Trabajo Final Integrador (TFI)

Informe Técnico - Trabajo Práctico Integrador

Integrantes:

- Franco Sarru: Services
- Nahuel Ayala: DAO
- Ignacio Valentin Roveres: App Menu
- Gonzalo Ronderos: Base de Datos

1. Introducción

Este trabajo práctico integrador consiste en el desarrollo de un sistema de gestión hospitalaria orientado a la gestión de pacientes y sus historias clínicas. La aplicación está desarrollada en Java, utilizando JDBC para la persistencia en base de datos MySQL y una arquitectura en capas (DAO, Service, Menu/Handler).

2. Arquitectura y Diseño

2.1. Estructura de Paquetes

config/

- DataBaseConnection.java: centraliza la configuración y apertura de la conexión JDBC a la base de datos MariaDB. Es el “puente” entre la aplicación y la base de datos.

dao/

- GenericDAO.java: interfaz genérica con operaciones CRUD básicas.
- HistoriaClinicaDAO.java: acceso a datos específico para la entidad Historia Clínica.
- PacienteDAO.java: acceso a datos específico para la entidad Paciente, incluyendo consultas con JOIN para traer la historia clínica asociada.

entities/

- GrupoSanguineo.java: enum que define los valores válidos de grupo sanguíneo y métodos de conversión segura.

- HistoriaClinica.java: clase que modela la historia clínica, con atributos como antecedentes, medicación y observaciones.
- Paciente.java: clase que modela al paciente, vinculada directamente con su historia clínica.

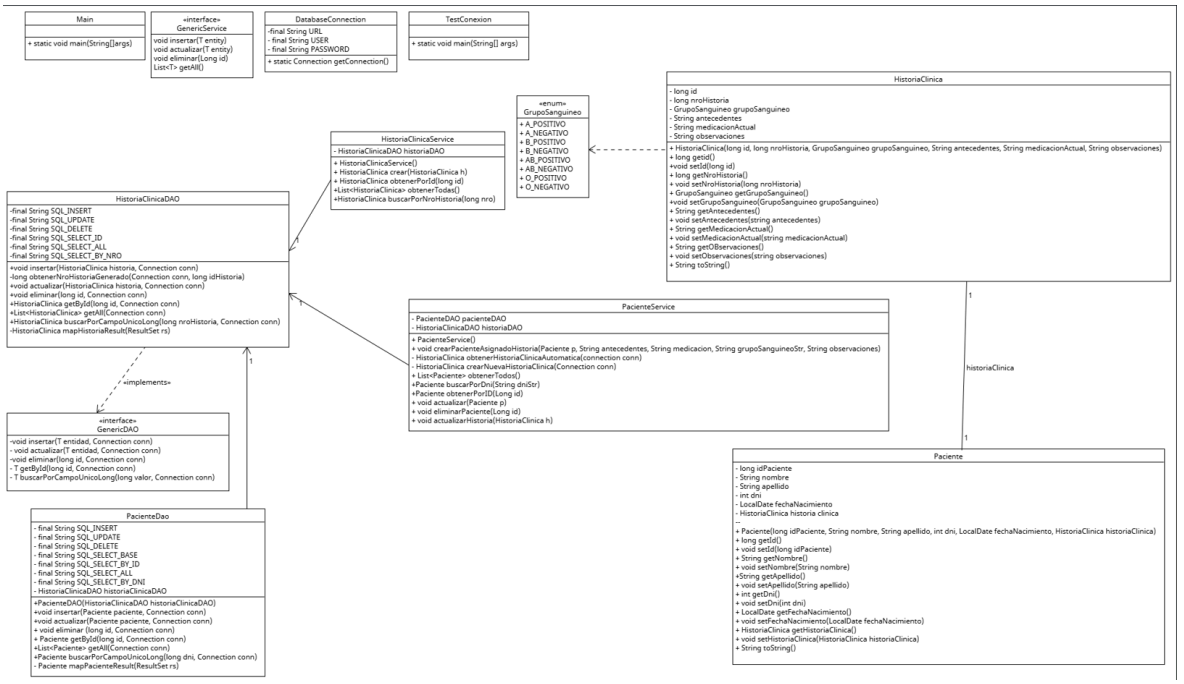
main/

- AppMenu.java: menú principal por consola, permite al usuario crear, listar, buscar, actualizar y eliminar pacientes.
- Main.java: punto de entrada de la aplicación, inicializa el menú.
- TestConection.java: clase auxiliar para verificar que la conexión con la base de datos funciona correctamente.

services/

- GenericService.java: interfaz de lógica de negocio genérica, define operaciones CRUD.
- HistoriaClinicaService.java: lógica de negocio específica para historias clínicas.
- PacienteServices.java: lógica de negocio específica para pacientes, incluye validaciones, manejo transaccional y reglas de unicidad.

2.2. Diagrama UML



El diagrama representa las relaciones entre entidades, DAOs, servicios y menú.

3. Implementación

3.1. Persistencia y DAO

- Se definió una interfaz genérica GenericDAO con operaciones CRUD básicas (insertar, actualizar, eliminar, getById, getAll, buscarPorCampoUnicoLong).

- Las clases PacienteDAO y HistoriaClinicaDAO implementan esta interfaz, utilizando **JDBC con** PreparedStatement para prevenir inyección SQL y manejar parámetros de forma segura.

- En PacienteDAO se utilizan consultas con **JOIN** para recuperar al paciente junto con su historia clínica asociada.

3.2. Entidades

- Paciente: modela los datos personales del paciente (ID, nombre, apellido, DNI, fecha de nacimiento) y mantiene una relación directa con su historia clínica.

- HistoriaClinica: modela los datos médicos básicos (grupo sanguíneo, antecedentes, medicación actual, observaciones).

- GrupoSanguineo: define los valores válidos de grupo sanguíneo (A+, O-, etc.), con un método fromValor para convertir cadenas de texto de la base de datos en valores tipados de Java.

3.3. Servicios

Los servicios encapsulan la **lógica de negocio y el manejo transaccional**.

PacienteService:

- Implementa GenericService<Paciente>.
- Maneja transacciones para asegurar la integridad entre paciente e historia clínica en operaciones de inserción, actualización y eliminación.
- Valida la **unicidad del DNI** y que una historia clínica no esté asignada a más de un paciente.
- Permite la creación automática de historias clínicas básicas para nuevos pacientes.

HistoriaClinicaService:

- Proporciona métodos de alto nivel para crear, obtener y buscar historias clínicas.
- Abstrae el acceso a datos y simplifica la interacción desde la capa de presentación.

3.4. Interfaz de Usuario

La aplicación se maneja por **consola**, mediante la clase AppMenu.

El menú principal ofrece las opciones:

- Crear paciente con historia clínica.
- Listar pacientes.
- Buscar paciente por DNI.
- Actualizar datos del paciente y su historia clínica.
- Eliminar paciente (baja física).

En la actualización se permite dejar campos en blanco para mantener los valores actuales.

Se manejan excepciones para entradas inválidas (por ejemplo, formato de fecha, IDs no numéricos) y errores de base de datos.

Main.java inicializa el menú y actúa como punto de entrada del sistema.

TestConexion.java permite verificar la conexión a la base de datos y ejecutar consultas simples de prueba.

4. Pruebas

4.1. Pruebas Manuales

- Se verificó la creación exitosa de pacientes con historias clínicas, incluyendo validaciones de campos obligatorios y unicidad.

- Se probaron búsquedas por ID y DNI, listados completos y manejo correcto de baja lógica.

- Se validó la actualización de datos tanto para paciente como para historia clínica, comprobando que los datos antiguos se mantienen si se deja vacío.

- Se validaron mensajes de error para entradas inválidas (por ejemplo, formato de fecha, IDs no numéricos).

4.2. Resultado esperado

- Los datos se almacenan y recuperan correctamente desde la base de datos.
- No se permiten duplicados de DNI ni número de historia clínica.
- El sistema responde adecuadamente a errores y confirma operaciones exitosas.

5. Capturas de Código

5.1. **Creación de Paciente con Historia Clínica (AppMenu.java)**

Este fragmento muestra cómo desde el menú principal se solicita la información al usuario y se crea un nuevo paciente con su historia clínica asociada:

```java
    // CREAR PACIENTE (Historia clínica automática)
private void crearPaciente() {
    try {
        System.out.println("\n--- Nuevo Paciente ---");

        System.out.print("Nombre: ");
        String nombre = scanner.nextLine();

        System.out.print("Apellido: ");
        String apellido = scanner.nextLine();

        System.out.print("DNI: ");
        int dni = Integer.parseInt(scanner.nextLine());

        System.out.print("Fecha de nacimiento (AAAA-MM-DD): ");
        LocalDate fecha = LocalDate.parse(scanner.nextLine().trim());

        System.out.print("Grupo sanguíneo (A+, A-, B+, O+, etc.): ");
        String grupoStr = scanner.nextLine().trim().toUpperCase();

        System.out.print("Antecedentes: ");
        String antecedentes = scanner.nextLine();

        System.out.print("Medicación actual: ");
        String medicacion = scanner.nextLine();

        System.out.print("Observaciones: ");
        String observaciones = scanner.nextLine();

        // Crear paciente (HC se asigna después)
        Paciente p = new Paciente(
                0L,
                nombre,
                apellido,
                dni,
                fecha,
                null
        );

        pacienteService.crearPacienteAsignandoHistoria(
                p,
                antecedentes,
                medicacion,
                grupoStr,
                observaciones
        );

        System.out.println("✅ Paciente creado correctamente.");

    } catch (Exception e) {
        System.out.println("❌ Error al crear paciente: " + e.getMessage());
    }
}
```

5.2. **Método insertar en PacienteService.java**

Este fragmento muestra cómo se maneja la inserción de un paciente y su historia clínica de manera **transaccional**, asegurando la integridad de los datos:

```java
    @Override
    public void insertar(Paciente p) throws SQLException {

        if (p.getDni() <= 0 || p.getNombre() == null || p.getHistoriaClinica() == null) {
            throw new SQLException("Error de validación: Datos de Paciente o HC incompletos/inválidos.");
        }

        Connection conn = null;
        try {
            conn = DatabaseConnection.getConnection();
            conn.setAutoCommit(false);


            if (pacienteDAO.buscarPorCampoUnicoLong(p.getDni(), conn) != null) {
                throw new SQLException("Violación de Unicidad: Ya existe un paciente con DNI " + p.getDni());
            }


            if (pacienteDAO.isHistoriaClinicaAsignada((int) p.getHistoriaClinica().getId(), conn)) {
                throw new SQLException("Violación 1:1: La Historia Clínica ID " + p.getHistoriaClinica().getId() + " ya está asignada a otro paciente.");
            }


            historiaDAO.insertar(p.getHistoriaClinica(), conn);

            pacienteDAO.insertar(p, conn);

            conn.commit();

        } catch (SQLException ex) {

            if (conn != null) {
                try {
                    conn.rollback();
                } catch (SQLException rb) {
                }
            }
            throw ex;
        } finally {

            try {
                if (conn != null) {
                    conn.setAutoCommit(true);
                    conn.close();
                }
            } catch (SQLException c) {
            }
        }
    }
```

6. Conclusiones

- Se logró implementar un **sistema funcional de gestión de pacientes e historias clínicas**, con una arquitectura en capas bien definida: configuración, persistencia (DAO), entidades, servicios y presentación por consola.

- La **persistencia** se maneja mediante JDBC y PreparedStatement, lo que asegura consultas seguras y parametrizadas, evitando riesgos de inyección SQL.

- La **capa de servicios** encapsula la lógica de negocio y garantiza la **integridad transaccional** en operaciones críticas como inserción, actualización y eliminación, manteniendo la relación 1:1 entre paciente e historia clínica.

- Se implementaron **validaciones de unicidad** (DNI y número de historia

clínica) para evitar duplicados y preservar la consistencia de los datos.

- La **interfaz por consola** (AppMenu) ofrece una interacción simple pero completa, permitiendo al usuario realizar todas las operaciones CRUD de manera intuitiva.

- La clase TestConexion facilita la verificación rápida de la conexión a la base de datos, lo que resulta útil para depuración y pruebas iniciales.

- El diseño modular y en capas favorece la **mantenibilidad y escalabilidad** del sistema, permitiendo futuras ampliaciones (por ejemplo, baja lógica, más validaciones, o interfaz gráfica).

- El sistema responde adecuadamente a errores de entrada y de base de datos, mostrando mensajes claros al usuario y asegurando que las operaciones se confirmen o se reviertan correctamente.