rollback()` si quedó activo.