

Trabajo Final Integrador

Sistema de Gestión de Pacientes e Historias Clínicas

Materia: Programación 2

Carrera: Tecnicatura Universitaria en Programación

Integrantes: Pablo Garay, Jose Dario Gimenez, Israel Garcia Moscoso, Juan Gelos

Fecha de Entrega: Noviembre 2025

Índice

1. Elección del Dominio y Justificación
2. Diseño del Sistema
 - Decisiones de Diseño
 - Diagrama UML
 - Relación 1→1 Unidireccional
3. Arquitectura por Capas
 - Estructura del Proyecto
 - Responsabilidades de Cada Paquete
4. Persistencia de Datos
 - Estructura de la Base de Datos
 - Orden de Operaciones y Transacciones
5. Validaciones y Reglas de Negocio
6. Pruebas Realizadas
 - Capturas del Menú
 - Consultas SQL Útiles
7. Conclusiones y Mejoras Futuras
8. Referencias y Herramientas Utilizadas
9. Links

1. Elección del Dominio y Justificación

1.1 Dominio Seleccionado

Se ha elegido el dominio de **Gestión de Pacientes e Historias Clínicas** para el desarrollo del sistema. Este dominio representa un caso de uso del mundo real en el ámbito de la salud, donde existe una relación natural y necesaria entre pacientes y sus historias clínicas.

1.2 Justificación de la Elección

Ventajas del Dominio Elegido:

- ✓ **Relación 1→1 Natural:** Un paciente tiene exactamente una historia clínica y cada historia clínica pertenece a un único paciente. Esta relación unidireccional es intuitiva y refleja la realidad del dominio médico.
- ✓ **Relevancia Práctica:** Los sistemas de gestión hospitalaria son fundamentales en cualquier institución de salud, haciendo que este proyecto tenga aplicabilidad real.
- ✓ **Complejidad Adecuada:** El dominio permite implementar todas las funcionalidades requeridas (CRUD, transacciones, validaciones) sin ser excesivamente complejo.
- ✓ **Datos Significativos:** Los atributos de ambas entidades son fáciles de comprender y validar (DNI, nombre, grupo sanguíneo, etc.).

1.3 Descripción del Sistema

El sistema permite gestionar información de pacientes y sus respectivas historias clínicas, ofreciendo las siguientes capacidades:

- **Gestión de Pacientes:** Crear, listar, buscar (por ID o DNI), actualizar y eliminar pacientes.
- **Gestión de Historias Clínicas:** Crear, listar, buscar (por ID o número de historia), actualizar y eliminar historias clínicas.
- **Operaciones Combinadas:** Crear un paciente junto con su historia clínica en una única transacción atómica.
- **Baja Lógica:** Los registros no se eliminan físicamente, sino que se marcan como "eliminados".
- **Validaciones:** Control de datos obligatorios, formatos (DNI), fechas, unicidad y restricciones de relación.

2. Diseño del Sistema

2.1 Decisiones Clave de Diseño

2.1.1 Relación 1→1 Unidireccional

Se implementó una relación **unidireccional** donde sólo la clase Paciente contiene una referencia a HistoriaClinica. Esto significa que:

Desde Paciente: Se puede acceder directamente a su historia clínica mediante `paciente.getHistoriaClinica()`

Desde HistoriaClinica: NO se puede navegar directamente al paciente. Solo se almacena el `idPaciente` como referencia.

2.1.2 FK Única vs PK Compartida

Se eligió usar **Foreign Key (FK) única** en lugar de Primary Key (PK) compartida:

Aspecto	FK Única (Elegida)	PK Compartida
Flexibilidad	✓ Permite crear entidades independientemente	✗ Requiere crear ambas simultáneamente
Mantenimiento	✓ Más fácil de modificar	✗ Cambios más complejos
Claridad	✓ Separación clara de identidades	✗ Dependencia fuerte
Orden de Creación	✓ Paciente primero, HC después	✗ Deben crearse juntas

2.1.3 Implementación en la Base de Datos

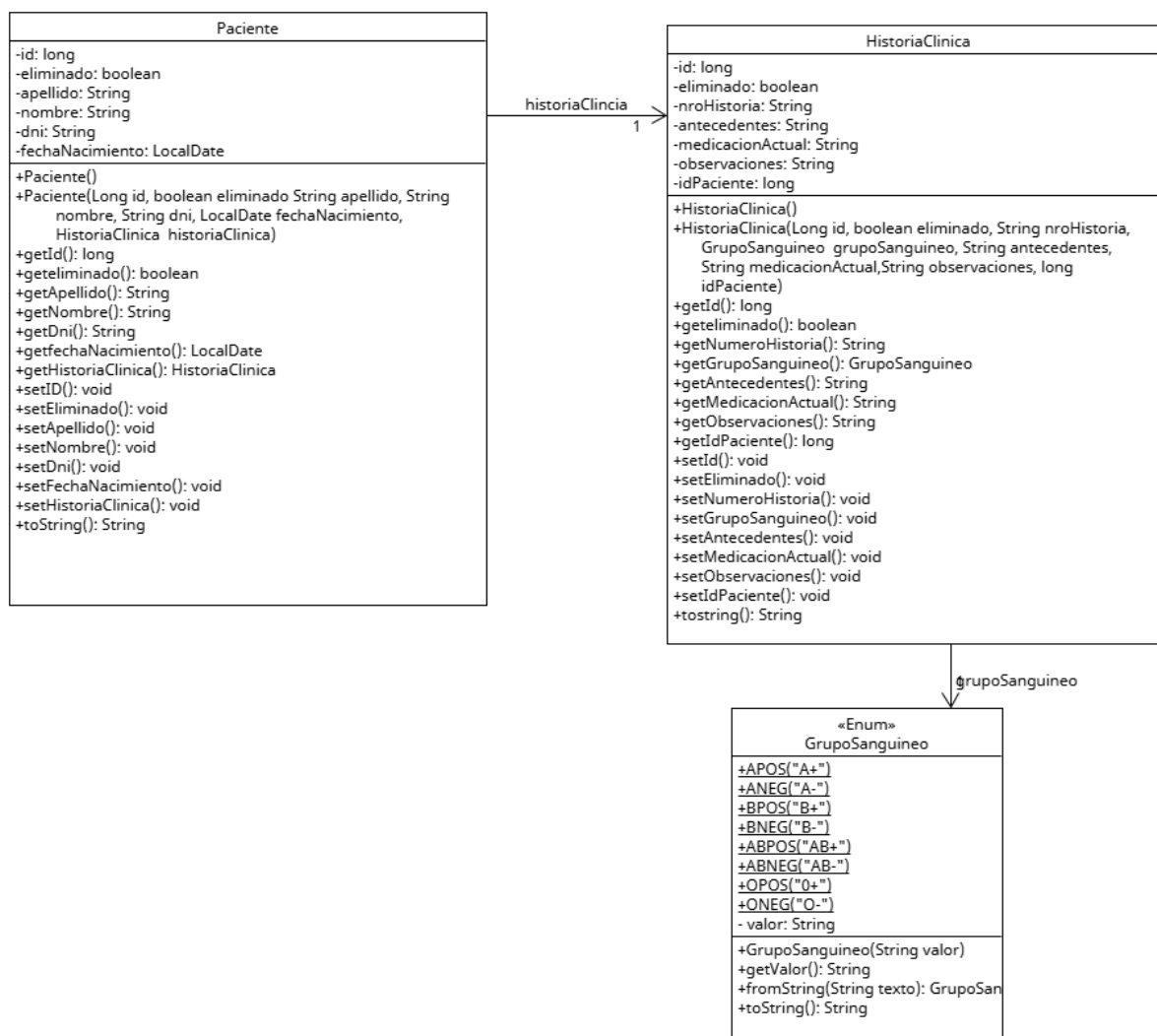
```
-- Tabla Paciente (Clase A)

CREATE TABLE paciente ( id INT AUTO_INCREMENT PRIMARY KEY NOT NULL, eliminado
BOOLEAN NOT NULL DEFAULT false, apellido VARCHAR(40) NOT NULL, nombre VARCHAR(40)
NOT NULL, dni VARCHAR(15) NOT NULL UNIQUE, fecha_nac DATE NOT NULL );

-- Tabla HistoriaClinica (Clase B)
```

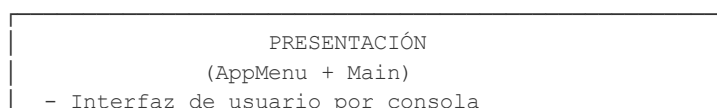
```
CREATE TABLE historiaClinica ( id INT AUTO_INCREMENT PRIMARY KEY NOT NULL,
eliminado BOOLEAN NOT NULL DEFAULT false, nro_historia VARCHAR(20) NOT NULL UNIQUE,
grupo_sangre VARCHAR(10) NOT NULL, antecedentes TEXT NOT NULL, observaciones TEXT
NOT NULL, medicacionActual VARCHAR(255), id_paciente INT NOT NULL UNIQUE, -- FK
ÚNICA CONSTRAINT fk_HC_paciente FOREIGN KEY (id_paciente) REFERENCES paciente (id)
);
```

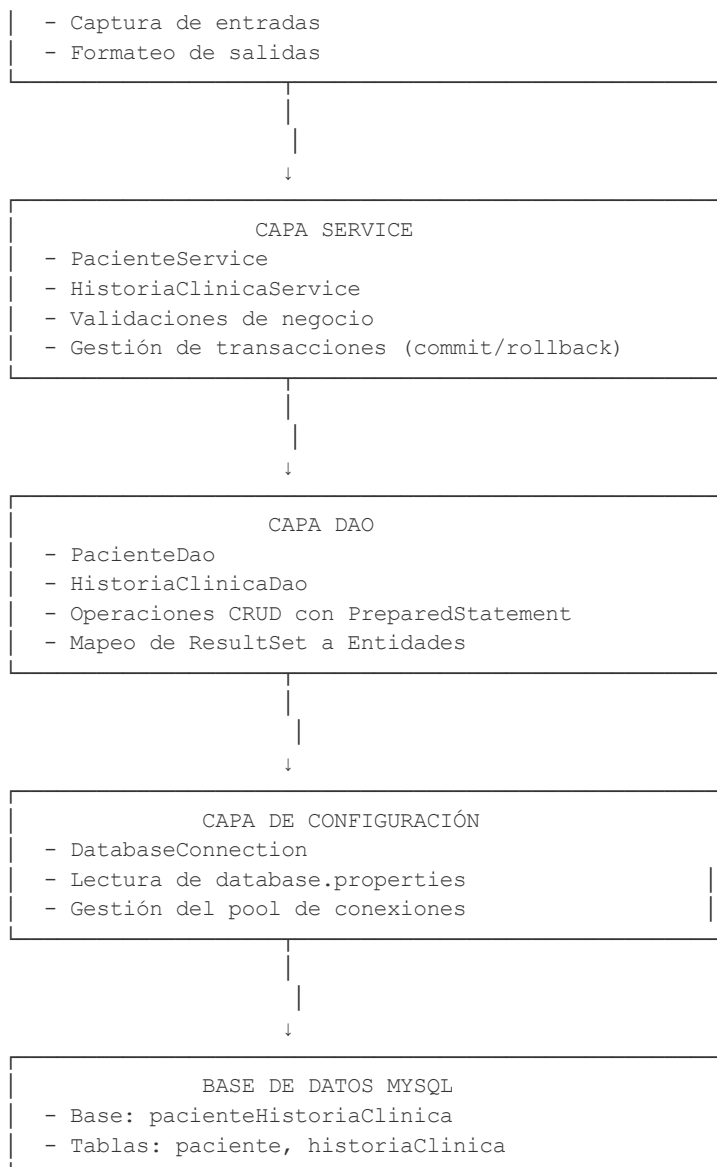
2.2 Diagrama UML de Clases



3. Arquitectura por Capas

3.1 Estructura del Proyecto





3.2 Responsabilidades de Cada Paquete



config/

Responsabilidad: Gestión de conexiones a la base de datos.

- `DatabaseConnection.java`: Carga propiedades y crea conexiones



entities/

Responsabilidad: Modelos de dominio (POJOs).

- `Paciente.java`

📦 service/

Responsabilidad: Lógica de negocio y transacciones.

- `GenericService.java`: Interfaz genérica
- `PacienteService.java`: Validaciones, transacciones
- `HistoriaClinicaService.java`: Validaciones, transacciones

📦 exceptions/

Responsabilidad: Excepciones personalizadas.

- `DatabaseException.java`: Errores de BD
- `ValidacionException.java`: Errores de validación

📦 util/

Responsabilidad: Utilidades y helpers.

- `Validador.java`: Validaciones de formato (DNI, fechas, emails)

📦 main/

Responsabilidad: Punto de entrada y menú.

- `Main.java`: Método `main()`
- `AppMenu.java`: Interfaz de consola interactiva

3.3 Ventajas de la Arquitectura en Capas

- **Separación de Responsabilidades:** Cada capa tiene una función específica y bien definida.
- **Mantenibilidad:** Los cambios en una capa no afectan a las demás.
- **Reutilización:** Los DAOs y Services pueden ser utilizados desde diferentes interfaces.
- **Testabilidad:** Cada capa puede ser probada de forma independiente.
- **Escalabilidad:** Es fácil agregar nuevas funcionalidades siguiendo el mismo patrón.

4. Persistencia de Datos

4.1 Estructura de la Base de Datos

4.1.1 Diseño de Tablas

Tabla: paciente

Campo	Tipo	Restricciones	Descripción
id	INT	PRIMARY KEY, AUTO_INCREMENT	Identificador único

eliminado	BOOLEAN	NOT NULL, DEFAULT false	Baja lógica
apellido	VARCHAR(40)	NOT NULL	Apellido del paciente
nombre	VARCHAR(40)	NOT NULL	Nombre del paciente
dni	VARCHAR(15)	NOT NULL, UNIQUE	DNI único
fecha_nac	DATE	NOT NULL	Fecha de nacimiento

Tabla: historiaClinica

Campo	Tipo	Restricciones	Descripción
id	INT	PRIMARY KEY, AUTO_INCREMENT	Identificador único
eliminado	BOOLEAN	NOT NULL, DEFAULT false	Baja lógica
nro_historia	VARCHAR(20)	NOT NULL, UNIQUE	Número de historia único
grupo_sangre	VARCHAR(10)	NOT NULL	Grupo sanguíneo
antecedentes	TEXT	NOT NULL	Antecedentes médicos
medicacionActual	VARCHAR(255)	NULL	Medicación actual (opcional)

observaciones	TEXT	NOT NULL	Observaciones generales
id_paciente	INT	NOT NULL, UNIQUE, FK	Referencia a paciente (1→1)

4.1.2 Restricciones de Integridad

- **Constraint FK:** fk_HC_paciente garantiza que cada historia clínica esté asociada a un paciente existente.
- **Constraint UNIQUE:** id_paciente asegura la relación 1→1 (un paciente, una historia).
- **Constraint CHECK:** Valida que eliminado sea 0 o 1.
- **Trigger:** trg_validar_fecha_nacimiento previene fechas futuras.

4.2 Orden de Operaciones y Transacciones

4.2.1 Gestión de Transacciones en la Capa Service

Las transacciones se manejan exclusivamente en la **capa Service**. Los DAOs reciben conexiones como parámetro cuando forman parte de una transacción.

4.2.2 Ejemplo: Crear Paciente con Historia Clínica

```
public Paciente crearConHistoriaClinica(Paciente paciente, HistoriaClinica hc) {
    Connection conexion = null;

    try {
        // 1. Obtener conexión
        conexion = DatabaseConnection.getConnection();

        // 2. DESACTIVAR autoCommit (INICIO DE TRANSACCIÓN)
        conexion.setAutoCommit(false);

        // 3. Crear el paciente (INSERT)
        Paciente pacienteCreado = pacienteDao.crear(conexion, paciente);

        // 4. Asociar HC al ID del paciente
        hc.setIdPaciente(pacienteCreado.getId());

        // 5. Crear la historia clínica (INSERT)
        HistoriaClinica hcCreada = historiaClinicaDao.crear(conexion, hc);

        // 6. COMMIT - Todo salió bien
        conexion.commit();
    }
}
```

```
return pacienteCreado;

} catch (Exception e) {

// 7. ROLLBACK - Revertir cambios si hubo error

if (conexion != null) {

try {

conexion.rollback();

System.err.println("Transacción revertida");

} catch (SQLException ex) {

System.err.println("Error en rollback");

}

} throw new DatabaseException("Error: " + e.getMessage(), e);

} finally {

// 8. Restablecer autoCommit y cerrar

if (conexion != null) {

try {

conexion.setAutoCommit(true);

conexion.close();

} catch (SQLException e) {

System.err.println("Error al cerrar conexión");

}

}
```

4.2.3 Flujo de una Transacción

1. **Obtener conexión:** `DatabaseConnection.getConnection()`
2. **Desactivar autoCommit:** `conexion.setAutoCommit(false)`
3. **Ejecutar operaciones:** Múltiples INSERTs/UPDATEs
4. **Commit o Rollback:**
 - Si todo OK → `conexion.commit()`
 - Si hay error → `conexion.rollback()`
5. **Restablecer y cerrar:** `setAutoCommit(true)` y `close()`

Importante: Las transacciones garantizan **ATOMICIDAD**. Si falla la creación de la historia clínica, también se revierte la creación del paciente, manteniendo la consistencia de los datos.

5. Validaciones y Reglas de Negocio

5.1 Validaciones en la Capa Service

5.1.1 Validaciones de Paciente

- **Apellido:** No vacío, máximo 40 caracteres, solo permite letras
- **Nombre:** No vacío, máximo 40 caracteres, solo permite letras
- **DNI:** No vacío, formato numérico de 7-8 dígitos, único en la BD
- **Fecha de Nacimiento:** No nula, no futura (validado también con trigger)

5.1.2 Validaciones de Historia Clínica

- **Número de Historia:** No vacío, máximo 20 caracteres, único en la BD
- **Grupo Sanguíneo:** Debe ser un valor del enum GrupoSanguineo
- **Antecedentes:** No vacío
- **Observaciones:** No vacío
- **ID Paciente:** Debe existir en la tabla paciente

5.1.3 Reglas de Negocio

Regla 1: Relación 1→1

Un paciente puede tener **máximo una** historia clínica. Si se intenta asociar una segunda, se lanza `ValidacionException`.

Regla 2: Historia Clínica Única

Una historia clínica solo puede estar asociada a **un paciente**. No se permite reasignar una HC ya vinculada.

Regla 3: Baja Lógica

Las eliminaciones son **lógicas**, no físicas. Los registros se marcan como `eliminado = true` pero permanecen en la BD para auditoría.

Regla 4: DNI Único

No pueden existir dos pacientes con el mismo DNI (incluso si uno está eliminado lógicamente).

5.2 Clase Validador (util/)

```
public class Validador {
```

```
// Patrón para DNI: 7-8 dígitos

private static final Pattern PATRON_DNI = Pattern.compile("[0-9]{7,8}$");

// Valida que no esté vacío

public static void validarNoVacio(String valor, String campo) {

    if (valor == null || valor.trim().isEmpty()) {

        throw new ValidacionException( "El campo " + campo + " es obligatorio" );

    }

}

// Valida formato de DNI public static void validarDni(String dni) {

    if (dni == null || !PATRON_DNI.matcher(dni).matches()) {

        throw new ValidacionException( "El DNI debe tener 7 u 8 dígitos numéricos" );

    }

}

// Valida que la fecha no sea futura

public static void validarFechaNoFutura(LocalDate fecha) {

    if (fecha != null && fecha.isAfter(LocalDate.now())) {

        throw new ValidacionException( "La fecha no puede ser futura" );

    }

}

}
```

5.3 Manejo de Excepciones

El sistema define dos excepciones personalizadas:

- **ValidacionException:** Errores de validación de datos (formato, obligatoriedad, reglas de negocio)
- **DatabaseException:** Errores de base de datos (conexión, SQL, transacciones)

Ambas extienden `RuntimeException` para evitar la obligatoriedad de declararlas en cada método, simplificando el código.

6. Pruebas Realizadas

6.1 Capturas del Menú Interactivo

6.1.1 Menú Principal

```
┌────────────────────────────────────────────────────────────────────────────────┐
│ SISTEMA DE GESTIÓN DE PACIENTES E HISTORIAS CLÍNICAS │
└────────────────────────────────────────────────────────────────────────────────┘

┌────────────────────────────────────────────────────────────────────────────────┐
│ MENÚ PRINCIPAL │
└────────────────────────────────────────────────────────────────────────────────┘

[1] Gestión de Pacientes |
| [2] Gestión de Historias Clínicas |
| [3] Operaciones Combinadas |
| [0] Salir |
└────────────────────────────────────────────────────────────────────────────────┘

➤ Seleccione una opción:
```

6.1.2 Menú Gestión de Pacientes

```
┌────────────────────────────────────────────────────────────────────────────────┐
│ GESTIÓN DE PACIENTES │
└────────────────────────────────────────────────────────────────────────────────┘

|
| [1] Crear Paciente |
| [2] Listar Todos los Pacientes |
| [3] Buscar Paciente por ID |
| [4] Buscar Paciente por DNI |
| [5] Actualizar Paciente |
| [6] Eliminar Paciente |
| [0] Volver al Menú Principal |
└────────────────────────────────────────────────────────────────────────────────┘
```

6.1.3 Ejemplo de Listado de Pacientes

```
══ LISTADO DE PACIENTES ══
```

ID	APELLIDO	NOMBRE	DNI	FECHA NAC.
1	RODRIGUEZ	CARLOS	12345678	15/03/1985
2	MARTINEZ	ANA	23456789	22/07/1990

3	GOMEZ	LUIS	34567890	10/11/1978
4	FERNANDEZ	MARIA	45678901	05/04/1995

Total de pacientes: 4

6.2 Consultas SQL Útiles

6.2.1 Ver Pacientes con Historia Clínica (INNER JOIN)

```
SELECT p.id AS paciente_id, p.apellido, p.nombre, p.dni, hc.id AS hc_id,
hc.nro_historia, hc.grupo_sangre FROM paciente p INNER JOIN historiaClinica hc ON
p.id = hc.id_paciente WHERE p.eliminado = false AND hc.eliminado = false ORDER BY
p.apellido;
```

6.2.2 Ver Pacientes SIN Historia Clínica (LEFT JOIN)

```
SELECT p.id, p.apellido, p.nombre, p.dni FROM paciente p LEFT JOIN historiaClinica
hc ON p.id = hc.id_paciente AND hc.eliminado = false WHERE p.eliminado = false AND
hc.id IS NULL ORDER BY p.apellido;
```

6.2.3 Distribución por Grupo Sanguíneo

```
SELECT grupo_sangre, COUNT(*) AS cantidad FROM historiaClinica WHERE eliminado =
false GROUP BY grupo_sangre ORDER BY cantidad DESC;
```

6.2.4 Verificar Integridad de Relación 1→1

```
-- Verificar que no haya pacientes con más de 1 HC SELECT id_paciente, COUNT(*) as
cantidad_hc FROM historiaClinica WHERE eliminado = false GROUP BY id_paciente
HAVING COUNT(*) > 1; -- Resultado esperado: 0 filas (sin duplicados)
```

6.3 Casos de Prueba Ejecutados

Caso de Prueba	Resultado Esperado	Estado
Crear paciente válido	Paciente creado exitosamente	✓ Passed
Crear paciente con DNI duplicado	ValidacionException	✓ Passed
Crear paciente con fecha futura	ValidacionException + Trigger	✓ Passed
Crear historia clínica válida	HC creada exitosamente	✓ Passed

Asociar 2ª HC a un paciente	ValidacionException (1→1)	✓ Passed
Crear paciente + HC (transacción)	Ambos creados o ninguno	✓ Passed
Rollback por error en HC	Paciente no creado (revertido)	✓ Passed
Buscar por DNI existente	Paciente encontrado	✓ Passed
Buscar por DNI inexistente	Optional.empty()	✓ Passed
Eliminar paciente (baja lógica)	eliminado = true	✓ Passed

7. Conclusiones y Mejoras Futuras

7.1 Conclusiones

7.1.1 Objetivos Cumplidos

- ✓ Se implementó correctamente una relación **1→1 unidireccional** entre Paciente e HistoriaClinica.
- ✓ Se utilizó **JDBC puro** (sin ORM) con PreparedStatement en todas las operaciones.
- ✓ Se aplicó el **patrón DAO** para separar la lógica de persistencia.
- ✓ Se implementó una **capa Service** con gestión de transacciones (commit/rollback).
- ✓ Se desarrolló un **menú de consola interactivo** con manejo de excepciones.
- ✓ Se implementaron **validaciones robustas** en todas las capas.
- ✓ Se utilizó **baja lógica** para preservar la integridad histórica.

7.1.2 Aprendizajes Clave

Arquitectura en Capas: La separación de responsabilidades facilita el mantenimiento y la evolución del sistema. Cada capa tiene un propósito claro y puede ser modificada independientemente.

Transacciones: El manejo correcto de transacciones es fundamental para garantizar la consistencia de los datos. El uso de commit/rollback previene estados inconsistentes.

Validaciones: Es crucial validar en múltiples niveles (aplicación y BD) para garantizar la integridad de los datos. Las excepciones personalizadas mejoran la claridad del código.

JDBC sin ORM: Aunque más verboso que un ORM, JDBC ofrece control total sobre las consultas SQL y ayuda a comprender mejor cómo funciona la persistencia de datos.

7.2 Mejoras Futuras

7.2.1 Funcionalidades

- **Historial de Consultas:** Agregar una entidad Consulta para registrar visitas médicas.
- **Gestión de Usuarios:** Sistema de autenticación con roles (médico, admin, recepcionista).
- **Búsquedas Avanzadas:** Filtros por rango de fechas, grupo sanguíneo, edad, etc.
- **Reportes:** Generar informes en PDF con información de pacientes e historias clínicas.
- **Gestión de Turnos:** Sistema de agenda para turnos médicos.
- Dial numérico para GS en la creación de HC
- Confirmación de la creación de HC
- Quitar actualizar el grupo sanguíneo (GS)
- Cuando hay un error de validación de datos, en cualquier parte del menú, debería permitir corregirlo. No cerrar y volver al menú principal
- Tras crear la operación combinada muestre todos los datos ingresados. Actualmente no muestra ni fecha nacimiento, observaciones, antecedentes y medicamentos
-

7.2.2 Técnicas

- **Pool de Conexiones:** Implementar HikariCP o Apache DBCP para mejorar el rendimiento.
- **Logging:** Integrar SLF4J + Logback para registro de operaciones y errores.
- **Testing:** Agregar pruebas unitarias con JUnit 5 y Mockito.
- **Interfaz Gráfica:** Migrar a JavaFX o Swing para una interfaz más amigable.
- **API REST:** Exponer funcionalidades mediante Spring Boot para integración web/móvil.
- **Migraciones:** Usar Flyway o Liquibase para control de versiones de la BD.

7.2.3 Seguridad

- **Encriptación:** Cifrar datos sensibles (antecedentes médicos).
- **Auditoría:** Registrar quién y cuándo modifica cada registro.
- **Respaldos:** Sistema automatizado de backups de la base de datos.

7.3 Reflexión Final

Este proyecto ha permitido aplicar conceptos fundamentales de programación orientada a objetos, patrones de diseño (DAO, Service), gestión de bases de datos relacionales y buenas prácticas de desarrollo de software. La experiencia adquirida en el manejo de transacciones, validaciones y arquitectura en capas será valiosa para futuros desarrollos profesionales.

El dominio elegido (Pacientes e Historias Clínicas) demostró ser adecuado para cumplir con los requisitos del trabajo integrador, ofreciendo casos de uso realistas y desafíos técnicos interesantes de resolver.

8. Referencias y Herramientas Utilizadas

8.1 Tecnologías Principales

Tecnología	Versión	Propósito
Java	21	Lenguaje de programación principal
MySQL	8.0+	Sistema de gestión de base de datos
JDBC (MySQL Connector/J)	8.3.0	Driver de conexión a MySQL
Maven	3.x	Gestión de dependencias y build

8.2 Herramientas de Desarrollo

- **IDE:** NetBeans / Visual Studio Code
- **Control de Versiones:** Git + GitHub
- **Cliente MySQL:** MySQL Workbench / DBeaver
- **Terminal:** PowerShell / Bash

8.3 Documentación y Referencias

Documentación Oficial:

- Oracle Java Documentation:
<https://docs.oracle.com/en/java/javase/21/>
- MySQL Reference Manual: <https://dev.mysql.com/doc/refman/8.0/en/>
- JDBC Tutorial: <https://docs.oracle.com/javase/tutorial/jdbc/>

- Maven Documentation: <https://maven.apache.org/guides/>

Patrones de Diseño:

- DAO Pattern: <https://www.baeldung.com/java-dao-pattern>
- Service Layer Pattern:
<https://martinfowler.com/eaCatalog/serviceLayer.html>

Bases de Datos:

- Relaciones 1:1 en MySQL:
<https://www.mysqltutorial.org/mysql-one-to-one-relationship/>
- Transacciones ACID:
<https://dev.mysql.com/doc/refman/8.0/en/mysql-acid.html>

8.4 Uso de Inteligencia Artificial

GitHub Copilot

Utilizado para:

- Autocompletado de código repetitivo (getters/setters, constructores)
- Sugerencias de validaciones y manejo de excepciones
- Corrección y apoyo en consultas SQL complejas
- Documentación JavaDoc de métodos

ChatGPT / Claude

Utilizado para:

- Consultas sobre mejores prácticas en JDBC
- Resolución de dudas sobre transacciones y rollback
- Diseño de la estructura de la base de datos
- Revisión de conceptos teóricos de patrones DAO y Service

Nota: Todo el código generado fue revisado, modificado y adaptado según las necesidades específicas del proyecto.

8.5 Bibliografía Complementaria

- Horstmann, Cay S. - *"Core Java Volume I: Fundamentals"* (12th Edition)
- Bloch, Joshua - *"Effective Java"* (3rd Edition)
- Fowler, Martin - *"Patterns of Enterprise Application Architecture"*

8.6 Repositorio del Proyecto

Estructura del Repositorio:

- /src/main/java/ - Código fuente Java
- /src/main/resources/ - Archivos de configuración
- db.sql - Script de creación de BD
- datos_prueba.sql - Datos de prueba
- consultas utiles.sql - Consultas SQL de verificación
- README.md - Documentación del proyecto
- pom.xml - Configuración Maven

8 Links

GitHub: <https://github.com/814942/UTN-TUP-P2-TP>

Video: <https://www.youtube.com/watch?v=XU6enONryOs>