2025

INFORME DE REFACTORIZACIÓN

Miguel Angel Patraca Lagunes

29/4/25

Indice

Introducción

En esta actividad llevaré a cabo la refactorización de un programa de gestión de biblioteca el cual tiene las funciones de registrar libros y usuarios, buscar libros y usuarios, prestar libros, devolver libros y mostrar los préstamos.

Este informe mostrará los problemas en el código que identifique , los cambios que realice y las técnicas de refactorización que seguí, además de justificar el por que tome esas decisiones en el diseño del programa.

Problemas identificados

Clase Usuario

En la clase usuario note problemas al momento de implementar herencia ya que solo se instanciaba a un usuario sin saber si es un estudiante o un profesor el que se registró al sistema

```java
public abstract class Usuario {
    private int id;
    private String nombre;
    private String email;
    private String telefono;

    public Usuario(int id, String nombre, String email, String telefono) {
        this.id = id;
        this.nombre = nombre;
        this.email = email;
        this.telefono = telefono;
    }

    // Getters y setters
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getEmail() {
        return email;
    }

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

    public String getTelefono() {
        return telefono;
    }

    public void setTelefono(String telefono) {
        this.telefono = telefono;
    }
}
```

También noté que tampoco hay una interfaz que demuestre el diferente comportamiento de los 2 tipos de usuario al momento de registrar, realizar un préstamo o devolver un libro

Clase Libro

En la clase libro fue el mismo problema que con la clase usuario ya que no se implementa la herencia para los diferentes tipos de libros (Audiovisual , Digital y Físico) , igualmente tampoco hay el polimorfismo para cuando se presta, se devuelve y se registra un nuevo libro

```java
// Libro.java    Libro.java is a non-project file, only syntax errors are reported
public class Libro {
    private int id;
    private String titulo;
    private String autor;
    private int anio;
    private String genero;
    private boolean disponible;

    public Libro(int id, String titulo, String autor, int anio, String genero, boolean disponible) {
        this.id = id;
        this.titulo = titulo;
        this.autor = autor;
        this.anio = anio;
        this.genero = genero;
        this.disponible = disponible;
    }

    // Getters y setters
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getTitulo() {
        return titulo;
    }

    public void setTitulo(String titulo) {
        this.titulo = titulo;
    }

    public String getAutor() {
        return autor;
    }

    public void setAutor(String autor) {
        this.autor = autor;
    }

    public int getAnio() {
        return anio;
    }

    public void setAnio(int anio) {
        this.anio = anio;
    }

    public String getGenero() {
        return genero;
    }

    public void setGenero(String genero) {
        this.genero = genero;
    }

    public boolean isDisponible() {
        return disponible;
    }

    public void setDisponible(boolean disponible) {
        this.disponible = disponible;
    }
}
```

De igual forma noté que algunos métodos de la clase main deberían estar en esta clase, igualmente para la clase usuario.

Clase préstamo

En la clase préstamo en mi opinión no tiene problemas de código en ese sector, pero si pienso que algunos métodos de la clase main deberían estar en esta clase

```java
// Prestamo.java
import java.util.Date;

public class Prestamo   {
    private int id;
    private int idLibro;
    private int idUsuario;
    private Date fechaPrestamo;
    private Date fechaDevolucion;
    private boolean devuelto;

    public Prestamo(int id, int idLibro, int idUsuario, Date fechaPrestamo, Date fechaDevolucion, boolean devuelto) {
        this.id = id;
        this.idLibro = idLibro;
        this.idUsuario = idUsuario;
        this.fechaPrestamo = fechaPrestamo;
        this.fechaDevolucion = fechaDevolucion;
        this.devuelto = devuelto;
    }

    // Getters y setters
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public int getIdLibro() {
        return idLibro;
    }

    public void setIdLibro(int idLibro) {
        this.idLibro = idLibro;
    }

    public int getIdUsuario() {
        return idUsuario;
    }

    public void setIdUsuario(int idUsuario) {
        this.idUsuario = idUsuario;
    }

    public Date getFechaPrestamo() {
        return fechaPrestamo;
    }
```

Clase BibliotecaApp

En la clase Main en mi opinión hay muchos problemas

- inicializarDatos

  Para mi este método esta de más por que viola el principio SOLID , los usuarios y libros de ejemplo ocupan líneas de código que se pueden solucionar creándolos dentro del programa no antes , o también haciendo una clase demo para probar estos datos

```
    // esta de más
    private static void inicializarDatos() {
        // Libros de ejemplo
        libros.add(new Audiolibro(id:1, titulo:"Don Quijote de la Mancha", autor:"Miguel de Cervantes", anio:1605, genero:"Ficción", dis
        libros.add(new Digital(id:2, titulo:"Cien años de soledad", autor:"Gabriel García Márquez", anio:1967, genero:"Novela", disponib
        libros.add(new Fisico(id:3, titulo:"El principito", autor:"Antoine de Saint-Exupéry", anio:1943, genero:"Fábula", disponible:tru

        // Usuarios de ejemplo
        usuarios.add(new Estudiante(id:101, nombre:"Jose Camacho", email:"jantonio@gmail.com", telefono:"123456789"));
        usuarios.add(new Profesor(id:102, nombre:"Patricia Moreno", email:"patricia@gmail.com", telefono:"987654321"));
    }
```

- Registrar libro

  En este método yo pienso que si está bien colocado, pero solo puede registrar un tipo de libro, dejando de lado los otros tipos como digital y audiolibro

```
    private static void registrarLibro() {
        System.out.println(x:"--- REGISTRAR NUEVO LIBRO ---");
        System.out.println(x:"");

        System.out.print(s:"ID: ");
        int id = scanner.nextInt();
        scanner.nextLine();  // Consumir el salto de línea

        System.out.print(s:"Título: ");
        String titulo = scanner.nextLine();

        System.out.print(s:"Autor: ");
        String autor = scanner.nextLine();

        System.out.print(s:"Año: ");
        int anio = scanner.nextInt();
        scanner.nextLine();  // Consumir el salto de línea

        System.out.print(s:"Género: ");
        String genero = scanner.nextLine();

        Libro nuevoLibro = new Fisico(id, titulo, autor, anio, genero, disponible:true);
        libros.add(nuevoLibro);

        System.out.println(x:"Libro registrado con éxito.");
    }
```

- registrarUsuario
  Mismo caso que el método registrarLibro , este método solo puede instanciar la clase usuario, pero no diferencia los 2 tipos (profesor y estudiante)

```java
private static void registrarUsuario() {
    System.out.println(x:"--- REGISTRAR NUEVO USUARIO ---");

    System.out.print(s:"ID: ");
    int id = scanner.nextInt();
    scanner.nextLine();  // Consumir el salto de línea

    System.out.print(s:"Nombre: ");
    String nombre = scanner.nextLine();

    System.out.print(s:"Email: ");
    String email = scanner.nextLine();

    System.out.print(s:"Teléfono: ");
    String telefono = scanner.nextLine();

    Usuario nuevoUsuario = new Estudiante(id, nombre, email, telefono);
    usuarios.add(nuevoUsuario);

    System.out.println(x:"Usuario registrado con éxito.");
}
```

- prestarLibro
  Este método en mi opinión debe estar en la clase Prestamo ya que es una responsabilidad de esa clase prestar y devolver un libro , además de que este método tiene más de 5 responsabilidades ya que
  - Busca el id de libro y usuario
  -Verifica si el libro está disponible
  -Verifica si el usuario tiene más de 3 libros prestados
  -Realiza el préstamo
  -Marca como no disponible el libro que se prestó

  Por lo que este método viola el principio de responsabilidad única

```java
// Refactorizar 5 responsabilidades
// Bucar libro y usuario por ID
// Verificar si el libro está disponible
// Verificar si el usuario tiene más de 3 libros prestados
// Realizar el préstamo
// Marcar libro como no disponible
private static void prestarLibro() {
    System.out.println(x:"--- PRESTAR LIBRO ---");

    System.out.print(s:"ID del libro: ");
    int idLibro = scanner.nextInt();
    scanner.nextLine();  // Consumir el salto de línea

    System.out.print(s:"ID del usuario: ");
    int idUsuario = scanner.nextInt();
    scanner.nextLine();  // Consumir el salto de línea

    Libro libro = null;
    Usuario usuario = null;

    // Buscar libro
    for (Libro l : libros) {
        if (l.getId() == idLibro) {
            libro = l;
            break;
        }
    }

    // Buscar usuario
    for (Usuario u : usuarios) {
        if (u.getId() == idUsuario) {
            usuario = u;
            break;
        }
    }

    if (libro == null) {
```

- devolverLibro
  Este método tiene más de 4 responsabilidades, aparte de que pienso que debería estar en la clase Prestamo , las responsabilidades de este método son:
  - Busca préstamo activo
  - Marca préstamo como devuelto
  - Marca libro como disponible
  - Devuelve el libro a la biblioteca

```java
// Refactorizar 4 responsabilidades
// Buscar préstamo activo para este libro
// Marcar préstamo como devuelto
// Marcar libro como disponible
// Devolver libro
private static void devolverLibro() {
    System.out.println(x:"--- DEVOLVER LIBRO ---");

    System.out.print(s:"ID del libro: ");
    int idLibro = scanner.nextInt();
    scanner.nextLine();  // Consumir el salto de línea

    Prestamo prestamo = null;

    // Buscar préstamo activo para este libro
    for (Prestamo p : prestamos) {
        if (p.getIdLibro() == idLibro && !p.getDevuelto()) {
            prestamo = p;
            break;
        }
    }

    if (prestamo == null) {
        System.out.println(x:"Error: No hay préstamos activos para este libro.");
        return;
    }

    // Marcar préstamo como devuelto
    prestamo.setDevuelto(devuelto:true);
    prestamo.setFechaDevolucion(new Date());

    // Marcar libro como disponible
    for (Libro l : libros) {
        if (l.getId() == idLibro) {
            l.setDisponible(disponible:true);
            break;
```

- buscarLibro y mostrarLibros
  En estos 2 métodos el problema que encontré es que 2 bucles for hacen exactamente la misma función por lo que se repite código y agrandan las líneas de código , además de que ambos duplican código de mostrar información de libros

```java
            if (coincide) {
                System.out.println("ID: " + libro.getId() + " | Título: " + libro.getTitulo() +
                                   " | Autor: " + libro.getAutor() + " | Año: " + libro.getAnio() +
                                   " | Género: " + libro.getGenero() +
                                   " | Disponible: " + (libro.getDisponible() ?"Sí" : "No"));
                encontrado = true;
            }
        }

        if (!encontrado) {
            System.out.println(x:"No se encontraron libros que coincidan con la búsqueda.");
        }
    }

    private static void mostrarLibros() {
        System.out.println(x:"--- LISTADO DE LIBROS ---");

        if (libros.isEmpty()) {
            System.out.println(x:"No hay libros registrados.");
            return;
        }

        for (Libro libro : libros) {
            System.out.println("ID: " + libro.getId() + " | Título: " + libro.getTitulo() +
                               " | Autor: " + libro.getAutor() + " | Año: " + libro.getAnio() +
                               " | Género: " + libro.getGenero() +
                               " | Disponible: " + (libro.getDisponible() ?"Sí" : "No"));
        }
    }
```

- mostrarUsuarios
  en este caso pienso que el único problema es que no divide los 2 tipos de roles en los usuarios
- mostrarPrestamosActivos
  En este método pienso que no hay problemas

Técnicas de factorización empleadas

Clase Usuario

- Clase de extracción
  Convertí la clase Usuario a una clase abstracta y cree 2 nuevas clases de Estudiante y Profesor para implementar la herencia y dividir los métodos únicos de cada clase

Clase Libro

- Clase de extracción
  Convertí la clase Usuario a una clase abstracta y cree 3 nuevas clases de libros para implementar la herencia y dividir los métodos únicos de cada clase

Clase Préstamo

Clase BibliotecaApp

- Método de extracción
  - Agrupé el menú por consola al momento de ejecutar el programa a un método independiente y remplacé el código por un llamada al método menu()
  - Agrupé la información de los libros y usuarios que se repetían en los métodos:
    - buscarLibros()
    - mostrarLibros()
    - mostrarUsuarios()
    - mostrarPrestamosActivos()

    Agrupé también la información de los prestamos activos a una llamada para futuras funciones

    Cree los métodos:

    - registrarLibroFisico();
    - registrarLibroDigital();
    - registrarLibroAudiolibro();
    - registrarEstudiante();
    - registrarProfesor();
    - verificarPrestamos();
    - marcarDevolucion();
    - realizarPrestamo();
    - buscarPrestamoActivo();
    - mostrarInfromacionLibro();
    - mostrarInfromacionUsuario();
    - mostrarInfromacionPrestamo();

- Método de línea
  Remplacé el cuerpo de los métodos:
  - prestarLibro()
  - devolverLibro

  porque tenían muchas responsabilidades lo que violaba el principio SOLID, creando los métodos:

  - verificarPrestamos();
  - marcarPrestamo();
  - realizarPrestamo();

- Mover método
  Movi los métodos
  - marcarDevolucion();
  - realizarPrestamo();

  A la clase préstamo por que pienso son funciones de esa clase

- Clase de extracción
  Extraje las funciones de la clase BibliotecaApp para dejar solo las opciones de la biblioteca y creé una nueva clase GestionarBiblioteca para tener solo 1 responsabilidad en ambas clases

Justificaciones de diseño

Clase usuario

Decidí hacer la clase de usuario abstracta para poder crear las clases Estudiante y Profesor para poder diferenciar los diferentes usuarios que se pueden registrar en la biblioteca y que estos hereden los atributos de la clase Usuario

Interface CreardorDeUsuarios

Decidí hacer esta interface para mostrar el polimorfismo de las clases Estudiante y Profesor al momento de registrar cada uno en la biblioteca a

Clase Libro

Aquí hice lo mismo que la clase Usuario , lo volví abstracta para que las clases Físico , Digital y Audiolibro puedan heredar los atributos de esta , además de tener atributos únicos para cada una y así diferenciar los tipos de libro en la biblioteca

Interface CreardorDeLibros

Igual que en CreadorDeUsuarios hice esta interface para crear el polimorfismo de los diferentes tipos de libros al momento de registrarlos en la biblioteca

Clase Prestamo

Decidí agregar los métodos realizarPrestamo y marcarDevolucion por que pienso que estas funciones son responsabilidad de esa clase y no de la BibliotecaApp

Clase GestionarBiblioteca

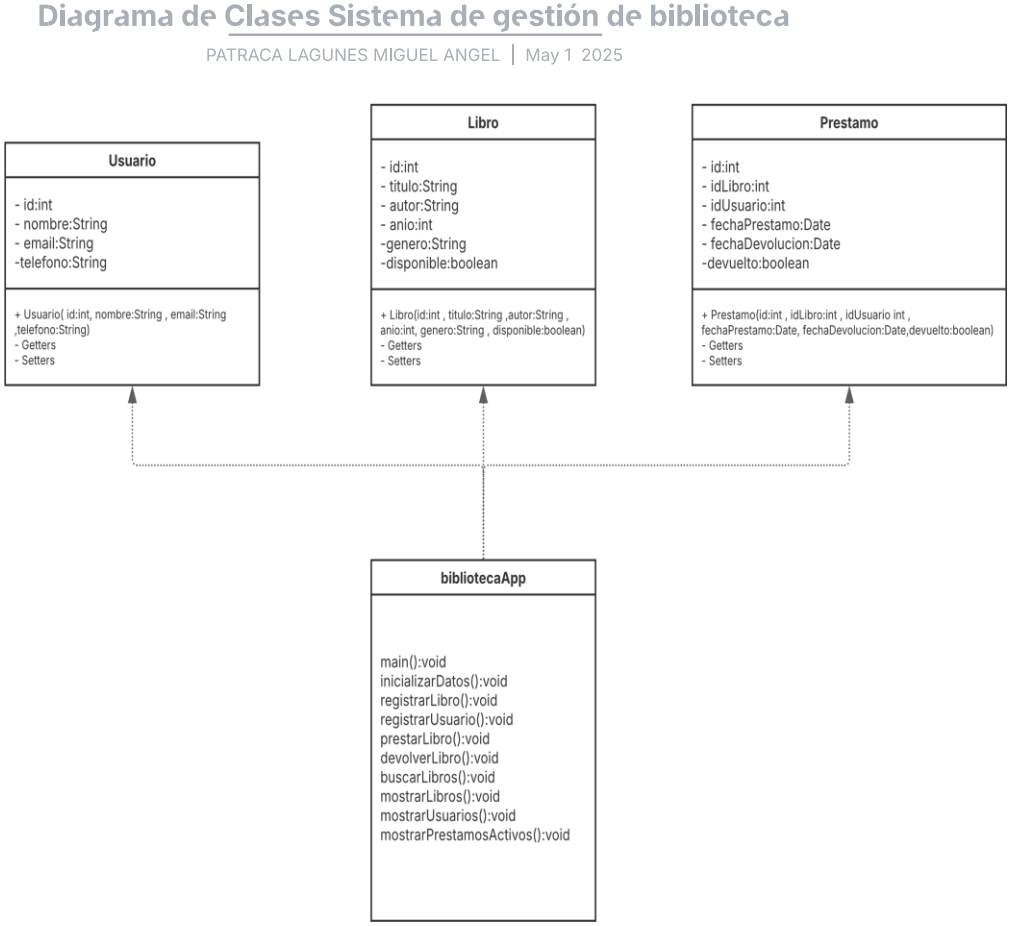
Cree esta clase para mantener el principio SOLID en cada clase, porque antes en la clase BibliotecaApp mantenía las funciones que podía hacer en la consola y las funciones que gestiona la biblioteca, así que moví todas las funciones a esta clase y las refactorice porque muchos métodos tenían mas de una responsabilidad y algunos bloques de código se repetían , además cree nuevos métodos para la buscqueda de un solo libro y usuario , lo registros para los tipos de usuario y libros

Clase BibliotecaApp

Como ya expliqué anteriormente decidí mantener en esta clase solo las salidas para el usuario y mover las funciones a una nueva clase para así separas las responsabilidades , aquí agrupe el menú de operaciones para que no abarque mucho código el método main

Diagrama de clases

Antes de refactorizar



Después de refactorizar