Trabajo Final Integrador de Programación 2.

“Libro--Ficha Bibliográfica”

Alumnos Grupo 96:

Alemis, Juan Cruz - juancruzalemis@gmail.com

Borda, Cristian Alexis - alexborda922@gmail.com

Wittmund, Luciano - wittmundluciano@gmail.com

Zalazar, Leandro - david26692@gmail.com

Profesores:

Ariel Enferrel

Cinthia Rigoni

Fecha de entrega: 20 / 11 / 2025

Informe Final:

- **Integrantes y Roles**

Integrante	Rol principal
Borda,Cristian Alexis	Diseño UML
Wittmund, Luciano	Entidades - DAO y persistencia SQL
Zalazar, Leandro	Introducción, service y transacción, commit y rollback.
Alemis, Juan Cruz	Módulo de menú (Main, AppMenu, MenuController)

1- Elección del dominio y justificación

Se eligió el dominio 'Biblioteca' porque permite modelar un caso real con libros y fichas bibliográficas. Este dominio facilita aplicar una relación 1→1 estricta (un libro posee exactamente una ficha), y permite implementar correctamente un CRUD completo con validaciones, integridad y reglas de negocio.

2- Diseño y Decisiones Clave

Se utilizó una relación 1→1 unidireccional: la clase Libro contiene un atributo fichaBibliografica. Se eligió una clave foránea única (fichaBibliografica_id UNIQUE) en vez de PK compartida para simplificar.

El diagrama UML incluye:

Libro (A) con atributos id, titulo, autor, editorial, anioEdicion, eliminado, y referencia a ficha.

FichaBibliografica (B) con id, isbn, clasificacionDewey, estanteria, idioma, eliminado.

A → B unidireccional (1..1).

3- Arquitectura por Capas

El sistema implementado sigue una **arquitectura en capas** que separa responsabilidades para mejorar la organización, mantenibilidad y escalabilidad del código. Cada capa cumple una función específica y colabora con las otras mediante interfaces bien definidas. Las siguientes capaz son:

❖ **Capa de Entidades (package Entities)**

Responsabilidad principal: Representar las clases del dominio del sistema (modelo de datos).

Descripción: Contiene las clases simples que representan objetos reales de la aplicación, como Libro, FichaBibliografica.
Estas clases incluyen atributos, getters/setters y ninguna lógica de negocio.

Objetivo:

- Ser un reflejo directo de las tablas de la base de datos.
- Permitir trasladar información entre capas sin lógica adicional.

❖ **Capa de Acceso a Datos – DAO (package Dao)**

Responsabilidad principal: Encapsular todas las operaciones SQL hacia la base de datos.

Descripción: Esta capa se encarga de ejecutar consultas INSERT, UPDATE, DELETE, SELECT, utilizando conexiones JDBC. Cada clase DAO es responsable de una tabla específica.

Objetivo:

- Aislar la lógica de persistencia.
- Evitar SQL dentro de los servicios o controladores.
- Recibir conexiones externas para permitir transacciones desde el Service.

❖ **Capa de Servicios – Service (package Service)**

Responsabilidad principal: Aplicar la lógica de negocio, validaciones y manejo de transacciones.

Descripción: Aquí se coordinan las operaciones entre múltiples DAOs. La capa de servicio implementa:

- Validaciones de dominio (campos obligatorios, formatos).
- Reglas de negocio (por ejemplo, relación 1→1 entre Libro y Ficha).
- Manejo de transacciones JDBC (commit, rollback).
- Uso del patrón GenericService<T> para unificar las operaciones CRUD.

Objetivo:

- Evitar lógica compleja dentro de los DAO.
- Permitir operaciones compuestas (ej.: insertar ficha + libro en una sola transacción).
- Asegurar integridad de datos a nivel aplicación.

❖ **Capa de Configuración (package Config)**

Responsabilidad principal: Proporcionar acceso centralizado a la base de datos.

Descripción: Incluye la clase DatabaseConnection, encargada de:

- Cargar el driver JDBC.
- Validar parámetros de conexión.
- Crear conexiones nuevas con getConnection().

Objetivo:

- Evitar duplicación de código de conexión.

- Crear un único punto responsable de la configuración de la base de datos.

❖ **Capa de Prueba o Presentación (package Main)**

Responsabilidad principal**:** Ejecutar pruebas manuales o iniciar la aplicación.

Descripción**:** Contiene clases utilizadas para probar el funcionamiento de los servicios, como MainPrueba.

Objetivo**:**

- Verificar el correcto funcionamiento del sistema.
- Simular flujos de uso sin interfaz gráfica.

Resumen General

La arquitectura por capas implementada divide el sistema en módulos independientes, donde:

- **Entities** define el modelo.
- **Dao** accede a los datos.
- **Service** aplica reglas de negocio y controla transacciones.
- **Config** centraliza la conexión.
- **Main** ejecuta pruebas.

Esto permite un software organizado, fácil de mantener y con responsabilidades claramente separadas

4- Persistencia y Transacciones

Primero pasemos a definir una persistencia en programación la persistencia es la capacidad de un sistema para **mantener la información (datos) más allá de la vida de**

la aplicación o del proceso que los creó. En programación esto se logra con una base de datos. En este caso se usó Mysql para crear la base de datos que nosotros llamamos biblioteca y dentro de ella se implementó las entidades FichaBibliografica y Libro.

En todas ellas se crearon sus respectivos códigos para realizar las transacciones en el paquete de servicio. Primero se definio un genericService, donde se coloco las transacciones de insertar, actualizar, eliminar, getById y gerAll.
Luego por cada entidad se fue definiendo las transacciones que iban ser usadas, donde la más completa es la de Libro service, donde también se implementó el fallo simulado para ver cómo funcionaba el sistema de commit y de rollback en una transacción fallida. Esto se llevó a cabo en la transacción de insertar.

Se brinda una breve descripción de esto

Inicio: La transacción comienza al deshabilitar el autocommit: conn.setAutoCommit(false);.
Commit: Se hace en la línea conn.commit(); dentro del bloque try. Esto indica que la inserción de la ficha y la del libro han sido exitosas.
Rollback: Se hace en la línea conn.rollback(); dentro del bloque catch. Esto ocurre si la inserción de la ficha fue exitosa, pero la inserción del libro falló (por ejemplo, con el "FALLO SIMULADO"), asegurando que la ficha huérfana sea eliminada.

5- Validaciones y Reglas de Negocio

ISBN no puede ser vacío.
ISBN debe ser único.
Libro debe tener siempre una ficha asociada.
No se permite ficha preexistente al crear libro nuevo (id != 0).
Eliminación lógica en vez de física.

6- Pruebas Realizadas

Pruebas del CRUD desde el menú:

```
====== MENU PRINCIPAL ======
1 - CRUD de Libros
2 - CRUD de Fichas Bibliograficas
0 - Salir
Opci□n: 1

===== GESTI□N DE LIBROS =====
1 - Crear libro
2 - Leer libro por ID
3 - Listar libros
4 - Actualizar libro
5 - Eliminar l□gico libro
0 - Volver
Opci□n:
```

Prueba de rollback: Se intenta crear un libro con un ISBN ya existente. Base de datos lanza error UNIQUE.
El Service ejecuta rollback() y ningún registro queda guardado.

```
USE biblioteca;

SELECT * FROM fichabibliografica WHERE isbn = '999-ROLLBACK-PRUEBA';
```

Conclusiones y Mejoras Futuras

El sistema cumple los requisitos funcionales para la relación que se implementó en este trabajo practico, pero es posible llevarlo a una mayor escala con más tablas y distintas relaciones como agregar módulo de préstamos a los y autenticación de usuarios por distintos roles, también sería deseable en el futuro agregar una interfaz gráfica más amigable con el usuario. también se demuestra que es la base de una arquitectura escalable capaz de ir agregando más funcionalidades a futuro.

Fuentes y Herramientas Utilizadas

Las herramientas mas utilizadas son:

Java 17
NetBeans.
MySQL + XAMPP.
JDBC.
GitHub.
Herramientas de edición de video “Chipchamp”
Herramientas de IA como Gemini y Chatgpt como herramienta de asistencia técnica y documentación.