PROGRAMACIÓN II
Trabajo Práctico Integrador

Integrantes:

Campana, Mario

Chiavón, Cristian

Chiavón, Facundo

Kohn, Shai

Repositorio GitHub: https://github.com/Farvon/UTN-TUPaD-P2-TPI

Enlace al video: https://drive.google.com/file/d/1iVQu0hwD831VUpfBHG_ovGSuK-kwDMLn/view?usp=drive_link

Integrantes y Roles

Integrante	Rol Principal	Responsabilidades Específicas
Facundo Chiavón	**Líder Técnico/Service**	Diseño de la arquitectura por capas, implementación de la lógica de negocio (EmpresaServiceImpl), gestión del flujo transaccional.
Mario Campana	**Persistencia/DAO**	Implementación de las clases DAO (EmpresaDao, DomicilioFiscalDao), construcción de las consultas SQL, mapeo de resultados y gestión de conexiones (DatabaseConnection).
Cristian Chiavón	**Entidades y Modelado**	Definición de las entidades (Empresa, DomicilioFiscal, Base), modelado de la relación 1:1, validaciones de entidad y diseño UML.
Shai Kohn	**Interfaz y Pruebas**	Desarrollo de la interfaz de consola (AppMenu, Main), pruebas unitarias del flujo CRUD y simulación del *rollback* transaccional.

Elección del Dominio y Justificación

El dominio elegido es la **Gestión de Empresas y sus Domicilios Fiscales**.

- **Justificación académica.** Este dominio se seleccionó porque refleja una necesidad clara de modelar una relación **Uno a Uno (1→1) unidireccional** entre dos entidades del mundo real: una empresa y su domicilio fiscal. A nivel conceptual, cada empresa tiene un domicilio fiscal único para fines legales e impositivos; a nivel de implementación, se permite que una empresa se cree inicialmente **sin domicilio** y que este se asocie luego (modelo 1 → 0..1), manteniendo la unicidad del vínculo mediante una clave foránea única.

Relevancia práctica. Este tipo de modelo aparece en:

- Sistemas de registro gubernamental (AFIP, registros públicos).
- Módulos de facturación y contabilidad de ERPs.
- Plataformas de gestión empresarial donde los datos fiscales deben ser consistentes y únicos.

Trabajar con este dominio nos obliga a cuidar:

- **Integridad referencial** (una empresa no debe apuntar a un domicilio inexistente).
- **Unicidad** (un domicilio fiscal no puede pertenecer a más de una empresa).
- **Transaccionalidad** (la creación de la empresa y su domicilio se trata como una operación atómica cuando se hacen juntas).

Diseño y Decisiones Clave de Modelado

A. Modelado de la Relación 1:1

La relación principal es entre la entidad **Empresa** y **DomicilioFiscal**.

En la base de datos (MySQL).

- Se utilizó el enfoque de **Clave Foránea Única** sobre la tabla Empresas.
- La columna Empresas.domicilioFiscal_id:
    - Es FOREIGN KEY a Domicilios.id.
    - Tiene restricción UNIQUE.
    - Usa ON UPDATE CASCADE y ON DELETE SET NULL.

Esto garantiza que:

1. Una empresa puede apuntar **a lo sumo** a un domicilio fiscal (0..1).
2. Un domicilio fiscal no puede estar asociado a más de una empresa (unicidad de la relación).

En el código Java.

- La relación se modela como **unidireccional** desde Empresa hacia DomicilioFiscal.
- En Empresa.java se define:

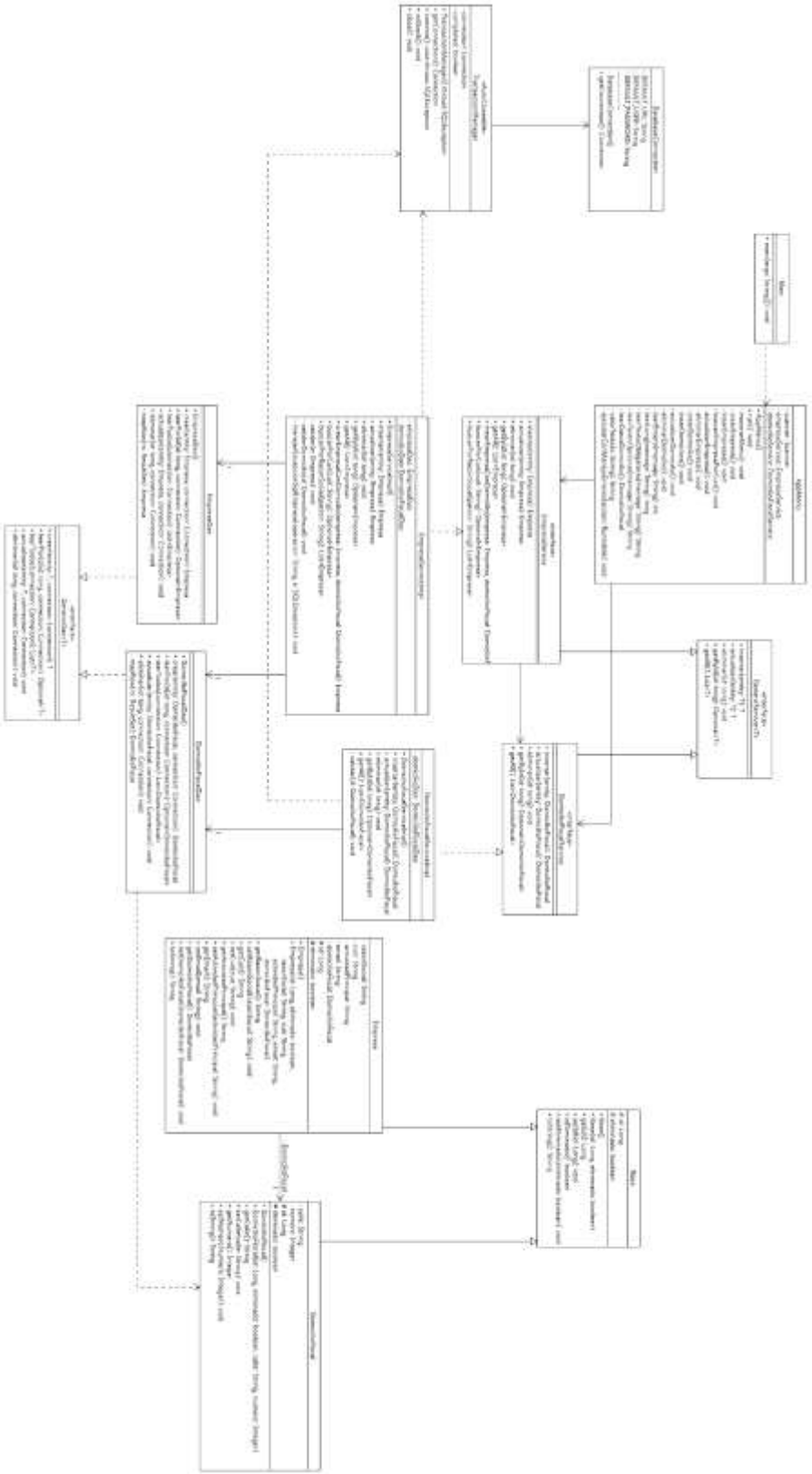
```
private DomicilioFiscal domicilioFiscal;
```

- DomicilioFiscal no conoce a Empresa, lo que coincide con la consigna de **1→1 unidireccional**.

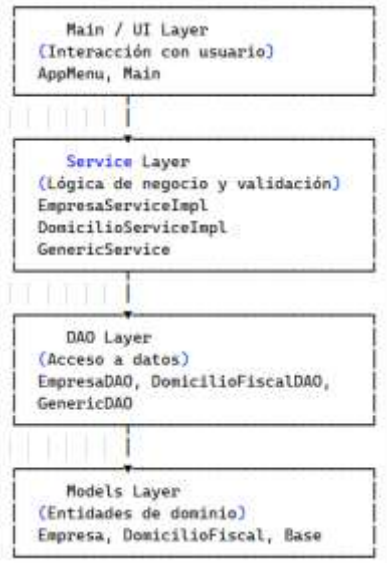
B. Diagrama de Clases UML

El diagrama de clases muestra:

- La herencia de Empresa y DomicilioFiscal desde la clase abstracta Base (que aporta id y eliminado).
- La relación Empresa 1 —— 0..1 DomicilioFiscal unidireccional.
- La arquitectura por capas:
  - config (DatabaseConnection, TransactionManager).
  - entities (Base, Empresa, DomicilioFiscal).
  - dao (GenericDao<T>, EmpresaDao, DomicilioFiscalDao).
  - service (GenericService<T>, EmpresaService, DomicilioFiscalService, implementaciones).
  - main (Main, AppMenu).



Arquitectura por Capas (JDBC Nativo)



El proyecto se implementa siguiendo una arquitectura de tres capas (Presentación, Negocio, Persistencia), separadas en paquetes Java con responsabilidades claras:

Paquete	Responsabilidad (¿Qué hace?)	Archivos Clave
main	**Capa de Presentación**	**AppMenu.java**: Interfaz de consola para la interacción del usuario. Es el único que interactúa con la capa Service. Implementa el manejo genérico de errores (ejecutarConManejoErrores).
service	**Capa de Negocio y Transaccionalidad**	**EmpresaServiceImpl.java**: Contiene la lógica de negocio (validar) y **coordina** las operaciones de múltiples DAOs dentro de una transacción.
dao	**Capa de Persistencia**	**EmpresaDao.java**, **DomicilioFiscalDao.java**: Aísla la lógica SQL de las otras capas. Mapea ResultSet a entidades y viceversa. Los métodos son **agnósticos a la conexión** (requieren Connection connection).
models	**Capa de Dominio/Modelo**	**Empresa.java**, **DomicilioFiscal.java**, **Base.java**: Define la estructura de datos. Base implementa el ID y el concepto de **Baja Lógica** (eliminado).
config	**Capa de Configuración**	**TransactionManager.java**, DatabaseConnection.java: Gestiona recursos críticos.

Persistencia y Transacciones

A. Estructura de la Base y Baja Lógica

Todas las entidades (Empresa, DomicilioFiscal) extienden Base.java, lo que implica el uso del campo eliminado para la **Baja Lógica**. Esto significa que la operación eliminar en los DAOs es en realidad un UPDATE... SET eliminado = TRUE.

Las tablas principales son Empresas y Domicilios. Ambas incluyen: id BIGINT (PK, AUTO_INCREMENT) y eliminado BOOLEAN para **baja lógica**.

- La operación eliminar(id) en cada DAO no borra físicamente el registro, sino que ejecuta:

```
UPDATE Empresas   SET eliminado = TRUE WHERE id = ?;
UPDATE Domicilios SET eliminado = TRUE WHERE id = ?;
```

- Los métodos leerPorId y leerTodos filtran por eliminado = FALSE, de modo que los elementos dados de baja no aparecen en el menú.

TABLA EMPRESAS

id	eliminado	razonSocial	cuit	actividadPrincipal	email	domicilioFiscal_id
1	0	El trébol S.A.	30-12345678-9	Fabricación de explosivos	contacto@acme.com	1
2	0	Tech Solutions SRL	30-87654321-0	Servicios de software	info@techsolutions.com	2
3	0	Transporte Norte S.A.	30-11112222-3	Transporte de cargas	contacto@tnorte.com	
4	0	Panificados del Sur	30-99998888-7	Panadería industrial	ventas@panificados.com	3

TABLA DOMICILIOS

Result Grid | Filter Rows: | Edit:

id	eliminado	calle	numero
1	0	Av. Córdoba	742
2	0	San Martín	1234
3	0	Belgrano	567
4	0	Oncativo	1999
NULL	NULL	NULL	NULL

B. Transaccionalidad con TransactionManager

La clase TransactionManager encapsula una Connection y maneja el ciclo de vida de una transacción JDBC:

- En el constructor se obtiene una conexión desde DatabaseConnection.getConnection() y se pone setAutoCommit(false).
- Implementa AutoCloseable, lo que permite usarla con try-with-resources:

Java

```
try (TransactionManager tx = new TransactionManager()) {
    Connection conn = tx.getConnection();
    // Operaciones con DAOs...
    tx.commit();
}
```

Si commit() no se llama, el método close() hace rollback() automáticamente:

```
@Override
public void close() {
    try {
        if (!completed) {
            rollback();
        }
        connection.close();
    } catch (SQLException e) {
        // Manejo de errores
    }
}
```

Esto asegura que cualquier excepción (checked o unchecked) dentro del bloque provoca la **reversión de todos los cambios**.

C. Orden de Operaciones en crearEmpresaConDomicilio

La lógica transaccional principal está en EmpresaServiceImpl.crearEmpresaConDomicilio(...):

1. Se crean el TransactionManager y la Connection.
2. Si se recibió un DomicilioFiscal:
    - Se validan sus datos.
    - Se inserta con domicilioDao.crear(domicilioFiscal, conn).
    - Se asigna el domicilio creado a la empresa.
3. Se inserta la empresa con empresaDao.crear(empresa, conn).
4. Si todo es exitoso, se ejecuta tx.commit().

Si ocurre un error (por ejemplo, CUIT duplicado que viola la restricción UNIQUE), se lanza un SQLException con estado 23000. El TransactionManager realiza rollback() en close(), deshaciendo tanto el alta del domicilio como el intento de alta de la empresa.

Esto cumple con el requerimiento de operaciones atómicas: o se guardan ambos registros (empresa + domicilio) o no se guarda ninguno.

D. Implementación del Rollback

La robustez se garantiza en el método close() de TransactionManager:

Java

```
// En TransactionManager.java

@Override

public void close() {

    try {

        if (!completed) { // Si commit() no se llamó (hay excepción)

            rollback();   // Ejecuta el rollback

        }

        connection.close();

    } catch (SQLException e) {

        // ... manejo de errores

    }

}
```

Esto asegura que **cualquier SQLException o RuntimeException** dentro del bloque try resulta en un rollback() automático, deshaciendo los cambios parciales (ej. la creación del Domicilio que se realizó antes del fallo).

Validaciones y Reglas de Negocio

- Las validaciones se concentran en la capa service, antes de llamar a los DAOs.

Validaciones de campos obligatorios

En EmpresaServiceImpl:

- razonSocial es obligatoria (null o vacía lanza IllegalArgumentException).

- cuit es obligatorio.

En DomicilioFiscalServiceImpl:

- calle es obligatoria.

Regla de negocio: unicidad del CUIT

A nivel de base de datos:

- La columna Empresas.cuit tiene un UNIQUE.

A nivel de aplicación:

- EmpresaServiceImpl captura SQLException con SQLState "23000" y devuelve un mensaje específico indicando que el CUIT ya existe.

Este doble nivel de validación (BD + aplicación) refuerza la **integridad de los datos**.

Pruebas Realizadas

Se realizaron pruebas funcionales para validar CRUD completo de Empresa y DomicilioFiscal, Funcionamiento de la relación 1→1, Baja lógica y Rollback automático ante errores.

A. Prueba CRUD y 1:1

- **Acción:** Listado inicial de datos.
- **Resultado esperado:** Ejecutar la opción “Listar empresas” y “Listar domicilios” en el menú.
- **Consulta SQL útil:**

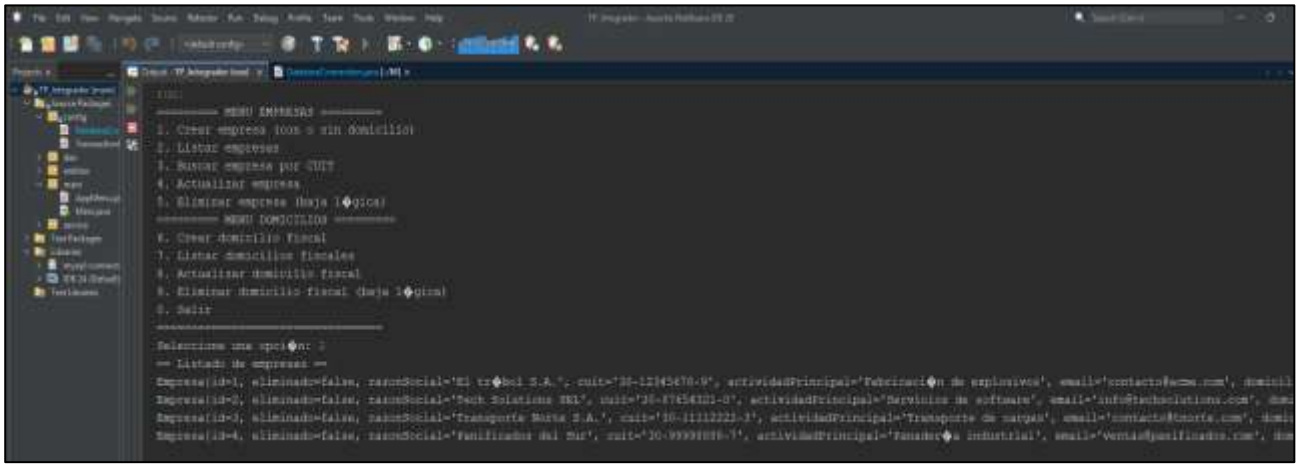
```
SQL

SELECT e.razonSocial, d.calle, e.domicilioFiscal_id

FROM Empresas e

LEFT JOIN Domicilios d ON e.domicilioFiscal_id = d.id

WHERE e.eliminado = FALSE;
```



```
==================================
Seleccione una opci□n: 7
== Listado de domicilios fiscales ==
DomicilioFiscal{id=1, eliminado=false, calle='Av. C□rdoba', numero=742}
DomicilioFiscal{id=2, eliminado=false, calle='San Mart□n', numero=1234}
DomicilioFiscal{id=3, eliminado=false, calle='Belgrano', numero=567}
DomicilioFiscal{id=4, eliminado=false, calle='Oncativo', numero=1999}
```

B. Creación de empresa con domicilio.

- **Acción:** Crear empresa (con domicilio), ingresando datos de empresa y eligiendo “S” para crear domicilio fiscal.

- **Resultado esperado:**
  - Se inserta una fila en Domicilios.
  - Se inserta una fila en Empresas con domicilioFiscal_id apuntando al nuevo domicilio.
- **Consulta SQL útil:**

```
SELECT e.razonSocial, e.cuit, d.calle, d.numero, e.domicilioFiscal_id
FROM Empresas e
LEFT JOIN Domicilios d ON e.domicilioFiscal_id = d.id
WHERE e.eliminado = FALSE;
```

C. Creación de empresa sin domicilio.

- **Acción:** Mismo menú, respondiendo “N” a la pregunta de domicilio.
- **Resultado esperado:**
  - La empresa se inserta con domicilioFiscal_id = NULL, lo que demuestra que el modelo soporta 1→0..1.

D. Actualización y baja lógica.

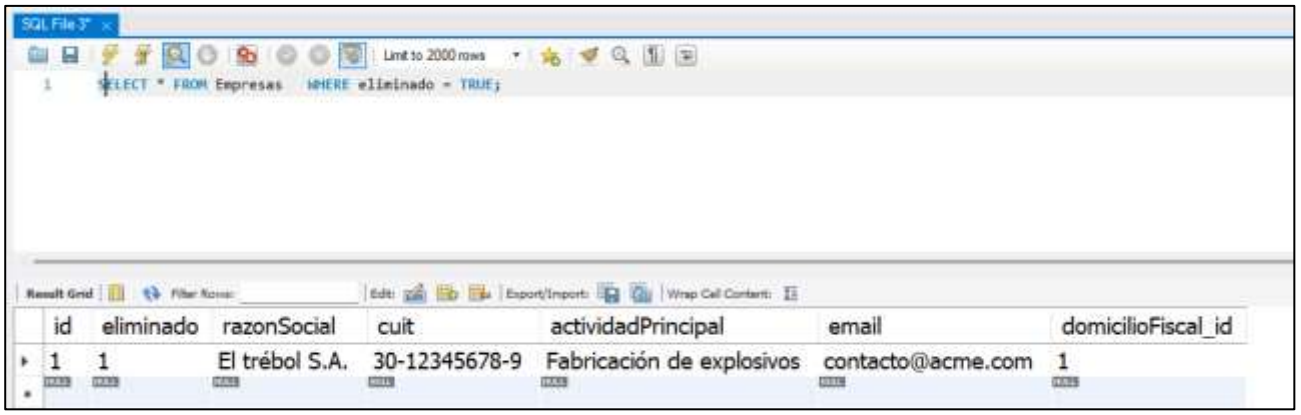
- Pruebas sobre ambas entidades:
  - Modificar datos (razón social, email, calle, número).
  - Eliminar empresa/domicilio (marcar eliminado = TRUE).
- Consultas SQL para verificar baja lógica:

```
SELECT * FROM Empresas   WHERE eliminado = TRUE;
SELECT * FROM Domicilios WHERE eliminado = TRUE;
```

```
Seleccione una opci�n: 5
== Eliminar empresa (baja l�gica) ==
Ingrese ID de la empresa a eliminar: 1
Si exist�a, la empresa fue marcada como eliminada.
```

Empresa con ID = 1 Eliminada

Muestra que la baja es lógica



E. Prueba de Rollback Transaccional

Método probado: crearEmpresaConDomicilio en EmpresaServiceImpl.java.

Estrategia de fallo controlado:

- Se forzó un error de integridad intentando crear una empresa con un CUIT duplicado (ya existente en la tabla Empresas), lo cual dispara una violación de la restricción UNIQUE y genera un SQLException con estado 23000.
- Resultado esperado: La aplicación debe mostrar el error, pero al consultar la base de datos, ningún registro (ni Empresa, ni Domicilio) debe existir.

Flujo observado:

1. El servicio entra en el bloque:

```
try (TransactionManager tx = new TransactionManager()) {

    Connection conn = tx.getConnection();

    domicilioDao.crear(domicilioFiscal, conn);

    empresaDao.crear(empresa, conn); // Aquí falla por CUIT duplicado
```

```
    tx.commit();
}
```

2. Al producirse la excepción:

- No se ejecuta tx.commit().
- close() detecta que completed == false y llama a rollback().
- Se cierra la conexión.

3. Verificación en la base de datos:

```
SELECT COUNT(*) FROM Domicilios;

SELECT * FROM Empresas WHERE razonSocial = 'Empresa con CUIT
duplicado';
```

- El contador de Domicilios **no aumenta** (no queda el domicilio “huérfano”).
- No existe ninguna empresa con la razón social usada en la prueba.

Falló la creación por CUIT duplicado

```
Seleccione una opci□n: 1
== Crear nueva empresa ==
Raz□n social: JPG HNOS
CUIT: 30-13369017-5
Actividad principal (opcional): Inversiones
Email (opcional):
□Desea crear un domicilio fiscal ahora? (S/N): n
Ocurri□ un error: Error al insertar empresa: posible violaci□n de restricci□n (CUIT duplicado u otra constraint).
```

En la BBDD no se agregó la nueva empresa

id	eliminado	razonSocial	cuit	actividadPrincipal	email	domicilioFiscal_id
1	1	El trébol S.A.	30-12345678-9	Fabricación de explosivos	contacto@acme.com	1
2	0	Tech Solutions SRL	30-87654321-0	Servicios de software	info@techsolutions.com	2
3	0	Transporte Norte S.A.	30-11112222-3	Transporte de cargas	contacto@tnorte.com	
4	0	Panificados del Sur	30-99998888-7	Panadería industrial	ventas@panificados.com	3
5	0	Martinez.SRL	30-13369017-5	Comercio	Martinezsrl@gmail.com	5

Conclusiones y Mejoras Futuras

A. Conclusiones

El proyecto ha cumplido exitosamente con los objetivos de:

1. Implementar una arquitectura de software por capas clara y modular (config, entities, dao, service, main), favoreciendo la separación de responsabilidades.
2. Modelar y gestionar una relación Uno a Uno de forma íntegra en las capas de persistencia y negocio, entre Empresa y DomicilioFiscal, garantizando la unicidad mediante una clave foránea única.
3. Establecer un mecanismo de transaccionalidad robusto mediante la clase TransactionManager y el patrón try-with-resources, garantizando la atomicidad de las operaciones críticas como la creación de Empresa/Domicilio. El sistema maneja errores de forma controlada gracias a la propagación de excepciones hasta la capa de presentación.
4. Implementa baja lógica en todas las entidades, evitando pérdida física de datos.
5. Aplica validaciones tanto en la base de datos como en la capa de servicio, especialmente para el CUIT y los campos obligatorios.

B. Mejoras Futuras

- **Validación de Negocio:** Agregar validación del formato del CUIT (expresión regular).
- **Optimización de Conexiones:** Implementar un *Connection Pool* (ej. HikariCP) para mejorar la performance en entornos concurrentes, en lugar de abrir y cerrar una conexión por transacción.
- **Interfaz Gráfica:** Migrar la capa de presentación de la consola a una interfaz gráfica (ej. Swing o JavaFX).
- **Testing Unitario:** Crear *mocks* de los DAOs para probar la lógica transaccional y las reglas de negocio del Service de forma aislada.

Fuentes y Herramientas Utilizadas

- Lenguaje: Java 17
- Base de datos: MySQL 8.x
- IDE: NetBeans
- Gestor de dependencias / build: (si usan Gradle o simple proyecto de NetBeans, aclarar)
- Diseño UML: Umletino
- Control de versiones: Git + GitHub (https://github.com/Farvon/UTN-TUPaD-P2-TPI)
- Herramientas de apoyo / documentación:
    - Documentación oficial de Java y JDBC.
    - Material teórico de la cátedra de Programación 2.
    - Asistencia de herramientas de IA generativa (ChatGPT, Gemini) para refinar la arquitectura y detectar errores en el código)