Sistema de gestión de préstamos biblioteca ITM

Daniel Jimenez Correa

Daniel Jaramillo Rivera

Ingeniería de sistemas

Programación distribuida

Instituto Tecnológico Metropolitano de Medellín

2025

Problema:
La biblioteca estudiantil ITM gestiona sus libros y préstamos de forma manual (libros de registro en papel y hojas Excel). Esto causa errores frecuentes: libros mal localizados, control deficiente de copias prestadas y vencidas, retrasos en la devolución, poca trazabilidad de usuarios, y dificultad para conocer la disponibilidad en tiempo real. Además no hay un registro claro de reservas ni de sanciones por retrasos.

Usuarios:

Los usuarios principales son:

- **Socios**: personas registradas que desean consultar, prestar o devolver libros.
- **Encargados**: personal de la biblioteca que gestiona los préstamos y devoluciones.

El valor esperado del sistema es:

- Controlar préstamos y devoluciones de libros físicos y digitales.
- Evitar que se presten más ejemplares de los que existen (control de stock).
- Permitir a los encargados visualizar la disponibilidad de libros.
- Facilitar el registro histórico de préstamos y devoluciones.

Biblioteca: Libro (abstracto) → Físico/Digital, Socio, Prestamo, IPoliticaPrestamo.

Historias de usuario (6 con criterios Given/When/Then)

1. **Registro de socio**
    - Given que un nuevo usuario llega a la biblioteca,
    - When el encargado registra sus datos,
    - Then el sistema debe almacenar al socio como activo.
2. **Consulta de disponibilidad**
    - Given que un socio desea un libro,

- When consulta su disponibilidad,
- Then el sistema debe mostrar cuántos ejemplares físicos hay disponibles o si está en formato digital.

3. **Préstamo de libro físico**
    - Given que un socio solicita un libro físico,
    - When hay ejemplares disponibles,
    - Then el sistema debe registrar el préstamo y descontar 1 unidad del stock disponible.
4. **Préstamo de libro digital**
    - Given que un socio solicita un libro digital,
    - When realiza el préstamo,
    - Then el sistema debe registrar el préstamo sin afectar stock.
5. **Devolución de libro físico**
    - Given que un socio devuelve un libro,
    - When el encargado registra la devolución,
    - Then el sistema debe aumentar en 1 el stock disponible de ese libro.
6. **Control de vencimiento**
    - Given que un préstamo supera la fecha de devolución,
    - When el encargado consulta,
    - Then el sistema debe marcar el préstamo como vencido y calcular la multa correspondiente.

Requerimientos de negocio (5)

1. El sistema debe permitir registrar socios y libros (físicos o digitales).
2. El sistema debe controlar la disponibilidad de libros físicos mediante stock.
3. El sistema debe registrar y consultar préstamos activos e históricos.
4. El sistema debe calcular penalizaciones por retrasos en devoluciones.
5. El sistema debe ser operado por personal autorizado de la biblioteca.

4. Requerimientos funcionales (8)

1. Registrar y modificar datos de socios.
2. Registrar y modificar datos de libros (físicos y digitales).
3. Consultar la disponibilidad de libros.
4. Realizar préstamos de libros físicos y digitales.
5. Registrar devoluciones de libros físicos.
6. Calcular penalizaciones automáticas al registrar devoluciones fuera de plazo.
7. Consultar historial de préstamos por socio.
8. Generar un log básico de operaciones (registro de acciones de préstamo y devolución).

5. Requerimientos no funcionales (6)

1. **Rendimiento**: El sistema debe registrar y consultar préstamos en menos de 1 segundo con hasta 100 libros y socios.
2. **Usabilidad**: Los menús en consola deben ser claros y guiar al usuario paso a paso.
3. **Manejo de errores**: Validar entradas incorrectas (ej. cédula repetida, stock agotado, montos inválidos) y mostrar mensajes explicativos.
4. **Escalabilidad**: El diseño debe permitir agregar nuevos tipos de libros (ej. audiolibros) o reglas sin reescribir todo el código.

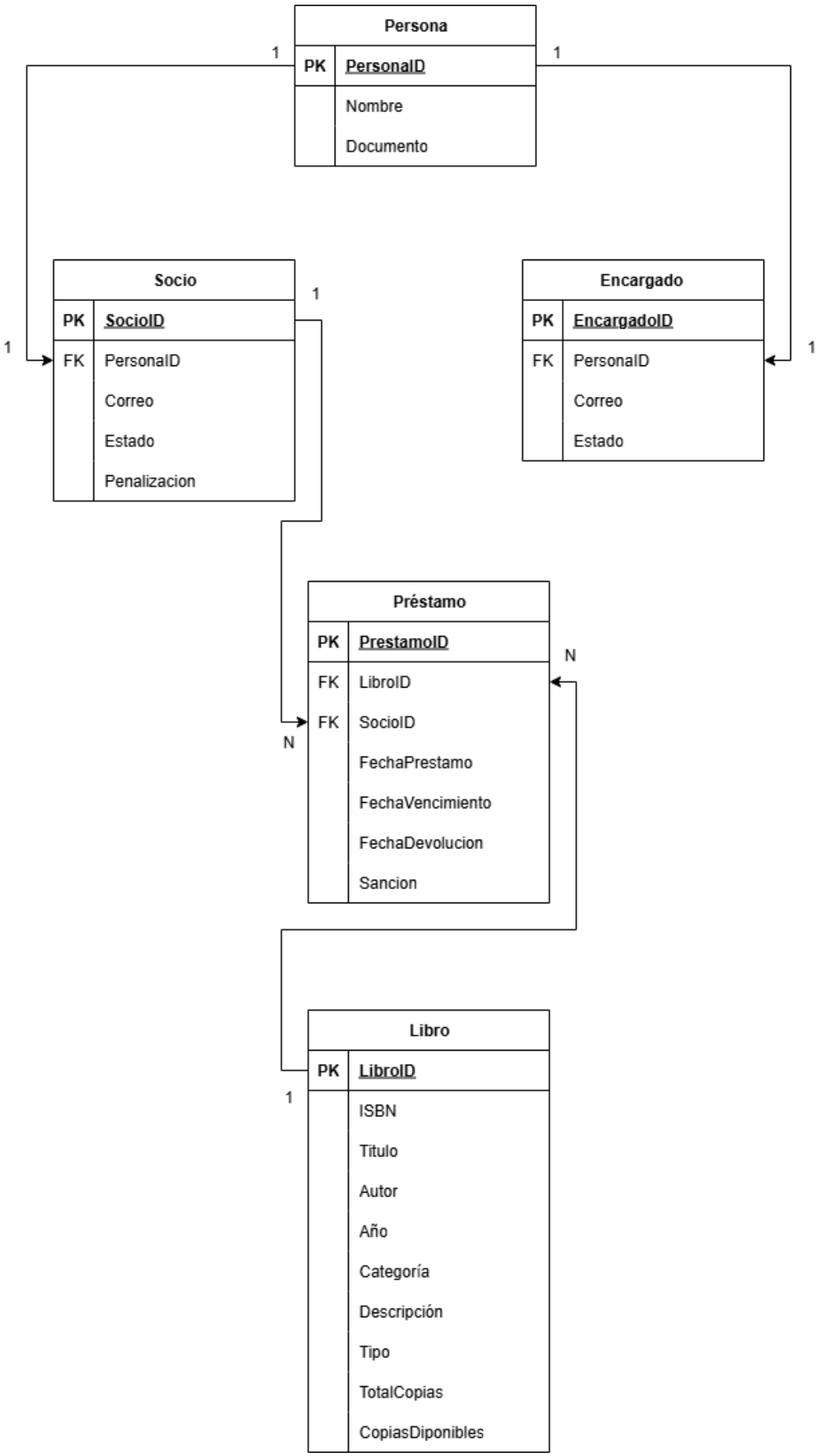
5. **Calidad del código**: Mantener principios de POO (encapsulación, clases separadas por responsabilidad).

6. **Portabilidad**: Ejecutarse en Windows o Linux con .NET 8 sin configuraciones extra.

6. Modelo conceptual (texto con relaciones)

Entidades principales:

- **Libro**
    - Atributos: LibroID (PK), ISBN, Título, Autor, Año, Categoría, Tipo (Digital/Físico), TotalCopias, CopiasDisponibles
    - Relación: 1 ibro — 0..N Préstamos
- **Personas (abstracta)**
    - Atributos: Nombre, Documento
    - **Socio (hereda de Persona)**
        - Atributos: SocioID(PK), Correo, Estado, Penalización
        - Relación: 1 Socio — 0..N Préstamos
    - **Encargado (hereda de Person)**
        - Atributos: EncargadoID (PK), Correo, Estado
        - Relación: encargado autoriza conceptual préstamos y devoluciones.
- **Préstamo**
    - Atributos: PrestamoID (PK), LibrosID (FK), SocioID (FK), FechaPrestamo, FechaVencimiento, FechaDevolucion, Sanción
    - Relación: vincula Libro y Socio.
    - Relación: 1 socio - 0..N Préstamos.
    - Relación: 1 Libro - 0..N Prestamos


Persona
PK
PersonalID
Nombre
Documento
1
1
Socio
PK
SocioID
FK
PersonalID
Correo
Estado
Penalizacion
1
1
Encargado
PK
EncargadoID
FK
PersonalID
Correo
Estado
1
Préstamo
PK
PrestamoID
FK
LibroID
FK
SocioID
FechaPrestamo
FechaVencimiento
FechaDevolucion
Sancion
N
N
Libro
PK
LibroID
ISBN
Titulo
Autor
Año
Categoría
Descripción
Tipo
TotalCopias
CopiasDiponibles
1

Encapsulamiento

Cada clase expone solo lo necesario:

- **Atributos privados** (ej.: disponibilidad de un libro, penalidades de un socio).

- **Métodos públicos controlados** (ej.: `Prestar()`, `Devolver()`).
  Esto protege los datos de modificaciones indebidas y asegura consistencia en las reglas del negocio.

Ejemplo conceptual:
Un libro no puede marcarse como *prestado* directamente alterando el stock; debe hacerse mediante un método que valide si hay copias disponibles.

Herencia aplicada

Se usa herencia donde hay una **relación natural de generalización/especialización**:

- Clase **Persona** (atributos comunes: nombre, documento, email, teléfono).

- Subclases **Socio (Member)** y **Encargado (Staff)** que extienden Persona, añadiendo sus responsabilidades particulares.

Esto evita duplicar atributos y permite manejar a las personas bajo una interfaz común.

Polimorfismo

El sistema permite que diferentes tipos de libros (digitales y físicos) se comporten de manera distinta, aunque compartan la misma interfaz:

- **Libro físico**: requiere control de stock y devoluciones.

- **Libro digital**: acceso inmediato, sin límite de stock.

El polimorfismo permite tratarlos como **`Libro`**, pero aplicar reglas de negocio distintas según su tipo.

Abstracción (clases/servicios/contratos)

El diseño abstraer conceptos clave para no depender de detalles de implementación:

La clase Libro encapsula propiedades comunes (título, autor, año, categoría) y mediante la variable TipoLibro diferencia si es físico o digital, sin necesidad de crear múltiples subclases. Esto evita duplicación de código y concentra la lógica en un único lugar, a la vez que permite condiciones específicas (ejemplo: validar stock solo en libros físicos). Servicios como **ProcesadorPrestamos** gestionan la lógica de prestar y devolver, sin que la clase Libro tenga que ocuparse de todas las reglas de negocio. Esta separación facilita extender el sistema a futuro (ejemplo: audiolibros como un nuevo valor en TipoLibro o nuevas reglas de préstamo) sin romper la estructura existente.

Estructura limpia (capas)

El diseño sigue principios de organización:

- **Capa de entidades**: define objetos del dominio (Libro, Socio, Encargado, Préstamo).
- **Capa de servicios**: contiene reglas de negocio (gestión de préstamos, reservas, penalidades).
- **Capa de presentación**: interfaz de consola simple para interactuar con el sistema.

Los nombres de clases, atributos y métodos son claros y consistentes, lo que hace el sistema fácil de entender y mantener.

Paso a paso (guía de trabajo)

5.1. Definir el caso de negocio

Problema (¿Qué duele hoy?)
La biblioteca no cuenta con un sistema sencillo para **controlar préstamos y devoluciones de libros**, lo que genera problemas como:

- Falta de trazabilidad de qué socio tiene un libro.
- Dificultad para aplicar penalizaciones por retrasos.
- Riesgo de prestar libros a socios suspendidos.

Usuarios

- **Encargado (Administrador/Bibliotecario):** opera el sistema, registra préstamos, devoluciones y cobros de multas.
- **Socio:** usuario final que solicita libros y paga multas.

Valor (¿qué mejora concreta entregará?)

- Llevar un control estructurado de préstamos y devoluciones.
- Aplicar automáticamente multas por retraso.
- Permitir suspender o reactivar socios según su estado.
- Mejorar la transparencia y confianza en el servicio de la biblioteca.

5.2. Historias de usuario

HU1. Registrar socios

Como Encargado
quiero registrar socios en el sistema
para permitirles acceder a los préstamos de libros.

Criterios de aceptación:

- **Given** un socio con cédula única
- **When** lo registro con nombre y correo válido
- **Then** queda disponible para realizar préstamos.

HU2. Registrar libros

Como Encargado
quiero registrar libros en el inventario
para que estén disponibles para préstamos.

Criterios de aceptación:

- **Given** un libro con ISBN único
- **When** lo registró con título, autor, año y ejemplares
- **Then** queda en inventario disponible para préstamos.

HU3. Registrar préstamo

Como Encargado
quiero registrar un préstamo de libro a un socio
para llevar control de la fecha de vencimiento.

Criterios de aceptación:

- **Given** que el socio está activo y hay ejemplares disponibles
- **When** registro el préstamo
- **Then** se descuenta un ejemplar y se guarda con fecha de vencimiento.

HU4. Registrar devolución

Como Encargado
quiero registrar la devolución de un libro
para actualizar el inventario y aplicar multa si hay retraso.

Criterios de aceptación:

- **Given** que el préstamo está activo
- **When** registro la devolución
- **Then** se libera un ejemplar y si hay retraso se genera penalización.

HU5. Pagar multa

Como Encargado
quiero registrar el pago de multas de un socio
para que quede habilitado nuevamente para prestar libros.

Criterios de aceptación:

- **Given** que el socio tiene multas pendientes
- **When** se paga la totalidad de la deuda
- **Then** el sistema actualiza la penalización a cero.

5.3. Requerimientos

De negocio (RN)

- **RN-01:** Solo los socios activos pueden realizar préstamos.
- **RN-02:** Un préstamo tiene una fecha de vencimiento definida al momento de su creación.
- **RN-03:** Si un socio devuelve tarde un libro, se genera multa fija por día de retraso.
- **RN-04:** Un socio suspendido no puede registrar préstamos.
- **RN-05:** Las multas deben pagarse en su totalidad.

Funcionales (RF)

- **RF-01:** CRUD de socios.
- **RF-02:** Registrar préstamos.
- **RF-03:** Registrar devoluciones.
- **RF-04:** Calcular y aplicar multas automáticamente.
- **RF-05:** Registrar pagos de multas.
- **RF-06:** Mostrar estado del socio y sus deudas.
- **RF-07:** Menú de consola guiado para navegar entre opciones.

No funcionales (RNF)

- **RNF-01:** Todas las operaciones comunes deben responder en < 1 segundo con hasta 1000 registros.
- **RNF-02:** El sistema debe mostrar mensajes claros en caso de error.
- **RNF-03:** La arquitectura debe separar **Common, Domain y Servicios** para mantener el código limpio.
- **RNF-04:** Compatible con .NET 8 y ejecutable en Windows/Linux.
- **RNF-05:** Código orientado a objetos siguiendo principios SOLID básicos.
- **RNF-06:** Fácil de probar (métodos aislados con `Result`).

5.4. Diseño POO (aplicar los 4 pilares)

- **Abstracción:**

Definimos modelos y servicios esenciales para representar el dominio sin entrar en detalles de implementación:

Socio, Libro, Prestamo, Encargado → abstraen los conceptos del sistema.

ProcesadorPrestamos → abstrae la lógica de registrar, devolver y pagar multas.

- **Encapsulamiento:** Propiedades privadas + validaciones.

Protegemos los datos con propiedades privadas y métodos de acceso controlados:

- En Socio, la penalización solo se puede modificar mediante AgregarPenalizacion y PagarPenalizacion.
- En Libro, CopiasDisponibles se modifica sólo a través de los métodos Prestar y Devolver, esto asegura que nadie pueda cambiar el estado interno de manera incorrecta (ej. poner copias en negativo).

Herencia: reutilizamos código y evitamos duplicación mediante jerarquías de clases:

- Persona es la clase base → Socio y Encargado heredan de ella.

Esto permite que todos compartan atributos como Documento y Nombre.

Si en un futuro se crean más roles (ej. Autor, Proveedor), se puede heredar de Persona sin reescribir todo.

Polimorfismo: Interfaces/override/estrategia.

Aplicamos comportamientos diferentes según el contexto:

- El método ToString() se sobreescribe en varias clases (Socio, Libro, Prestamo) para mostrar información adaptada a cada entidad.
- La penalización se calcula dinámicamente según la fecha de devolución (Penalizacion.CalcularMulta).
- Con el campo TipoLibro (Físico/Digital), podemos extender la lógica: un libro físico valida stock, uno digital no necesita esa restricción.

Github :https://github.com/Jcdanieljimenez/BibliotecaITM.git