Sistema de Gestión de Biblioteca

- Programacion II Virtual
- Nombre: Jose Andres Flores Barco
- CARNE: 0910-24-25339
- Repositorio GitHub

Este proyecto es un **sistema de gestión de biblioteca** desarrollado en **Java** con conexión a **SQL Server (JDBC)**. Permite administrar **usuarios**, **materiales**, **préstamos** y **devoluciones** de manera automatizada.

Características

- Registro y consulta de usuarios (estudiantes y profesores).
- Administración de materiales (libros, revistas, tesis).
- Control de préstamos con fechas de vencimiento.
- Cálculo automático de multas por devoluciones tardías.
- Registro de devoluciones y actualización de disponibilidad.

Estructura del Proyecto

```
src/
├── model/
│ ├── User.java
│ ├── Student.java
│ ├── Professor.java
│ ├── Material.java
│ ├── Book.java
│ ├── Journal.java
│ ├── Thesis.java
│ ├── Loan.java
│ └── Return.java
├── dao/
│ ├── UserDAO.java
│ ├── MaterialDAO.java
│ ├── LoanDAO.java
│ ├── ReturnDAO.java
│ └──ConnectionDB.java
├── controller/
│ └── LibraryController.java
├── service/
│ └── ConsoleMenu.java
├── app/
│ └── Main.java
```

▧ Documentación

Esta sección describe a detalle el funcionamiento interno del sistema, su arquitectura, clases principales, consultas utilizadas y flujo de operaciones. Su propósito es servir como referencia técnica para desarrolladores que deseen comprender, mantener o extender el proyecto.

📐 Arquitectura General

El sistema está diseñado bajo una arquitectura **MVC simplificada**, donde:

- **Model** → Clases como `User`, `Student`, `Professor`, `Material`, `Book`, `Journal`, `Thesis`, `Loan`, `Return`.
- **DAO (Data Access Object)** → Maneja la conexión con SQL Server: `UserDAO`, `MaterialDAO`, `LoanDAO`, `ReturnDAO`
- **Controller** → Coordina la lógica del sistema: `LibraryController`
- **Database** → Estructura en SQL Server compuesta por las tablas `Users`, `Materials`, `Loans`, `Returns`.

🧱 Clases del Modelo

User (Abstract)

Clase base para todos los usuarios.

- `userId`
- `name`
- `email`
- `userType`

Herencia:

- `Student`
- `Professor`

Material (Abstract)

Clase base para los recursos disponibles.

- `materialId`
- `title`
- `author`
- `publicationDate`
- `materialType`
- `totalQuantity`
- `availableQuantity`

Herencia:

- `Book`
- `Journal`

- `Thesis`

Loan

Representa un préstamo:

- `loanId`
- `userId`
- `materialId`
- `loanDate`
- `dueDate`
- `returnDate`
- `fine`

Return

Registra las devoluciones asociadas a préstamos:

- `returnId`
- `loanId`
- `returnDate`
- `fine`

🗄 DAO (Data Access Object)

Los DAO encapsulan toda la lógica SQL y acceso a la base de datos gracias a JDBC.

📌 UserDAO

- `getUserById(int userId)`
- `createUser(User user)`
- `getAllUsers()`

📌 MaterialDAO

- `getMaterialById(int id)`
- `updateAvailability(int id, int newQuantity)`
- `getAllMaterials()`

📌 LoanDAO

- `createLoan(Loan loan)`
- `getLoanById(int id)`
- `updateReturn(int loanId, double fine)`

📌 ReturnDAO

- `registerReturn(Return returnObj)`
- `getAllReturns()`

🔁 Flujo de Préstamo

```
Usuario solicita material
        ↓
Verificar disponibilidad
        ↓
Registrar préstamo en `Loans`
        ↓
Reducir `available_quantity` en `Materials`
        ↓
Mostrar fecha de devolución
```

🗄 Configuración de la Base de Datos

1️⃣ Crear Base de Datos y Tablas

```
-- Crear la base de datos
CREATE DATABASE LibraryDB;
GO

USE LibraryDB;
GO

-- ============================================
-- TABLA: Usuarios (Estudiantes y Profesores)
-- ============================================
CREATE TABLE Users (
    user_id INT IDENTITY(1,1) PRIMARY KEY,
    name NVARCHAR(100) NOT NULL,
    email NVARCHAR(100) UNIQUE NOT NULL,
    user_type NVARCHAR(20) CHECK (user_type IN ('student', 'professor')) NOT NULL,
    registration_date DATETIME DEFAULT GETDATE()
);
GO

-- ============================================
-- TABLA: Materiales (Libros, Revistas, Tesis)
-- ============================================
CREATE TABLE Materials (
    material_id INT IDENTITY(1,1) PRIMARY KEY,
    title NVARCHAR(200) NOT NULL,
    author NVARCHAR(100),
    material_type NVARCHAR(20) CHECK (material_type IN ('book', 'journal',
'thesis')) NOT NULL,
    total_quantity INT NOT NULL CHECK (total_quantity >= 0),
    available_quantity INT NOT NULL CHECK (available_quantity >= 0),
    publication_date DATE
);
GO

-- ============================================
-- TABLA: Pr�stamos
-- ============================================
CREATE TABLE Loans (
    loan_id INT IDENTITY(1,1) PRIMARY KEY,
    user_id INT NOT NULL,
    material_id INT NOT NULL,
    loan_date DATETIME DEFAULT GETDATE(),
    due_date DATE NOT NULL,
    return_date DATE NULL,
    fine DECIMAL(8,2) DEFAULT 0 CHECK (fine >= 0),
    CONSTRAINT FK_Loans_Users FOREIGN KEY (user_id) REFERENCES Users(user_id),
    CONSTRAINT FK_Loans_Materials FOREIGN KEY (material_id) REFERENCES
Materials(material_id)
);
GO

-- ============================================
-- TABLA: Devoluciones
-- ============================================
CREATE TABLE Returns (
    return_id INT IDENTITY(1,1) PRIMARY KEY,
    loan_id INT NOT NULL,
    return_date DATETIME DEFAULT GETDATE(),
    fine DECIMAL(8,2) DEFAULT 0 CHECK (fine >= 0),
    CONSTRAINT FK_Returns_Loans FOREIGN KEY (loan_id) REFERENCES Loans(loan_id)
);
GO
```

2 Datos de Ejemplo

```
-- ============================================
-- INSERTS: Usuarios (Estudiantes y Profesores)
-- ============================================
INSERT INTO Users (name, email, user_type)
VALUES
('Juan P�rez', 'juan.perez@universidad.edu', 'student'),
('Mar�a L�pez', 'maria.lopez@universidad.edu', 'student'),
('Carlos G�mez', 'carlos.gomez@universidad.edu', 'student'),
('Ana Torres', 'ana.torres@universidad.edu', 'professor'),
('Luis Rodr�guez', 'luis.rodriguez@universidad.edu', 'professor');
GO

-- ============================================
-- INSERTS: Materiales (Libros, Revistas, Tesis)
-- ============================================
INSERT INTO Materials (title, author, material_type, total_quantity,
available_quantity, publication_date)
VALUES
('El Quijote', 'Miguel de Cervantes', 'book', 5, 5, '1605-01-01'),
('Cien años de soledad', 'Gabriel García Márquez', 'book', 4, 4, '1967-05-30'),
('Revista Científica de Tecnología', 'Varios autores', 'journal', 10, 10, '2024-
01-01'),
('Revista de Historia Moderna', 'Instituto de Historia', 'journal', 8, 8, '2023-
06-15'),
('Tesis sobre Energías Renovables', 'José Martínez', 'thesis', 2, 2, '2022-09-
01'),
('Programación en Java', 'James Gosling', 'book', 6, 6, '2020-03-01'),
('Introducción a la Inteligencia Artificial', 'Andrew Ng', 'book', 5, 5, '2018-06-
01'),
('Avances en Medicina 2025', 'OMS', 'journal', 12, 12, '2025-01-15'),
('Energía Solar en el Siglo XXI', 'Laura Herrera', 'thesis', 3, 3, '2023-08-20'),
('Revista de Computación Cuántica', 'IBM Research', 'journal', 7, 7, '2024-10-
05');
GO

-- ============================================
-- INSERTS: Pr�stamos
-- ============================================
INSERT INTO Loans (user_id, material_id, loan_date, due_date, fine)
VALUES
(1, 1, GETDATE(), DATEADD(DAY, 7, GETDATE()), 0),   -- Juan presta El Quijote
(2, 2, GETDATE(), DATEADD(DAY, 7, GETDATE()), 0),   -- María presta Cien años de
soledad
(3, 3, GETDATE(), DATEADD(DAY, 5, GETDATE()), 0),   -- Carlos presta Revista
Científica
(4, 4, GETDATE(), DATEADD(DAY, 10, GETDATE()), 0),  -- Ana presta Revista Historia
Moderna
(5, 5, GETDATE(), DATEADD(DAY, 15, GETDATE()), 0),  -- Luis presta Tesis Energías
Renovables
(1, 6, GETDATE(), DATEADD(DAY, 10, GETDATE()), 0),  -- Juan presta Programación en
Java
(2, 7, GETDATE(), DATEADD(DAY, 10, GETDATE()), 0),  -- María presta Inteligencia
Artificial
(3, 8, GETDATE(), DATEADD(DAY, 5, GETDATE()), 0),   -- Carlos presta Avances en
Medicina
(4, 9, GETDATE(), DATEADD(DAY, 12, GETDATE()), 0),  -- Ana presta Energía Solar
(5, 10, GETDATE(), DATEADD(DAY, 8, GETDATE()), 0);  -- Luis presta Computación
Cuántica
GO

-- ============================================
-- INSERTS: Devoluciones
-- ============================================
INSERT INTO Returns (loan_id, return_date, fine)
VALUES
(1, DATEADD(DAY, 6, GETDATE()), 0),        -- Juan devolvió antes del vencimiento
(2, DATEADD(DAY, 10, GETDATE()), 6.00),    -- María devolvió 3 días tarde
(3, DATEADD(DAY, 5, GETDATE()), 0),        -- Carlos a tiempo
(4, DATEADD(DAY, 13, GETDATE()), 6.00),    -- Ana 3 días tarde
(5, DATEADD(DAY, 14, GETDATE()), 0),       -- Luis antes de tiempo
```

```
(6, DATEADD(DAY, 12, GETDATE()), 4.00),   -- Juan 2 días tarde
(7, DATEADD(DAY, 9, GETDATE()), 0),       -- María a tiempo
(8, DATEADD(DAY, 8, GETDATE()), 6.00),    -- Carlos 3 días tarde
(9, DATEADD(DAY, 13, GETDATE()), 2.00),   -- Ana 1 día tarde
(10, DATEADD(DAY, 7, GETDATE()), 0);      -- Luis a tiempo
GO
```

🧠 Clases Principales

App

Main.java

```
package com.library.app;

import com.library.service.ConsoleMenu;
import java.io.PrintStream;
import java.nio.charset.StandardCharsets;

/**
 *
 * @author josef
 */
public class Main {

    public static void main(String[] args) {
        System.setOut(new PrintStream(System.out, true, StandardCharsets.UTF_8));
        ConsoleMenu.start();
    }
}
```

Controller

LibraryController.java

```
package com.library.controller;

import com.library.dao.LoanDAO;
import com.library.dao.MaterialDAO;
import com.library.dao.ReturnDAO;
import com.library.dao.UserDAO;
import com.library.model.Loan;
import com.library.model.Material;
import com.library.model.Return;
import com.library.model.User;
import java.time.LocalDate;
import java.util.List;
```

```
/**
 *
 * @author josef
 */
public class LibraryController {

    private final UserDAO userDAO;
    private final MaterialDAO materialDAO;
    private final LoanDAO loanDAO;
    private final ReturnDAO returnDAO;

    public LibraryController() {
        this.userDAO = new UserDAO();
        this.materialDAO = new MaterialDAO();
        this.loanDAO = new LoanDAO();
        this.returnDAO = new ReturnDAO();
    }

    // ================================
    //          USUARIOS
    // ================================
    public void registerUser(User user) {
        try {
            userDAO.addUser(user);
            System.out.println("✅ Usuario registrado: " + user.getName());
        } catch (Exception e) {
            System.err.println("❌ Error al registrar usuario: " +
e.getMessage());
        }
    }

    public List<User> listUsers() {
        try {
            return userDAO.getAllUsers();
        } catch (Exception e) {
            System.err.println("❌ Error al obtener usuarios: " +
e.getMessage());
            return null;
        }
    }

    // ================================
    //          MATERIALES
    // ================================
    public void registerMaterial(Material material) {
        try {
            materialDAO.addMaterial(material);
            System.out.println("✅ Material registrado: " + material.getTitle());
        } catch (Exception e) {
            System.err.println("❌ Error al registrar material: " +
e.getMessage());
        }
    }
    public List<Material> listMaterials() {
        try {
            return materialDAO.getAllMaterials();
        } catch (Exception e) {
            System.err.println("❌ Error al obtener materiales: " +
e.getMessage());
            return null;
        }
    }

    // ================================
    //         PRÉSTAMOS
    // ================================
    public void makeLoan(int userId, int materialId) {
        try {
            User user = userDAO.getUserById(userId);
            Material material = materialDAO.getMaterialById(materialId);

            if (user == null || material == null) {
                System.err.println("❌ Usuario o material no encontrado.");
                return;
            }

            if (material.getAvailableQuantity() <= 0) {
                System.err.println("❌ No hay ejemplares disponibles.");
                return;
            }

            int days = material.getMaxLoanDays();
            LocalDate dueDate = LocalDate.now().plusDays(days);

            Loan loan = new Loan(userId, materialId, dueDate);
            loanDAO.addLoan(loan);

            // Actualiza cantidad disponible
            materialDAO.updateAvailability(materialId,
material.getAvailableQuantity() - 1);

            System.out.println("✅ Préstamo registrado correctamente. Entregar
antes del: " + dueDate);

        } catch (Exception e) {
            System.err.println("❌ Error al registrar préstamo: " +
e.getMessage());
        }
    }

    public List<Loan> listLoans() {
        try {
            return loanDAO.getAllLoans();
        } catch (Exception e) {
            System.err.println("❌ Error al obtener préstamos: " +
e.getMessage());
            return null;
        }
    }

    // ================================
    //         DEVOLUCIONES
    // ================================
    public void returnMaterial(int loanId) {
        try {
            Loan loan = loanDAO.getLoanById(loanId);
            if (loan == null) {
                System.err.println("❌ No se encontró el préstamo con ID " +
loanId);
                return;
            }

            LocalDate today = LocalDate.now();
            double fine = 0;

            // Calcular multa si hay retraso
            if (today.isAfter(loan.getDueDate())) {
                long daysLate =
java.time.temporal.ChronoUnit.DAYS.between(loan.getDueDate(), today);
                fine = daysLate * 2.0; // Q2 por día de retraso
                System.out.println("⚠ Devolución con retraso: " + daysLate + "
días. Multa total: Q" + fine);
            }

            // 1️⃣ Actualizar registro en Loans
            loanDAO.updateReturn(loanId, fine);

            // 2️⃣ Insertar registro en Returns
            Return ret = new Return(loanId, today, fine);
            returnDAO.addReturn(ret);

            // 3️⃣ Actualizar cantidad disponible del material
            Material material = materialDAO.getMaterialById(loan.getMaterialId());
            if (material != null) {
                materialDAO.updateAvailability(material.getMaterialId(),
material.getAvailableQuantity() + 1);
            }

            System.out.println("✅ Devolución registrada correctamente.");

        } catch (Exception e) {
            System.err.println("❌ Error al procesar devolución: " +
e.getMessage());
        }
    }

    public List<Return> listReturns() {
        try {
            return returnDAO.getAllReturns();
        } catch (Exception e) {
            System.err.println("❌ Error al obtener devoluciones: " +
e.getMessage());
            return null;
        }
    }
}
```

🧮 Cálculo de Multa

```
if (today.isAfter(loan.getDueDate())) {
 long daysLate = DAYS.between(loan.getDueDate(), today);
 fine = daysLate * 2.0;
}
```

Service

ConsoleMenu.java

```
package com.library.service;

import com.library.controller.LibraryController;
import com.library.model.Loan;
import com.library.model.Material;
import com.library.model.Professor;
import com.library.model.Return;
import com.library.model.Student;
import com.library.model.User;
import java.util.List;
import java.util.Scanner;

/**
 *
 * @author josef
 */
public class ConsoleMenu {

    private static final Scanner scanner = new Scanner(System.in);
    private static final LibraryController controller = new LibraryController();

    // ===========================================================
    // ===================== MÉTODOS VISUALES ====================
    // ===========================================================
    /**
     * Limpia la consola de forma compatible con Windows, macOS y Linux.
     */
    public static void clearConsole() {
        try {
            if (System.getProperty("os.name").contains("Windows")) {
                new ProcessBuilder("cmd", "/c",
"cls").inheritIO().start().waitFor();
            } else {
                System.out.print("\033[H\033[2J");
                System.out.flush();
            }
        } catch (Exception e) {
            System.out.println("No se pudo limpiar la consola");
        }
    }

    /**
     * Imprime un encabezado con formato.
     */
    public static void printHeader(String title, String... infoLines) {
        String line = "=".repeat(150);
        System.out.printf("\n+ %-150s +\n", line);
        System.out.printf("| %-150s |\n", title.toUpperCase());
        System.out.printf("+ %-150s +\n", line);

        for (String info : infoLines) {
            System.out.printf("| %-150s |\n", info);
        }

        System.out.printf("+ %-150s +\n", "-".repeat(150));
    }

    /**
     * Imprime un menú con opciones dinámicas y retorna la opción elegida.
     */
    public static int showMenu(String menuTitle, List<String> options) {
        String line = "=".repeat(150);
        printHeader(menuTitle);

        for (int i = 0; i < options.size(); i++) {
            System.out.printf("| [%d] %-" + (150 - 4) + "s |\n", i + 1,
options.get(i));
        }

        System.out.printf("| %-150s |\n", "[0] Salir");
        System.out.printf("+ %-150s +\n", line);
        System.out.print("\nSeleccione una opción: ");

        int choice = -1;
        try {
            choice = Integer.parseInt(scanner.nextLine());
        } catch (NumberFormatException e) {
            System.out.println("Entrada inválida. Intente de nuevo.");
        }

        return choice;
    }

    /**
     * Imprime una tabla simple con encabezados y filas.
     */
    public static void printTable(String[] headers, List<String[]> rows) {
        int[] widths = new int[headers.length];
        for (int i = 0; i < headers.length; i++) {
            widths[i] = headers[i].length();
        }

        // Calcular ancho máximo por columna
        for (String[] row : rows) {
            for (int i = 0; i < row.length; i++) {
                widths[i] = Math.max(widths[i], row[i].length());
            }
        }

        printSeparator(widths);
        printRow(headers, widths);
        printSeparator(widths);

        for (String[] row : rows) {
            printRow(row, widths);
        }

        printSeparator(widths);
    }

    private static void printRow(String[] cells, int[] widths) {
        StringBuilder sb = new StringBuilder("|");
        for (int i = 0; i < cells.length; i++) {
            sb.append(" ").append(String.format("%-" + widths[i] + "s",
cells[i])).append(" |");
        }
        System.out.println(sb);
    }

    private static void printSeparator(int[] widths) {
        StringBuilder sb = new StringBuilder("+");
        for (int width : widths) {
            sb.append("-".repeat(width + 2)).append("+");
        }
        System.out.println(sb);
    }

    // ===========================================================
    // ===================== MENÚ PRINCIPAL ======================
    // ===========================================================
    public static void start() {
        clearConsole();
        printHeader(
                "PROYECTO FINAL - PROGRAMACIÓN II",
                "NOMBRE: José Andrés Flores Barco",
                "CARNE: 0910-24-25339",
                "SECCIÓN: Sábado Virtual"
        );
        List<String> options = List.of(
                "Manejo de usuarios",
                "Manejo de materiales",
                "Préstamos",
                "Devoluciones"
        );

        int choice;
        do {
            choice = showMenu("MENÚ PRINCIPAL", options);

            switch (choice) {
                case 1 ->
                    userMenu();
                case 2 ->
                    materialMenu();
                case 3 ->
                    loanMenu();
                case 4 ->
                    returnMenu();
                case 0 ->
                    System.out.println("Saliendo del sistema...");
                default ->
                    System.out.println("Opción no válida.");
            }

            if (choice != 0) {
                pauseAndClear();
            }
        } while (choice != 0);
    }

    // ===========================================================
    // ====================== SUBMENÚS ============================
    // ===========================================================
    private static void userMenu() {
        List<String> options = List.of(
                "Registrar usuario (Tipo Estudiante)",
                "Registrar usuario (Tipo Profesor)",
                "Listar usuarios"
        );

        int choice;
        do {
            choice = showMenu("GESTIÓN DE USUARIOS", options);

            switch (choice) {
                case 1 ->
                    registerStudent();
                case 2 ->
                    registerProfessor();
                case 3 -> {
                    List<User> users = controller.listUsers();
                    if (users.isEmpty()) {
                        System.out.println("No hay usuarios registrados.");
                    } else {
                        List<String[]> rows = new java.util.ArrayList<>();

                        for (User user : users) {
                            rows.add(new String[]{
                                String.valueOf(user.getUserId()),
                                user.getName(),
                                user.getEmail(),
                                user.getUserType()
                            });
                        }

                        printTable(
                                new String[]{"ID", "NOMBRE", "EMAIL", "TIPO DE
USUARIO"},
                                rows
                        );
                    }
                }
                case 0 ->
                    System.out.println("Volviendo al menú principal...");
                default ->
                    System.out.println("Opción no válida.");
            }

            if (choice != 0) {
                pauseAndClear();
            }
        } while (choice != 0);
    }

    private static void materialMenu() {
        List<String> options = List.of(
                "Registrar material",
                "Listar materiales",
                "Eliminar material"
        );

        int choice;
        do {
            choice = showMenu("GESTIÓN DE MATERIALES", options);

            switch (choice) {
                case 1 ->
                    registerMaterial();
                case 2 -> {
                    List<Material> materials = controller.listMaterials();

                    if (materials.isEmpty()) {
                        System.out.println("No hay Materiales (Libros, Revistas y
Thesis) registrados.");
                    } else {
                        List<String[]> rows = new java.util.ArrayList<>();

                        for (Material material : materials) {
                            rows.add(new String[]{
                                String.valueOf(material.getMaterialId()),
                                material.getTitle(),
                                material.getAuthor(),
                                material.getMaterialType(),
                                String.valueOf(material.getTotalQuantity()),
                                String.valueOf(material.getAvailableQuantity()),
                                String.valueOf(material.getPublicationDate())
                            });
                        }

                        printTable(
                                new String[]{"ID", "TITULO", "AUTOR", "TIPO DE
MATERIAL", "TOTAL", "DISPONIBLE", "FECHA DE PUBLICACION"},
                                rows
                        );
                    }
                }
                case 3 ->
                    deleteMaterial();
                case 0 ->
                    System.out.println("Volviendo al menú principal...");
                default ->
                    System.out.println("Opción no válida.");
            }

            if (choice != 0) {
                pauseAndClear();
            }
        } while (choice != 0);
    }

    private static void loanMenu() {
        List<String> options = List.of(
                "Registrar préstamo",
                "Listar préstamos"
        );

        int choice;
        do {
            choice = showMenu("GESTIÓN DE PRÉSTAMOS", options);

            switch (choice) {
                case 1 ->
                    registerLoan();
                case 2 -> {
                    List<Loan> loans = controller.listLoans();

                    if (loans.isEmpty()) {
                        System.out.println("No hay Prestamos registrados.");
                    } else {
                        List<String[]> rows = new java.util.ArrayList<>();

                        for (Loan loan : loans) {
                            rows.add(new String[]{
                                String.valueOf(loan.getLoanId()),
                                String.valueOf(loan.getUserId()),
                                String.valueOf(loan.getMaterialId()),
                                String.valueOf(loan.getLoanDate()),
                                String.valueOf(loan.getDueDate()),
                                String.valueOf(loan.getReturnDate()),
                                String.valueOf(loan.getFine())
                            });
                        }

                        printTable(
                                new String[]{"ID", "ID USUARIO", "ID MATERIAL ",
"FECHA PRESTAMO", "FECHA VENCIMIENTO", "FECHA RETORNO", "MULTA"},
                                rows
                        );
                    }
                }
                case 0 ->
                    System.out.println("Volviendo al menú principal...");
                default ->
                    System.out.println("Opción no válida.");
            }

            if (choice != 0) {
                pauseAndClear();
            }
        } while (choice != 0);
    }

    private static void returnMenu() {
        List<String> options = List.of(
                "Registrar devolución",
                "Listar devoluciones"
        );

        int choice;
        do {
            choice = showMenu("GESTIÓN DE DEVOLUCIONES", options);

            switch (choice) {
                case 1 ->
                    registerReturn();
                case 2 -> {
                    List<Return> returns = controller.listReturns();

                    if (returns.isEmpty()) {
                        System.out.println("No hay Retornos registrados.");
                    } else {
                        List<String[]> rows = new java.util.ArrayList<>();
                        for (Return return1 : returns) {
                            rows.add(new String[]{
                                String.valueOf(return1.getReturnId()),
                                String.valueOf(return1.getLoanId()),
                                String.valueOf(return1.getReturnDate()),
                                String.valueOf(return1.getFine()),});
                        }

                        printTable(
                                new String[]{"ID", "ID PRESTAMO", "FECHA DE
RETORNO ", "MULTA"},
                                rows
                        );
                    }
                }
                case 0 ->
                    System.out.println("Volviendo al menú principal...");
                default ->
                    System.out.println("Opción no válida.");
            }

            if (choice != 0) {
                pauseAndClear();
            }
        } while (choice != 0);
    }

    // ===========================================================
    // ================= FUNCIONALIDAD BÁSICA ====================
    // ===========================================================
    private static void registerStudent() {
        System.out.print("Ingrese nombre del usuario: ");
        String name = scanner.nextLine();
        System.out.print("Ingrese correo del usuario: ");
        String email = scanner.nextLine();

        Student student = new Student(name, email);
        controller.registerUser(student);

        System.out.println("✅ Usuario tipo estudiante registrado con éxito.");
    }

    private static void registerProfessor() {
        System.out.print("Ingrese nombre del usuario: ");
        String name = scanner.nextLine();
        System.out.print("Ingrese correo del usuario: ");
        String email = scanner.nextLine();

        Professor professor = new Professor(name, email);
        controller.registerUser(professor);

        System.out.println("✅ Usuario tipo profesor registrado con éxito.");
    }
```

```java
    private static void registerMaterial() {
        System.out.print("Ingrese título del material: ");
        String title = scanner.nextLine();
        System.out.print("Ingrese tipo de material: ");
        String type = scanner.nextLine();

        //controller.registerMaterial(title, type);
        System.out.println("✅ Material registrado correctamente.");
    }

    private static void deleteMaterial() {
        System.out.print("Ingrese ID del material a eliminar: ");
        int id = Integer.parseInt(scanner.nextLine());
        //controller.deleteMaterial(id);
    }

    private static void registerLoan() {
        System.out.print("Ingrese ID del usuario: ");
        int userId = Integer.parseInt(scanner.nextLine());
        System.out.print("Ingrese ID del material: ");
        int materialId = Integer.parseInt(scanner.nextLine());

        controller.makeLoan(userId, materialId);
        System.out.println("✅ Préstamo registrado correctamente.");
    }

    private static void registerReturn() {
        System.out.print("Ingrese ID del préstamo: ");
        int loanId = Integer.parseInt(scanner.nextLine());
        controller.returnMaterial(loanId);
        System.out.println("✅ Devolución registrada correctamente.");
    }

    private static void pauseAndClear() {
        System.out.println("\nPresione ENTER para continuar...");
        scanner.nextLine();
        clearConsole();
    }
}
```

Service

ConectionDB.java

```java
package com.library.dao;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.Properties;
import java.io.InputStream;
```

```
public class ConnectionDB {
    private static Connection connection;

    // Cargar configuración desde db.properties
    public static Connection getConnection() throws SQLException {
        if (connection == null || connection.isClosed()) {
            try (InputStream input = ConnectionDB.class.getClassLoader()
                    .getResourceAsStream("db.properties")) {

                Properties props = new Properties();
                props.load(input);

                String url = props.getProperty("db.url");
                String user = props.getProperty("db.user");
                String password = props.getProperty("db.password");

                connection = DriverManager.getConnection(url, user, password);
                System.out.println("✅ Conexión establecida con SQL Server.");
            } catch (Exception e) {
                System.err.println("❌ Error al conectar a la base de datos: " +
e.getMessage());
                throw new SQLException(e);
            }
        }
        return connection;
    }
}
```

LoanDAO.java

```
package com.library.dao;

import com.library.model.Loan;
import java.sql.Connection;
import java.sql.Date;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;

/**
 *
 * @author josef
 */
public class LoanDAO {

    public void addLoan(Loan loan) {
        String sql = "INSERT INTO Loans (user_id, material_id, loan_date,
due_date, fine) VALUES (?, ?, ?, ?, ?)";
        try (Connection conn = ConnectionDB.getConnection(); PreparedStatement
stmt = conn.prepareStatement(sql)) {

            stmt.setInt(1, loan.getUserId());
            stmt.setInt(2, loan.getMaterialId());
            stmt.setDate(3, Date.valueOf(loan.getLoanDate()));
            stmt.setDate(4, Date.valueOf(loan.getDueDate()));
            stmt.setDouble(5, loan.getFine());
            stmt.executeUpdate();
        } catch (SQLException e) {
            System.err.println("❌ Error al registrar préstamo: " +
e.getMessage());
        }
    }

    public List<Loan> getAllLoans() {
        List<Loan> loans = new ArrayList<>();
        String sql = "SELECT * FROM Loans";

        try (Connection conn = ConnectionDB.getConnection(); Statement stmt =
conn.createStatement(); ResultSet rs = stmt.executeQuery(sql)) {

            while (rs.next()) {
                Loan loan = new Loan(
                        rs.getInt("loan_id"),
                        rs.getInt("user_id"),
                        rs.getInt("material_id"),
                        rs.getDate("loan_date").toLocalDate(),
                        rs.getDate("due_date").toLocalDate(),
                        rs.getDate("return_date") != null ?
rs.getDate("return_date").toLocalDate() : null,
                        rs.getDouble("fine")
                );
                loans.add(loan);
            }
        } catch (SQLException e) {
            System.err.println("❌ Error al obtener préstamos: " +
e.getMessage());
        }

        return loans;
    }

    public Loan getLoanById(int loanId) {
        Loan loan = null;
        String query = "SELECT loan_id, user_id, material_id, loan_date, due_date,
return_date, fine "
                + "FROM Loans WHERE loan_id = ?";

        try (Connection conn = ConnectionDB.getConnection(); PreparedStatement
stmt = conn.prepareStatement(query)) {
```

```
            stmt.setInt(1, loanId);

            try (ResultSet rs = stmt.executeQuery()) {
                if (rs.next()) {
                    LocalDate loanDate = rs.getDate("loan_date").toLocalDate();
                    LocalDate dueDate = rs.getDate("due_date").toLocalDate();
                    LocalDate returnDate = null;

                    Date sqlReturn = rs.getDate("return_date");
                    if (sqlReturn != null) {
                        returnDate = sqlReturn.toLocalDate();
                    }

                    loan = new Loan(
                            rs.getInt("loan_id"),
                            rs.getInt("user_id"),
                            rs.getInt("material_id"),
                            loanDate,
                            dueDate,
                            returnDate,
                            rs.getDouble("fine")
                    );
                }
            }

        } catch (SQLException e) {
            System.err.println("❌ Error al obtener préstamo con ID " + loanId +
": " + e.getMessage());
        }

        return loan;
    }

    public void updateReturn(int loanId, double fine) {
        String query = "UPDATE Loans SET return_date = ?, fine = ? WHERE loan_id =
?";

        try (Connection conn = ConnectionDB.getConnection(); PreparedStatement
stmt = conn.prepareStatement(query)) {

            stmt.setDate(1, java.sql.Date.valueOf(LocalDate.now()));
            stmt.setDouble(2, fine);
            stmt.setInt(3, loanId);

            int rows = stmt.executeUpdate();

            if (rows > 0) {
                System.out.println("✅ Préstamo actualizado correctamente. ID: "
+ loanId);
            } else {
                System.out.println("⚠️ No se encontró el préstamo con ID: " +
loanId);
            }
        } catch (SQLException e) {
            System.err.println("❌ Error al actualizar devolución del préstamo: "
+ e.getMessage());
        }
    }

}
```

ReturnDAO.java

```
package com.library.dao;

import com.library.model.Return;
import java.sql.Connection;
import java.sql.Date;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;

/**
 *
 * @author josef
 */
public class ReturnDAO {

    public void addReturn(Return ret) {
        String sql = "INSERT INTO Returns (loan_id, return_date, fine) VALUES (?,
?, ?)";
        try (Connection conn = ConnectionDB.getConnection(); PreparedStatement
stmt = conn.prepareStatement(sql)) {

            stmt.setInt(1, ret.getLoanId());
            stmt.setDate(2, Date.valueOf(ret.getReturnDate()));
            stmt.setDouble(3, ret.getFine());
            stmt.executeUpdate();
        } catch (SQLException e) {
            System.err.println("❌ Error al registrar devolución: " +
e.getMessage());
        }
    }

    public List<Return> getAllReturns() {
        List<Return> returns = new ArrayList<>();
        String sql = "SELECT * FROM Returns";

        try (Connection conn = ConnectionDB.getConnection(); Statement stmt =
conn.createStatement(); ResultSet rs = stmt.executeQuery(sql)) {
```

```
            while (rs.next()) {
                Return ret = new Return(
                        rs.getInt("return_id"),
                        rs.getInt("loan_id"),
                        rs.getDate("return_date").toLocalDate(),
                        rs.getDouble("fine")
                );
                returns.add(ret);
            }

        } catch (SQLException e) {
            System.err.println("❌ Error al obtener devoluciones: " +
e.getMessage());
        }

        return returns;
    }
}
```

UserDAO.java

```
package com.library.dao;

import com.library.model.Professor;
import com.library.model.Student;
import com.library.model.User;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;

/**
 *
 * @author josef
 */
public class UserDAO {

    public void addUser(User user) {
        String sql = "INSERT INTO Users (name, email, user_type) VALUES (?, ?,
?)";
        try (Connection conn = ConnectionDB.getConnection(); PreparedStatement
stmt = conn.prepareStatement(sql)) {

            stmt.setString(1, user.getName());
            stmt.setString(2, user.getEmail());
            stmt.setString(3, user.getUserType());
            stmt.executeUpdate();
            System.out.println("✅ Usuario registrado: " + user.getName());
        } catch (SQLException e) {
            System.err.println("❌ Error al registrar usuario: " +
e.getMessage());
        }
    }

    public void addStudent(User user) {
    }

    public List<User> getAllUsers() {
        List<User> users = new ArrayList<>();
        String sql = "SELECT * FROM Users";

        try (Connection conn = ConnectionDB.getConnection(); Statement stmt =
conn.createStatement(); ResultSet rs = stmt.executeQuery(sql)) {

            while (rs.next()) {
                int id = rs.getInt("user_id");
                String name = rs.getString("name");
                String email = rs.getString("email");
                String type = rs.getString("user_type");

                User user;
                if ("student".equalsIgnoreCase(type)) {
                    user = new Student(id, name, email);
                } else {
                    user = new Professor(id, name, email);
                }
                users.add(user);
            }
        } catch (SQLException e) {
            System.err.println("❌ Error al obtener usuarios: " +
e.getMessage());
        }

        return users;
    }

    public User getUserById(int userId) {
        User user = null;
        String query = "SELECT user_id, name, email, user_type FROM Users WHERE
user_id = ?";

        try (Connection conn = ConnectionDB.getConnection(); PreparedStatement
stmt = conn.prepareStatement(query)) {

            stmt.setInt(1, userId);

            try (ResultSet rs = stmt.executeQuery()) {
                if (rs.next()) {
                    String type = rs.getString("user_type");

                    // Crear la subclase adecuada
                    switch (type.toLowerCase()) {
                        case "student" -> {
                            user = new Student(
                                    rs.getInt("user_id"),
                                    rs.getString("name"),
                                    rs.getString("email")
                            );
                        }
                        case "professor" -> {
                            user = new Professor(
                                    rs.getInt("user_id"),
                                    rs.getString("name"),
                                    rs.getString("email")
                            );
                        }
                        default -> {
                            System.err.println("⚠️ Tipo de usuario desconocido: "
+ type);
                        }
                    }
                }
            }

        } catch (SQLException e) {
            System.err.println("❌ Error al obtener el usuario con ID " + userId
+ ": " + e.getMessage());
        }

        return user;
    }

}
```

MaterialDAO.java

```
package com.library.dao;

import com.library.model.Book;
import com.library.model.Journal;
import com.library.model.Material;
import com.library.model.Thesis;
import java.sql.Connection;
import java.sql.Date;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;
```

```
/**
 *
 * @author josef
 */
public class MaterialDAO {

    public void addMaterial(Material material) {
        String sql = "INSERT INTO Materials (title, author, material_type,
total_quantity, available_quantity, publication_date) VALUES (?, ?, ?, ?, ?, ?)";
        try (Connection conn = ConnectionDB.getConnection(); PreparedStatement
stmt = conn.prepareStatement(sql)) {

            stmt.setString(1, material.getTitle());
            stmt.setString(2, material.getAuthor());
            stmt.setString(3, material.getMaterialType());
            stmt.setInt(4, material.getTotalQuantity());
            stmt.setInt(5, material.getAvailableQuantity());
            stmt.setDate(6, Date.valueOf(material.getPublicationDate()));

            stmt.executeUpdate();
            System.out.println("✅ Material agregado: " + material.getTitle());

        } catch (SQLException e) {
            System.err.println("❌ Error al agregar material: " +
e.getMessage());
        }
    }

    public List<Material> getAllMaterials() {
        List<Material> materials = new ArrayList<>();
        String sql = "SELECT * FROM Materials";

        try (Connection conn = ConnectionDB.getConnection(); Statement stmt =
conn.createStatement(); ResultSet rs = stmt.executeQuery(sql)) {

            while (rs.next()) {
                int id = rs.getInt("material_id");
                String title = rs.getString("title");
                String author = rs.getString("author");
                String type = rs.getString("material_type");
                int total = rs.getInt("total_quantity");
                int available = rs.getInt("available_quantity");
                LocalDate date = rs.getDate("publication_date").toLocalDate();

                Material material;
                switch (type.toLowerCase()) {
                    case "book":
                        material = new Book(id, title, author, total, available,
date);
                        break;
                    case "journal":
                        material = new Journal(id, title, author, total,
available, date);
                        break;
```

```java
                    default:
                        material = new Thesis(id, title, author, total, available,
date);
                        break;
                }
                materials.add(material);
            }

        } catch (SQLException e) {
            System.err.println("❌ Error al obtener materiales: " +
e.getMessage());
        }

        return materials;
    }

    public void updateAvailability(int materialId, int newQuantity) {
        String sql = "UPDATE Materials SET available_quantity = ? WHERE
material_id = ?";
        try (Connection conn = ConnectionDB.getConnection(); PreparedStatement
stmt = conn.prepareStatement(sql)) {

            stmt.setInt(1, newQuantity);
            stmt.setInt(2, materialId);
            stmt.executeUpdate();
        } catch (SQLException e) {
            System.err.println("❌ Error al actualizar disponibilidad: " +
e.getMessage());
        }
    }

    public Material getMaterialById(int materialId) {
        Material material = null;
        String query = "SELECT material_id, title, author, material_type,
total_quantity, available_quantity, publication_date "
                + "FROM Materials WHERE material_id = ?";

        try (Connection conn = ConnectionDB.getConnection(); PreparedStatement
stmt = conn.prepareStatement(query)) {

            stmt.setInt(1, materialId);

            try (ResultSet rs = stmt.executeQuery()) {
                if (rs.next()) {
                    String type = rs.getString("material_type").toLowerCase();
                    LocalDate publicationDate = null;

                    Date sqlDate = rs.getDate("publication_date");
                    if (sqlDate != null) {
                        publicationDate = sqlDate.toLocalDate();
                    }

                    // Crear la subclase adecuada según el tipo
                    switch (type) {
                        case "book" -> {
                            material = new Book(
                                    rs.getInt("material_id"),
                                    rs.getString("title"),
                                    rs.getString("author"),
                                    rs.getInt("total_quantity"),
                                    rs.getInt("available_quantity"),
                                    publicationDate
                            );
                        }
                        case "journal" -> {
                            material = new Journal(
                                    rs.getInt("material_id"),
                                    rs.getString("title"),
                                    rs.getString("author"),
                                    rs.getInt("total_quantity"),
                                    rs.getInt("available_quantity"),
                                    publicationDate
                            );
                        }
                        case "thesis" -> {
                            material = new Thesis(
                                    rs.getInt("material_id"),
                                    rs.getString("title"),
                                    rs.getString("author"),
                                    rs.getInt("total_quantity"),
                                    rs.getInt("available_quantity"),
                                    publicationDate
                            );
                        }
                        default -> {
                            System.err.println("⚠ Tipo de material desconocido:
" + type);
                        }
                    }
                }
            }

        } catch (SQLException e) {
            System.err.println("❌ Error al obtener material con ID " +
materialId + ": " + e.getMessage());
        }

        return material;
    }

}
```

Model

Book.java

```
package com.library.model;

import java.time.LocalDate;

/**
 *
 * @author josef
 */
public class Book extends Material {

    public Book(String title, String author, int totalQuantity) {
        super(title, author, totalQuantity);
    }

    public Book(int materialId, String title, String author, int totalQuantity,
int availableQuantity, LocalDate publicationDate) {
        super(materialId, title, author, "book", totalQuantity, availableQuantity,
publicationDate);
    }



    @Override
    public int getMaxLoanDays() {
        return 14; // Los libros se prestan por 14 días
    }
}
```

Journal.java

```
package com.library.model;

import java.time.LocalDate;

/**
 *
 * @author josef
 */
public class Journal extends Material {

    public Journal(int materialId, String title, String author, int totalQuantity,
            int availableQuantity, LocalDate publicationDate) {
        super(materialId, title, author, "journal", totalQuantity,
availableQuantity, publicationDate);
    }

    @Override
    public int getMaxLoanDays() {
        return 5; // Las revistas solo 5 días
    }
}
```

Loan.java

```
package com.library.model;

import java.time.LocalDate;

/**
 *
 * @author josef
 */
public class Loan {

    private int loanId;
    private int userId;
    private int materialId;
    private LocalDate loanDate;
    private LocalDate dueDate;
    private LocalDate returnDate;
    private double fine;

    public Loan() {
    }

    public Loan(int loanId, int userId, int materialId, LocalDate loanDate,
LocalDate dueDate, LocalDate returnDate, double fine) {
        this.loanId = loanId;
        this.userId = userId;
        this.materialId = materialId;
        this.loanDate = loanDate;
        this.dueDate = dueDate;
        this.returnDate = returnDate;
        this.fine = fine;
    }

    // Constructor para nuevos préstamos (sin ID todavía)
    public Loan(int userId, int materialId, LocalDate dueDate) {
        this.userId = userId;
        this.materialId = materialId;
        this.loanDate = LocalDate.now();
        this.dueDate = dueDate;
        this.fine = 0.0;
    }

    // Getters y Setters
    public int getLoanId() {
        return loanId;
    }
    public int getUserId() {
        return userId;
    }

    public int getMaterialId() {
        return materialId;
    }

    public LocalDate getLoanDate() {
        return loanDate;
    }

    public LocalDate getDueDate() {
        return dueDate;
    }

    public LocalDate getReturnDate() {
        return returnDate;
    }

    public double getFine() {
        return fine;
    }

    public void setReturnDate(LocalDate returnDate) {
        this.returnDate = returnDate;
    }

    public void setFine(double fine) {
        this.fine = fine;
    }

    @Override
    public String toString() {
        return "Loan #" + loanId
                + " | User: " + userId
                + " | Material: " + materialId
                + " | Due: " + dueDate
                + (returnDate != null ? " | Returned: " + returnDate : "");
    }
}
```

Material.java

```
package com.library.model;

import java.time.LocalDate;

/**
 *
 * @author josef
```

```java
 */
public abstract class Material {

    protected int materialId;
    protected String title;
    protected String author;
    protected String materialType;
    protected int totalQuantity;
    protected int availableQuantity;
    protected LocalDate publicationDate;

    public Material() {
    }

    public Material(String title, String author, int totalQuantity) {
        this.title = title;
        this.author = author;
        this.totalQuantity = totalQuantity;
    }

    public Material(int materialId, String title, String author, String
materialType, int totalQuantity, int availableQuantity, LocalDate publicationDate)
{
        this.materialId = materialId;
        this.title = title;
        this.author = author;
        this.materialType = materialType;
        this.totalQuantity = totalQuantity;
        this.availableQuantity = availableQuantity;
        this.publicationDate = publicationDate;
    }

    public int getMaterialId() {
        return materialId;
    }

    public void setMaterialId(int materialId) {
        this.materialId = materialId;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getAuthor() {
        return author;
    }

    public void setAuthor(String author) {
        this.author = author;
    }

    public String getMaterialType() {
        return materialType;
    }

    public void setMaterialType(String materialType) {
        this.materialType = materialType;
    }

    public int getTotalQuantity() {
        return totalQuantity;
    }

    public void setTotalQuantity(int totalQuantity) {
        this.totalQuantity = totalQuantity;
    }

    public int getAvailableQuantity() {
        return availableQuantity;
    }

    public void setAvailableQuantity(int availableQuantity) {
        this.availableQuantity = availableQuantity;
    }

    public LocalDate getPublicationDate() {
        return publicationDate;
    }

    public void setPublicationDate(LocalDate publicationDate) {
        this.publicationDate = publicationDate;
    }

    // Método abstracto: cada tipo puede tener reglas distintas
    public abstract int getMaxLoanDays();

    @Override
    public String toString() {
        return "[" + materialType.toUpperCase() + "] " + title + " - " + author
                + " (" + availableQuantity + "/" + totalQuantity + "
disponibles)";
    }
}
```

Professor.java

```
package com.library.model;

/**
 *
 * @author josef
 */
public class Professor extends User {

    public Professor(String name, String email) {
        super(name, email, "professor");
    }

    public Professor(int userId, String name, String email) {
        super(userId, name, email, "professor");
    }

    @Override
    public int getMaxLoanDays() {
        return 14; // Máximo 14 días de préstamo
    }

    @Override
    public double calculateFine(double baseFine, int overdueDays) {
        return (baseFine * overdueDays) * 0.5; // 50% menos multa
    }
}
```

Return.java

```
package com.library.model;

import java.time.LocalDate;

/**
 *
 * @author josef
 */
public class Return {

    private int returnId;
    private int loanId;
    private LocalDate returnDate;
    private double fine;

    public Return(int loanId, LocalDate returnDate, double fine) {
        this.loanId = loanId;
        this.returnDate = returnDate;
        this.fine = fine;
    }

    public Return(int returnId, int loanId, LocalDate returnDate, double fine) {
        this.returnId = returnId;
        this.loanId = loanId;
        this.returnDate = returnDate;
        this.fine = fine;
    }

    // Constructor sin ID (para nueva devolución)
    public Return(int loanId, double fine) {
        this.loanId = loanId;
        this.returnDate = LocalDate.now();
        this.fine = fine;
    }

    // Getters
    public int getReturnId() {
        return returnId;
    }

    public int getLoanId() {
        return loanId;
    }

    public LocalDate getReturnDate() {
        return returnDate;
    }

    public double getFine() {
        return fine;
    }

    @Override
    public String toString() {
        return "Return #" + returnId
                + " | Loan: " + loanId
                + " | Date: " + returnDate
                + " | Fine: Q" + fine;
    }
}
```

Student.java

```
package com.library.model;

/**
 *
 * @author josef
 */
public class Student extends User {

    public Student() {
    }

    public Student(String name, String email) {
        super(name, email, "student");
    }
```

```java
    public Student(int userId, String name, String email) {
        super(userId, name, email, "student");
    }

    @Override
    public int getMaxLoanDays() {
        return 7; // Máximo 7 días de préstamo
    }

    @Override
    public double calculateFine(double baseFine, int overdueDays) {
        return baseFine * overdueDays; // Multa base por cada día de retraso
    }
}
```

Thesis.java

```java
package com.library.model;

import java.time.LocalDate;

/**
 *
 * @author josef
 */
public class Thesis extends Material {

    public Thesis(int materialId, String title, String author, int totalQuantity,
            int availableQuantity, LocalDate publicationDate) {
        super(materialId, title, author, "thesis", totalQuantity,
availableQuantity, publicationDate);
    }

    @Override
    public int getMaxLoanDays() {
        return 10; // Las tesis se prestan 10 días
    }
}
```

User.java

```java
package com.library.model;

/**
 *
 * @author josef
 */
```

```
public abstract class User {

    protected int userId;
    protected String name;
    protected String email;
    protected String userType;

    public User() {
    }

    public User(String name, String email) {
        this.name = name;
        this.email = email;
    }

    public User(String name, String email, String userType) {
        this.userId = userId;
        this.name = name;
        this.email = email;
        this.userType = userType;
    }

    public User(int userId, String name, String email, String userType) {
        this.userId = userId;
        this.name = name;
        this.email = email;
        this.userType = userType;
    }

    public int getUserId() {
        return userId;
    }

    public void setUserId(int userId) {
        this.userId = userId;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getUserType() {
        return userType;
    }

    public void setUserType(String userType) {
        this.userType = userType;
    }

    // Método abstracto: cada tipo de usuario tiene límites distintos
    public abstract int getMaxLoanDays();

    public abstract double calculateFine(double baseFine, int overdueDays);

    @Override
    public String toString() {
        return "[" + userType.toUpperCase() + "] " + name + " (" + email + ")";
    }
}
```

🧰 Requisitos Técnicos

- ☕ Java 17 o superior
- 🗄 SQL Server 2019 o superior
- 🔗 JDBC Driver para SQL Server
- 🧱 NetBeans / IntelliJ IDEA / Eclipse

🧪 Ejecución

1. Configura la conexión en `ConnectionDB.java`:

```
String url = "jdbc:sqlserver://localhost:1433;databaseName=LibraryDB";

String user = "usuario";

String password = "contraseña";
```

2. Compila y ejecuta `Main.java`.

3. Prueba operaciones como: registrar usuario, crear préstamo o devolver material.

👨‍💻 Autor

Desarrollado por: José Andrés Flores Barco

📧 jfloresb9@miumg.edu.gt

📅 Proyecto académico 2025