TECNICATURA UNIVERSITARIA
EN PROGRAMACIÓN
A DISTANCIA

UTN
UNIVERSIDAD TECNOLÓGICA NACIONAL

Trabajo Práctico Integrador
Programación

Integrante	Rol / Responsabilidad Principal
Nicolas	*Team Lead* (Coordinación, Visión General del Proyecto)
Cristian	Desarrollo de **Lógica de Negocio** (Implementación de Capa *Service*, Validaciones y Transacciones)
Mariano	Conexión con **Base de Datos** y Gestión de Transacciones
Axel	Modelado de **Entidades** y redacción de informe

2. Elección del Dominio y Justificación

El dominio seleccionado para el trabajo práctico es la Gestión Bibliotecaria, centrado específicamente en la relación entre las entidades Libro y Ficha Bibliográfica.

Justificación

Se eligió este dominio con la motivación de implementar y gestionar una relación uno a uno (1:1), donde la información se distribuye intencionalmente entre dos entidades distintas.

- Libro contendría información operativa y editorial (*título, autor, editorial, año de edición*).
- Ficha Bibliográfica contendría información descriptiva y de ubicación (*ISBN, Clasificación Dewey, estantería, idioma*).

Este modelo justifica la relación 1:1 porque:

1. Unicidad: Cada instancia de Libro tiene una única descripción bibliográfica oficial.
2. Exclusividad: Dicha Ficha Bibliográfica no debería estar asociada a más de una instancia de Libro.
3. Separación de Responsabilidades: Permite separar la información más operativa del detalle descriptivo único.

3. Diseño: Decisiones Clave y UML

3.1. Decisiones Clave: Relación Uno a Uno (1:1)

En la relación 1:1 entre **Libro (A)** y **FichaBibliografica (B)**, se observa que la clase **Libro** tiene una referencia al objeto **FichaBibliografica**:

Esto se traduce a nivel de persistencia en que la tabla Libro contendrá una **Clave Foránea (FK)** que apunta a la clave primaria (PK) de la tabla FichaBibliografica.

Esto se traduce a nivel de persistencia en que la tabla `Libro` contendrá una Clave Foránea (FK) que apunta a la clave primaria (PK) de la tabla `FichaBibliografica`.

Decisión Adoptada: Clave Foránea Única (FK única)

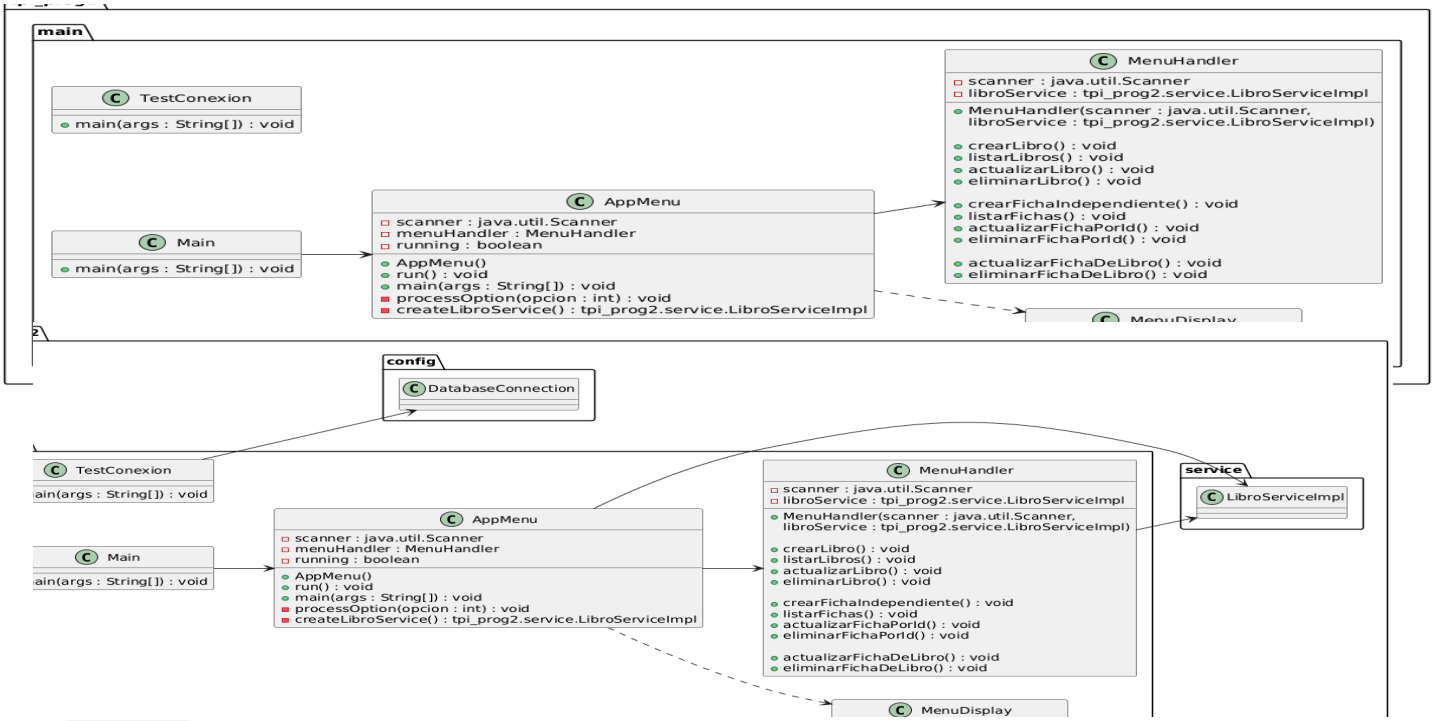
El dominio seleccionado para el trabajo práctico es la **Gestión Bibliotecaria**, centrado

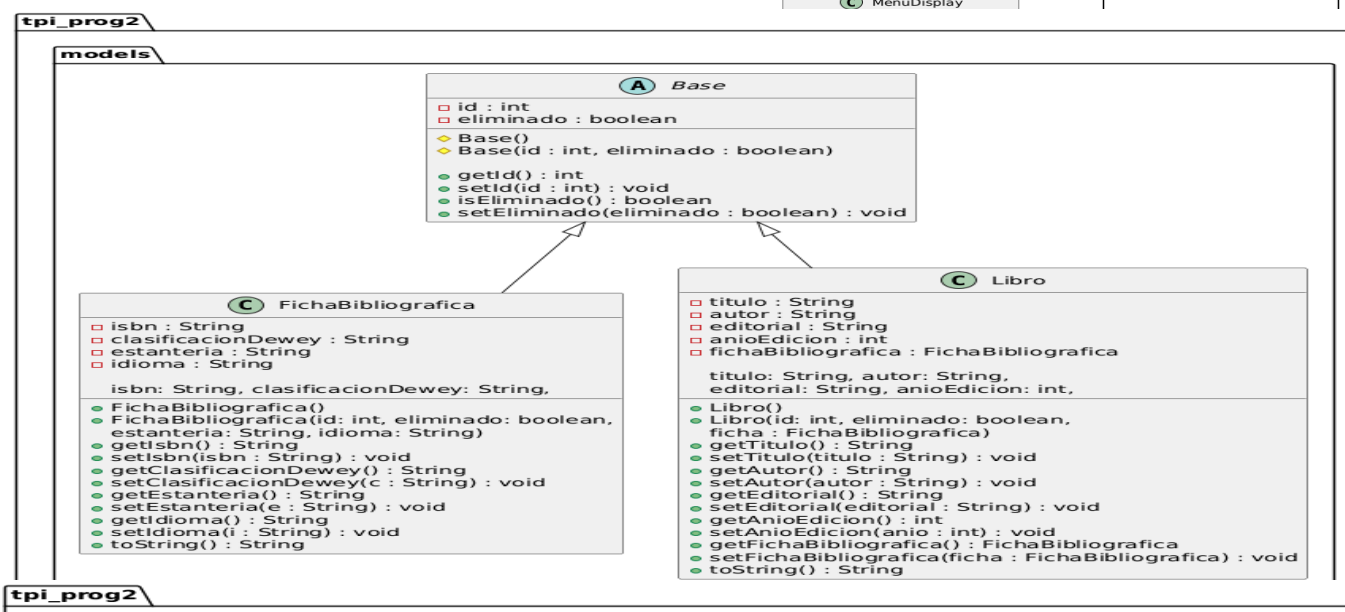
Se decidió usar una **clave foránea única** en la tabla **Libro** que referencie a la clave primaria de **FichaBibliografica**.

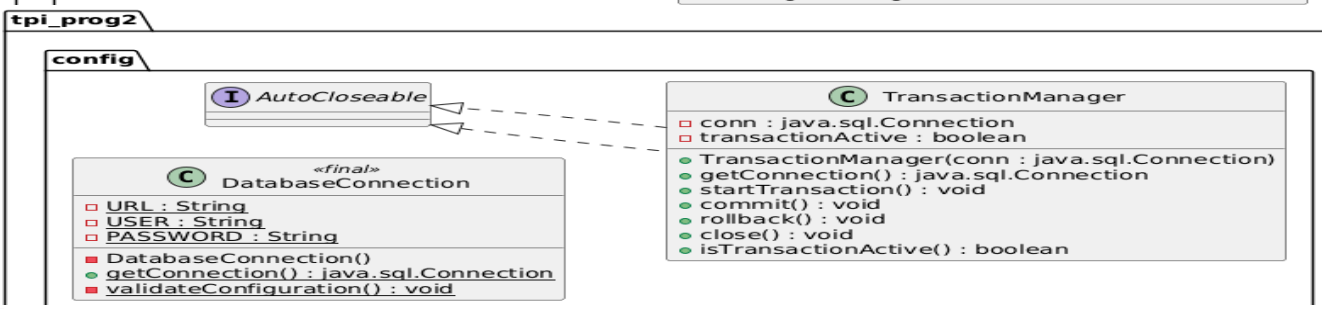
- **Tabla en BD:**
  - Libro: id (PK), titulo, autor, ..., fichaBibliografica_id (FK a FichaBibliografica.id, con restricción **UNIQUE**).
- **Justificación de la Decisión:**
  - **Simplicidad en la Entidad "A" (Libro):** Permite que la entidad Libro mantenga su propia clave primaria (id) independiente de FichaBibliografica.
  - **Independencia de PK:** No obliga a que el id del libro sea el mismo que el id de la ficha (como sucedería con una PK compartida), lo que facilita la gestión y posible creación de cada entidad por separado antes de vincularlas.
  - **Garantía de Unicidad:** La restricción **UNIQUE** impuesta sobre el campo de la FK (fichaBibliografica_id) asegura el cumplimiento de la relación 1:1: un libro apunta a una sola ficha, y una ficha solo puede ser referenciada por un único libro.

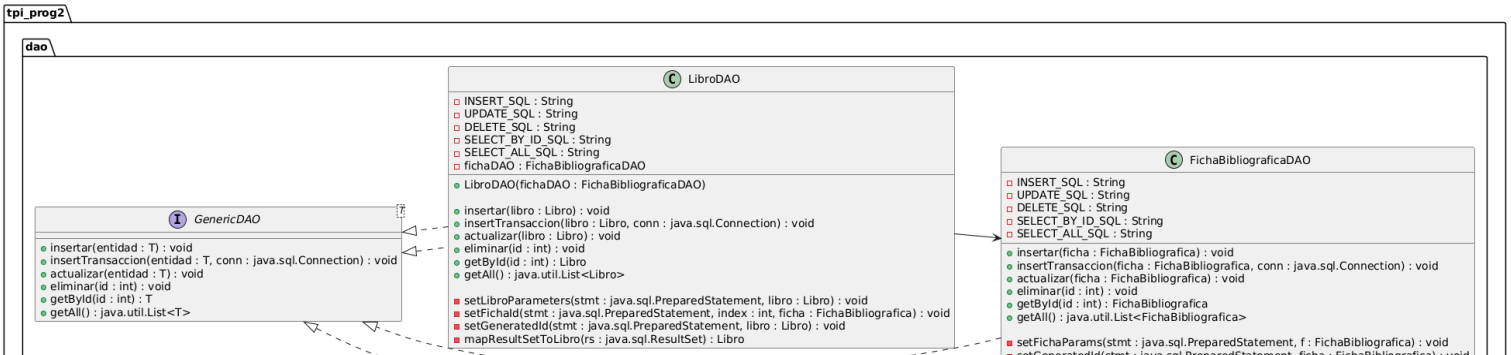
TECNICATURA UNIVERSITARIA
EN PROGRAMACIÓN
A DISTANCIA

UTN
UNIVERSIDAD TECNOLÓGICA NACIONAL









4. Arquitectura por Capas (Responsabilidades de cada Paquete)

El proyecto utiliza una arquitectura de **cinco capas lógicas** bien definidas, implementadas a través de los siguientes paquetes Java:

tpi_prog2.models (Capa de Entidades / Modelo)

- **Responsabilidad:** Contiene la definición de las **entidades de negocio** y las estructuras de datos principales.
- **Elementos:**
  - Base.java: Clase base que contiene atributos comunes como id y eliminado.
  - Libro.java y FichaBibliografica.java: Clases que representan las entidades del dominio.

tpi_prog2.dao (Capa de Acceso a Datos)

- **Responsabilidad:** Manejar la **persistencia** de los datos. Se encarga de la comunicación directa con la base de datos (BD), realizando operaciones **CRUD** (*Create, Read, Update, Delete*). Esta capa **no contiene lógica de negocio**.
- **Elementos:**
  - GenericDAO.java: Interfaz o clase abstracta para operaciones comunes de persistencia.
  - LibroDAO.java y FichaBibliograficaDAO.java: Implementaciones específicas para manipular las tablas correspondientes.

tpi_prog2.service (Capa de Lógica de Negocio / Servicios)

- **Responsabilidad:** Contiene toda la **lógica de negocio**, las **validaciones** complejas y la orquestación de las operaciones. Utiliza el DAO para obtener o guardar datos, y puede manejar **transacciones** que involucren múltiples operaciones de persistencia.
- **Elementos:**
  - GenericService.java: Interfaz o clase abstracta para operaciones de servicio comunes.
  - LibroServiceImpl.java y FichaBibliograficaServiceImpl.java: Implementaciones donde se encuentran las reglas de negocio del dominio.

tpi_prog2.config (Capa de Configuración y Utilidades de BD)

- **Responsabilidad:** Encapsular la configuración técnica y la gestión de la conexión a la base de datos y las transacciones.
- **Elementos:**
  - DatabaseConnection.java: Gestiona la conexión física a la BD (credenciales, URL).
  - TransactionManager.java: Clase clave para iniciar, confirmar o revertir (commit/rollback) operaciones transaccionales.

tpi_prog2.main (Capa de Presentación / Interfaz de Usuario)

- **Responsabilidad:** Actuar como el punto de entrada de la aplicación y manejar la **interfaz de usuario (UI)** en este caso, una interfaz de consola/menú.
- **Elementos:**
  - MainMenu.java, MenuHandler.java, MenuDisplay.java: Lógica de interacción y manejo del flujo de la aplicación.
  - Main.java: El método principal de ejecución de la aplicación.
  - TestConexion.java: Clase utilitaria para verificar la configuración inicial.

5. Persistencia

5.1. Estructura de la Base de Datos

El modelo relacional se compone de dos tablas principales que implementan la relación **uno a uno (1:1)** elegida en el diseño.

Tabla	Clave Primaria (PK)	Clave Foránea (FK)	Restricciones Clave
FichaBibliografica	id_ficha (INT, AUTO_INCREMENT)	N/A	isbn es **UNIQUE**.
Libro	id_libro (INT, AUTO_INCREMENT)	id_ficha (referencia a FichaBibliografica)	id_ficha es **UNIQUE** (para garantizar la relación 1:1).

5.2. Orden de Operaciones y Transacciones

Las operaciones que modifican tanto un Libro como su FichaBibliografica requieren **transacciones** para asegurar la **Atomicidad** (ACID): si falla la inserción de cualquiera de los dos, toda la operación debe revertirse.

5.3. Ubicación del Commit/Rollback

El manejo transaccional (commit/rollback) se centraliza en la capa de **Servicio** (Lógica de Negocio), utilizando la clase de configuración TransactionManager.

- **¿Dónde se hace commit/rollback?**
  - En la clase de servicio que orquesta la operación compuesta **LibroServiceImpl** (ya que el Libro es la entidad principal).
- **Mecanismo de Control:**
  - La clase TransactionManager maneja la conexión, estableciendo setAutoCommit(false) al iniciar la transacción y ejecutando **conn.commit()** o **conn.rollback()** en caso de éxito o fallo, respectivamente.

- El patrón try-with-resources (AutoCloseable) en TransactionManager asegura que, incluso si hay una excepción no manejada, el rollback() se ejecute al cerrar el recurso, evitando dejar la BD en un estado inconsistente.

6. Validaciones y Reglas de Negocio

6.1. Validaciones (Consistencia del Dato)

Las validaciones aseguran que los datos mínimos necesarios estén presentes y cumplan con los requisitos básicos de formato:

Entidad	Campo	Regla / Validación Implementada	Ubicación en el Código
Libro	titulo	**No puede ser nulo** o vacío (trim().isEmpty()).	LibroServiceImpl.validateLibro()
FichaBibliografica	isbn	**No puede ser nulo** o vacío (isBlank()).	FichaBibliograficaServiceImpl.validateFicha()
FichaBibliografica	idioma	**No puede ser nulo** o vacío (isBlank()).	FichaBibliograficaServiceImpl.validateFicha()
General	id	Debe ser **mayor a 0** para operaciones de actualización y eliminación.	Métodos actualizar y eliminar en ambos Services.

6.2. Reglas de Negocio (Lógica Operativa)

Las reglas de negocio más importantes se centran en la gestión de la relación **1:1** de forma transaccional, garantizando la integridad de la base de datos a nivel de lógica de negocio.

1. **Atomicidad de Inserción/Actualización (Regla Transaccional):**
   - **Regla:** La inserción o actualización de un **Libro** y su **Ficha Bibliográfica** asociada deben ser una operación **atómica**.

- **Implementación:** El método insertar (y actualizar) de LibroServiceImpl centraliza la lógica:
  - Obtiene una conexión con autoCommit = false.
  - Llama a los métodos transaccionales de FichaBibliograficaService y LibroDAO, pasando la misma conexión.
  - Llama a **conn.commit()** solo si ambas operaciones fueron exitosas.
  - Ejecuta **conn.rollback()** en el bloque catch si alguna operación falla.

2. **Gestión de Entidad Dependiente (Ficha):**
   - **Regla:** Si se inserta un nuevo **Libro**, se debe **insertar o actualizar primero** su **Ficha Bibliográfica**.
   - **Implementación:** Dentro de LibroServiceImpl.insertar(), se verifica si la ficha asociada ya tiene un ID asignado (getId() == 0). Si no lo tiene, se inserta una **nueva** ficha (fichaService.insertar(..., conn)); si ya tiene ID, se **actualiza**. Esto asegura que la ficha exista antes de que el libro la referencie.
3. **Restricción de Integridad (Unicidad 1:1):**
   - **Regla:** Un Libro no puede vincularse a una Ficha Bibliográfica que ya está en uso por otro libro.
   - **Implementación:** Esta regla es impuesta por el diseño de la base de datos (Sección 5) a través de la restricción **UNIQUE** en el campo id_ficha de la tabla Libro.

7)

```
-=-=-=-=-= LIBROS -=-=-=-=-=
1. Crear libro
2. Listar libros
3. Actualizar libro
4. Eliminar libro

-=-=-=-=-= FICHAS -=-=-=-=-=
5. Crear ficha bibliogr�fica
6. Listar fichas bibliogr�ficas
7. Actualizar ficha bibliogr�fica por ID
8. Eliminar ficha bibliogr�fica por ID

-=-=-=-=-= OTROS -=-=-=-=-=
9. Actualizar ficha bibliogr�fica por ID de libro
10. Eliminar ficha bibliogr�fica por ID de libro
0. SALIR
Ingrese una opci�n:
```

```sql
-- Fichas
INSERT INTO FichaBibliografica (isbn, clasificacionDewey, estanteria, idioma)
VALUES
('9780140449136', '873.1', 'E12', 'Español'),
('9789500750124', '821.1', 'B03', 'Español'),
('9780307474278', '823.9', 'F21', 'Inglés'),
('9789876123456', '500.2', 'C19', 'Español'),
('9788497592208', '863.3', 'D07', 'Español'),
('9788420674170', '840.9', 'E01', 'Francés'),
('9780307277671', '813.5', 'A14', 'Inglés');

-- Libros
INSERT INTO Libro (titulo, autor, editorial, anioEdicion, id_ficha)
VALUES
('La Odisea', 'Homero', 'Alianza', 2010, 1),
('El Quijote', 'Miguel de Cervantes', 'Santillana', 2005, 2),
('1984', 'George Orwell', 'Penguin Books', 2013, 3),
('Breves respuestas a las grandes preguntas', 'Stephen Hawking', 'Crítica', 2018, 4
('El Principito', 'Antoine de Saint-Exupéry', 'Debolsillo', 2008, 5),
('Madame Bovary', 'Gustave Flaubert', 'Alianza', 2012, 6),
('The Road', 'Cormac McCarthy', 'Vintage', 2006, 7);
```

8. Conclusiones y Mejoras Futuras

8.1. Conclusiones del Trabajo

El proyecto logró implementar un sistema de gestión bibliotecaria basado en una arquitectura sólida de **capas de servicio y persistencia**, centralizando la lógica en la capa `Service`.

- **Integridad y Atomicidad:** El mayor logro técnico fue la correcta implementación del modelo transaccional en `LibroServiceImpl`. Al utilizar una única `Connection` compartida y gestionada con `setAutoCommit(false)`, se garantizó la **atomicidad** en la inserción de **Libro** y **Ficha Bibliográfica**, evitando inconsistencias de datos en la base. El manejo explícito de `commit()` y `rollback()` centralizó la responsabilidad de la integridad de la base de datos en la capa de negocio.
- **Diseño 1:1 Efectivo:** La elección de la relación **uno a uno** implementada mediante una **Clave Foránea Única** (`id_ficha` con restricción UNIQUE en la tabla `Libro`) demostró ser un método limpio y eficaz para separar el dato descriptivo (`FichaBibliografica`) del dato operativo (`Libro`).
- **Separación de Responsabilidades:** La arquitectura por capas (*models, dao, service, config*) se mantuvo rigurosa, permitiendo que cada componente se enfocara en su tarea principal (ej: DAO solo accede a datos, `Service` aplica la lógica).

8.2. Mejoras Futuras

Para evolucionar el sistema, se proponen las siguientes mejoras:

1. **Implementación de Baja Lógica Completa:** Actualmente, la baja lógica del libro solo marca un *flag*. La mejora implicaría implementar la **Regla de Negocio** de restricción: un libro con un **préstamo activo** no puede ser marcado como eliminado.
2. **Manejo de Préstamos:** Introducir una nueva entidad (`Préstamo`) con relaciones **uno a muchos** (`Libro` a `Préstamo`), lo que obligaría a reevaluar y ampliar la lógica transaccional para operaciones como realizar un préstamo o una devolución.
3. **Refactorización del Transaccional:** Si bien el manejo de la conexión en `LibroServiceImpl` funciona, una mejora sería migrar la gestión transaccional al patrón **AOP (Programación Orientada a Aspectos)** o usar **anotaciones de *frameworks*** (como Spring) para centralizar aún más el `commit/rollback` y limpiar el código de la capa `Service` de la gestión explícita de `try/catch/finally`.
4. **Validación de Formato:** Agregar validaciones más robustas a campos como el ISBN (ej: usando expresiones regulares para verificar el formato ISBN-13) o el año de edición (no posterior al año actual).

Link al video de youtube: https://youtu.be/ssby3XFB6Kw

Link al repositorio de github: https://github.com/NicolasArrastia/TPI_Prog2