Laboratorio #1: Diseño e Implementación de Clases en Java

Sistema de Gestión de una Librería

Nombre del Estudiante:
Brayan Felipe Tovar Hernández

Asignatura:
Programación Avanzada / Laboratorio #1

Profesor:
Javier Díaz

Institución:
Universidad Internacional de la Rioja (UNIR)

Fecha:
22 de Febrero de 2026

Tabla de Contenidos

Resumen

El presente documento detalla el diseño, la arquitectura y la implementación de un Sistema de Gestión para una Librería desarrollado en el lenguaje de programación Java. El objetivo del laboratorio es aplicar los conceptos fundamentales de la Programación Orientada a Objetos (POO), incluyendo herencia, polimorfismo, encapsulamiento y abstracción.

1. Introducción

En el contexto del "Laboratorio #1: Diseño e implementación de clases", se planteó la necesidad de construir una aplicación de consola que permita la administración integral de una librería. Los requisitos del sistema exigen la gestión de un catálogo de libros (categorizados en Novelas y Libros de Texto), el registro de sus respectivos autores, la administración de clientes y el procesamiento de ventas que afecten dinámicamente el inventario.

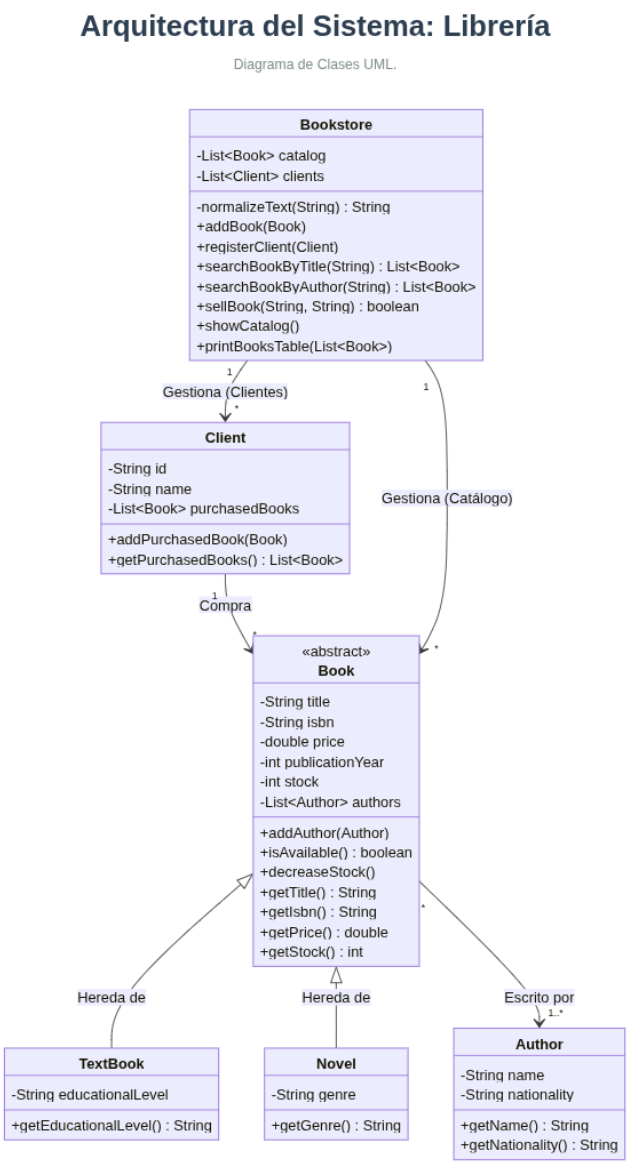
Para resolver esta problemática, se optó por una arquitectura basada en capas lógicas (Modelos y Servicios) y un enfoque interactivo mediante una Interfaz de Línea de Comandos (CLI).

2. Análisis y Diseño del Sistema

El diseño del sistema se basó en la identificación de las entidades principales del dominio del problema y sus interacciones.

2.1. Diagrama de Clases (UML)

A continuación, se presenta el diagrama de clases en notación UML, el cual define la estructura estática del sistema:



2.2. Explicación de la Arquitectura UML

El diseño respeta el Principio de Responsabilidad Única (SRP). Las relaciones principales son:

- **Herencia (Book <|-- Novel y Book <|-- TextBook):** Se utilizó una clase abstracta Book para centralizar los atributos comunes (ISBN, título, precio, stock). Las clases hijas extienden esta funcionalidad para agregar atributos específicos (género y nivel educativo), cumpliendo con el Principio Abierto/Cerrado (OCP).
- **Asociación Dirigida (Book --> Author):** Un libro conoce a sus autores a través de una lista de objetos Author.
- **Composición de Servicios (Bookstore):** Actúa como el controlador principal que encapsula las listas de clientes y el catálogo, orquestando las transacciones.

3. Memoria del Desarrollo de la Aplicación

Esta sección documenta el proceso cronológico y las decisiones técnicas tomadas durante la codificación del sistema, cumpliendo con los entregables del laboratorio.

3.1. Entorno de Desarrollo y Configuración

Aunque la guía sugiere el uso de Eclipse, el proyecto fue desarrollado utilizando un entorno moderno y ágil basado en **Visual Studio Code (VS Code)** con la extensión *Extension Pack for Java*. Se configuró el entorno sobre un sistema operativo Linux (Fedora), instalando el Java Development Kit a través del comando sudo dnf install java-17-openjdk-devel. Esta configuración permitió realizar un proceso de compilación nativo por línea de comandos, separando la dependencia de un IDE pesado y fomentando prácticas de desarrollo más pragmáticas.

Se estableció una estructura de carpetas basada en paquetes (src/libreria/models/ y src/libreria/services/) para garantizar una correcta organización de los archivos .java.

3.2. Proceso de Construcción y Pruebas

El desarrollo se llevó a cabo en las siguientes fases interactivas:

1. **Modelado de Entidades:** Se programaron en primer lugar las clases base (Author, Client, y la clase abstracta Book). Se añadieron constructores con validaciones para impedir la instanciación con datos corruptos.
2. **Aplicación de Herencia:** Se construyeron las clases Novel y TextBook aplicando la palabra reservada extends, reutilizando con éxito el código de la clase Book.
3. **Desarrollo del Servicio Central:** Se creó Bookstore.java para manejar las colecciones (Listas) del catálogo y los clientes. Aquí se resolvieron desafíos técnicos como la "búsqueda inteligente" utilizando flujos de datos (Streams) y normalización de cadenas de texto (para ignorar mayúsculas y tildes).
4. **Pruebas de Compilación y Ejecución:** En lugar de depender exclusivamente de botones gráficos, el proyecto se probó continuamente mediante consola usando los comandos del JDK:
    a. javac libreria/Main.java (Para compilar la aplicación asegurando que no hubiese errores de sintaxis).
    b. java libreria.Main (Para levantar la máquina virtual y probar la interfaz interactiva).
5. **Refinamiento Visual:** Se iteró sobre el diseño de la consola implementando un renderizado tabulado mediante System.out.printf(), mejorando sustancialmente la legibilidad del inventario para el usuario final.

4. Implementación y Descripción del Código

El código fuente fue organizado en dos paquetes principales para asegurar la modularidad: libreria.models y libreria.services.

4.1. Paquete de Modelos (libreria.models)

Contiene las entidades del dominio de negocio (POJOs):

- **Author.java y Client.java:** Implementan validaciones en sus constructores para evitar datos nulos o vacíos. Client posee un método getPurchasedBooks() que retorna una copia defensiva (new ArrayList<>) de la lista de compras, protegiendo su encapsulamiento.
- **Book.java:** Maneja de forma interna la lógica de disponibilidad. El método decreaseStock() asegura que no se puedan vender libros con inventario cero, arrojando una excepción IllegalStateException en caso de anomalías.

4.2. Paquete de Servicios (libreria.services)

- **Bookstore.java:** Es el núcleo transaccional. Se encarga de:
    - **Búsquedas Inteligentes:** Utiliza la API de Streams de Java 8+ y un método privado normalizeText() que elimina tildes y convierte textos a minúsculas, garantizando que el usuario encuentre los libros sin importar errores de formato ortográfico.
    - **Ventas (sellBook):** Valida la existencia del cliente y del libro (usando la clase Optional para evitar NullPointerException), verifica el stock, debita el inventario y añade el libro al historial del cliente en una sola operación atómica.
    - **Formatos visuales:** Cuenta con el método printBooksTable(), que utiliza System.out.printf() para renderizar el catálogo en un formato de tabla profesional por consola.

4.3. Clase Principal (Main.java)

Punto de entrada de la aplicación. Mantiene un ciclo de vida interactivo (while) apoyado en la clase Scanner y un switch para el menú de opciones. Incluye un método seedData() que inyecta datos de prueba (17 libros y 2 clientes) en memoria para facilitar las pruebas del laboratorio.

5. Buenas Prácticas y Patrones Aplicados

Durante el desarrollo, se priorizó la calidad del código adoptando dos filosofías principales de la ingeniería de software:

5.1. Principios de Clean Code (Código Limpio)

- **Nombres Reveladores de Intención:** Los métodos como decreaseStock() o findBookByIsbn() describen exactamente su función, eliminando la necesidad de comentarios excesivos.
- **DRY (Don't Repeat Yourself):** Toda la lógica de impresión de tablas se centralizó en printBooksTable(), siendo reutilizada tanto por el catálogo completo como por los resultados de búsqueda.
- **Uso de Streams:** Se reemplazaron las estructuras repetitivas for tradicionales por flujos de datos (filter(), map(), collect()), haciendo el código más declarativo y limpio.

5.2. Filosofía del Programador Pragmático

- **Validaciones *Fail-Fast*:** Los constructores validan inmediatamente parámetros inválidos (ej. precios negativos) y detienen la ejecución lanzando excepciones (IllegalArgumentException), evitando propagar errores silenciosos en el sistema.
- **Copias Defensivas para Inmutabilidad:** Los *getters* que devuelven listas (como los autores de un libro o los libros de un cliente) retornan instancias nuevas de ArrayList. Esto evita que módulos externos modifiquen el estado interno de los objetos.
- **Fina Coquetería (Experiencia de Usuario):** Se consideró la experiencia del usuario final, normalizando las cadenas de texto (eliminación de tildes) en las búsquedas y presentando la información en tablas tabuladas y alineadas en la terminal.

6. Conclusiones

1. La implementación de los conceptos de la Programación Orientada a Objetos permite modelar escenarios reales de forma intuitiva. La herencia y el polimorfismo reducen drásticamente la duplicación de código al tratar diferentes tipos de libros de forma uniforme.
2. La separación de responsabilidades (SRP) entre las clases de modelo, los servicios y la interfaz de usuario por consola facilita el mantenimiento y escalabilidad futura del sistema. Si en el futuro se desea cambiar la consola por una interfaz gráfica, el paquete libreria.models y libreria.services se mantendrán intactos.
3. La aplicación de prácticas como *Clean Code* y el enfoque pragmático no solo previene errores técnicos (como estados inválidos de inventario), sino que resulta en un código altamente legible y profesional.

7. Referencias

- Hunt, A. & Thomas, D. (1999). *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley.
- Martin, R. C. (2008). *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall.