Sistema de Gestión de Biblioteca

Trabajo Práctico – Unidad 3: Diseño Arquitectónico

Materia: Ingeniería de Software II

Profesor: Víctor Hugo Contreras

Estudiante: Alejo Ezequiel Escurra

Fecha: Octubre 2025

1. Arquitectura en Tres Capas del Sistema

El sistema de gestión de biblioteca implementa una arquitectura en tres capas que separa las responsabilidades y facilita el mantenimiento, escalabilidad y testing del software.

1.1. Capa de Presentación (UI - User Interface)

Es la interfaz con la que interactúa el usuario, en este caso el bibliotecario o socio de la biblioteca. Esta capa es responsable de toda la interacción visual y la experiencia del usuario.

Funciones principales:

- Mostrar menús, formularios y resultados de forma clara y organizada.
- Recibir las acciones del usuario (buscar libro, registrar préstamo, consultar disponibilidad, etc.).
- Validar datos básicos antes de enviarlos a la capa de negocio (formato de entrada, campos obligatorios).
- Proporcionar retroalimentación inmediata al usuario sobre las operaciones realizadas.

> **Ejemplo práctico:** Pantalla de "Registro de Préstamo" donde el bibliotecario selecciona el socio y el libro mediante una interfaz intuitiva. La validación de que los campos no estén vacíos se realiza en esta capa antes de procesar la solicitud.

Componente principal: Clase `Main.java` que implementa todos los menús y la lógica de interacción con el usuario.

1.2. Capa de Lógica de Negocio (Business Layer)

Contiene las reglas del negocio y la lógica que hace funcionar el sistema. Esta capa es el cerebro del sistema de la biblioteca, se encarga de validar y procesar las peticiones de la Capa de Presentación, implementa las reglas del sistema y coordina las operaciones entre las capas de Presentación y la capa de Datos.

Funciones principales:

- Validar si un socio puede realizar un préstamo (verificar que no tenga deudas, que no esté suspendido, que no exceda el límite de préstamos simultáneos).
- Calcular multas por devolución tardía aplicando las tarifas establecidas por día de retraso.
- Gestionar las operaciones de préstamo, devolución y registro de socios aplicando todas las reglas de negocio necesarias.
- Verificar disponibilidad de libros antes de permitir un préstamo.
- Controlar los estados de los libros (disponible, prestado, en mantenimiento).

> **Ejemplo práctico:** Clase `GestorPrestamos` con métodos como `realizarPrestamo()` y `registrarDevolucion()`. Antes de realizar un préstamo, el gestor verifica que el socio esté activo, que no tenga préstamos vencidos, y que el libro esté disponible.

Componentes principales:

- `GestorLibros.java` - Gestión de operaciones relacionadas con libros
- `GestorSocios.java` - Gestión de operaciones relacionadas con socios
- `GestorPrestamos.java` - Gestión de préstamos, devoluciones y cálculo de multas

1.3. Capa de Datos (Data Layer)

Esta capa es la responsable de la persistencia de la información del sistema. Se encarga de conectar con la base de datos, de realizar operaciones CRUD (Create, Read, Update, Delete) de las entidades del sistema y de abstraer la lógica de acceso a la base de datos de la Capa de Lógica de Negocios.

Funciones principales:

- Consultar, insertar, modificar o eliminar registros de libros, socios y préstamos en la base de datos.
- Asegurar la integridad y persistencia de la información mediante transacciones controladas.
- Proporcionar una interfaz clara y consistente para el acceso a datos (patrón DAO).
- Manejar la conexión a la base de datos de forma centralizada y eficiente.
- Implementar consultas optimizadas para mejorar el rendimiento del sistema.

> **Ejemplo práctico:** Clase `LibroDAO` con métodos como `obtenerLibroPorId()` o `actualizarEstadoLibro()`. Estas clases abstraen completamente las operaciones de base de datos, permitiendo que la capa de negocio trabaje con objetos Java sin preocuparse por el SQL subyacente.

Componentes principales:

- `ConexionDB.java` - Gestión centralizada de conexión (Patrón Singleton)
- `LibroDAO.java` - Operaciones CRUD para libros
- `SocioDAO.java` - Operaciones CRUD para socios
- `PrestamoDAO.java` - Operaciones CRUD para préstamos

2. Problema Identificado y Solución con Patrón de Diseño

2.1. Problema: Acceso Centralizado a la Base de Datos

El problema que se identificó para este trabajo es el de **"Acceso centralizado a la Base de Datos"**, ya que es un aspecto fundamental el hecho de que la conexión a la base de datos y el objeto que gestiona las operaciones de bajo nivel sean únicos y que se acceda a él de forma controlada desde la capa de Datos.

Problemas que se evitan con esta solución:

- **Conexiones múltiples y simultáneas:** Sin control centralizado, cada operación podría crear su propia conexión, agotando los recursos del servidor de base de datos.
- **Inconsistencias en las configuraciones:** Diferentes partes del sistema podrían usar diferentes parámetros de conexión.
- **Sobrecarga de recursos:** La creación y destrucción constante de conexiones consume memoria y tiempo de procesamiento.
- **Dificultad en el mantenimiento:** Cambiar la configuración de la base de datos requeriría modificar múltiples lugares del código.
- **Problemas de concurrencia:** Múltiples conexiones podrían causar bloqueos o deadlocks en la base de datos.

2.2. Solución: Patrón Singleton

El patrón que se eligió a raíz de este problema es el **Patrón Singleton**, ya que garantiza que una clase tenga una única instancia y proporciona un punto de acceso global a ella.

Implementación del Patrón:

La forma en la que se implementa es con la creación de una clase de gestor de conexión que implemente el patrón Singleton. Esta clase manejaría:

- La configuración de conexión (credenciales, URL, puerto, etc.)
- La conexión física a la base de datos
- El control de acceso a esta única instancia

Todas las clases dentro de la capa de Datos que necesiten interactuar con la persistencia obtienen la misma y única instancia de la clase de gestor de conexión.

Pseudocódigo de Implementación:

```
public class ConexionDB { // Instancia única (static) private static ConexionDB instancia; private Connection conexion; // Constructor privado - evita instanciación externa private ConexionDB() { conexion = DriverManager.getConnection( "jdbc:mysql://localhost:3306/biblioteca" ); } // Método estático para obtener la única instancia public static ConexionDB getInstancia() { if (instancia == null) { instancia = new ConexionDB(); } return instancia; } // Método para obtener la conexión public Connection getConexion() { return conexion; } }
```

Uso en las Clases DAO:

```
// Así, cualquier clase que necesite acceder a la base de datos llama: Connection con = ConexionDB.getInstancia().getConexion(); // Ejemplo en LibroDAO: public class LibroDAO { private ConexionDB conexion; public
```

```
LibroDAO() { this.conexion = ConexionDB.getInstancia(); } public Libro
obtenerPorId(int id) { Connection con = conexion.getConexion(); //
Realizar operaciones de base de datos... } }
```

Ventajas de esta Implementación:

- ✓ Una única conexión a la base de datos durante toda la ejecución del programa
- ✓ Acceso global y controlado desde cualquier punto del sistema
- ✓ Optimización significativa de recursos del sistema
- ✓ Facilita el mantenimiento - cambios de configuración en un solo lugar
- ✓ Previene problemas de concurrencia relacionados con múltiples conexiones
- ✓ Mejora el rendimiento al evitar la sobrecarga de crear/destruir conexiones
- ✓ Consistencia en toda la aplicación al usar la misma configuración

Características Clave del Patrón Singleton:

1. **Constructor Privado:** Impide que otras clases creen instancias directamente usando `new ConexionDB()`
2. **Instancia Estática:** La variable `instancia` es static, lo que significa que pertenece a la clase y no a objetos individuales
3. **Método de Acceso Estático:** `getInstancia()` es el único punto de acceso para obtener la instancia
4. **Inicialización Perezosa (Lazy Initialization):** La instancia se crea solo cuando se necesita por primera vez

3. Conclusión

La implementación de una arquitectura en tres capas junto con el patrón Singleton para la

gestión de conexiones proporciona una solución robusta, escalable y mantenible para el sistema de gestión de biblioteca. Esta arquitectura permite:

- Separación clara de responsabilidades entre presentación, lógica de negocio y acceso a datos
- Facilidad para realizar cambios y mejoras sin afectar otras partes del sistema
- Optimización de recursos mediante el uso de una única conexión a la base de datos
- Mejor organización del código siguiendo principios de diseño sólidos
- Mayor facilidad para realizar pruebas unitarias en cada capa de forma independiente

El patrón Singleton aplicado a la conexión de base de datos demuestra cómo un patrón de diseño bien elegido puede resolver problemas fundamentales de arquitectura de software, mejorando tanto el rendimiento como la mantenibilidad del sistema.
