Relatório do Sistema de Gerenciamento de Biblioteca

Luan Mickael da Rocha

January 17, 2025

Abstract

Este relatório descreve o desenvolvimento de um sistema de gerenciamento de biblioteca utilizando os princípios da Orientação a Objetos (OO). O sistema foi projetado para gerenciar livros, usuários e empréstimos, aplicando conceitos fundamentais de OO como herança, polimorfismo e encapsulamento. Além disso, são discutidos os padrões de projeto utilizados e os motivos para sua escolha.

1 Introdução

O objetivo deste projeto foi criar um sistema de biblioteca utilizando conceitos de Orientação a Objetos. O sistema permite cadastrar livros, usuários, realizar empréstimos e gerir a devolução de livros. A escolha dos conceitos de OO facilita a criação de um sistema modular, reutilizável e de fácil manutenção.

2 Conceitos de Orientação a Objetos Aplicados

A Orientação a Objetos (OO) é um paradigma de programação que utiliza objetos e classes para organizar e manipular dados. No sistema desenvolvido, os principais conceitos de OO aplicados foram:

2.1 Herança

A herança foi utilizada para criar diferentes tipos de usuários no sistema, como ‘Aluno‘ e ‘Professor‘. Ambos herdam de uma classe abstrata ‘Usuario‘, que define comportamentos e atributos comuns, como o nome e a lista de empréstimos. Isso permite a reutilização de código e facilita a manutenção do sistema.

```
public class Aluno extends Usuario {
    public Aluno(String nome) {
        super(nome);
    }

    @Override
    public int getLimiteEmprestimos() {
        return 3; // Limite de 3 empréstimos para alunos
    }
}
```

Neste exemplo, ‘Aluno‘ herda a classe ‘Usuario‘ e sobrescreve o método ‘getLimiteEmprestimos()‘ para definir o limite de empréstimos específico para alunos.

2.2 Polimorfismo

O polimorfismo foi utilizado para permitir que o sistema trate objetos de diferentes classes de forma uniforme. A classe ‘Usuario‘ define o método abstrato ‘getLimiteEmprestimos()‘, que é implementado de maneira diferente para ‘Aluno‘ e ‘Professor‘. Isso permite que, ao chamar esse método em um objeto ‘Usuario‘, o comportamento específico seja escolhido em tempo de execução.

```
public class Professor extends Usuario {
    public Professor(String nome) {
        super(nome);
    }

    @Override
    public int getLimiteEmprestimos() {
        return 5; // Limite de 5 empréstimos para professores
    }
}
```

No código acima, o método ‘getLimiteEmprestimos()‘ é sobrescrito na classe ‘Professor‘, permitindo que o sistema trate tanto alunos quanto professores de forma polimórfica.

2.3 Encapsulamento

O encapsulamento foi utilizado para proteger os dados sensíveis e permitir que a interação com os objetos seja feita apenas por meio de métodos. Por exemplo, a classe ‘Livro‘ contém atributos privados como ‘titulo‘, ‘autor‘ e ‘disponivel‘, e oferece métodos públicos para acessar e modificar esses atributos.

```
public class Livro {
    private String titulo;
    private String autor;
    private boolean disponivel;

    public Livro(String titulo, String autor) {
        this.titulo = titulo;
        this.autor = autor;
        this.disponivel = true; // Por padrão, o livro está disponível
    }

    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 boolean isDisponivel() {
        return disponivel;
    }

    public void setDisponivel(boolean disponivel) {
        this.disponivel = disponivel;
    }
}
```

Neste caso, os atributos privados são acessados e modificados por meio de métodos 'get' e 'set', garantindo que o acesso aos dados seja controlado.

3 Padrões de Projeto Utilizados

No desenvolvimento do sistema, dois padrões de projeto foram utilizados para facilitar a implementação e garantir que o código fosse modular e reutilizável:

3.1 Factory Method

O padrão *Factory Method* foi utilizado para a criação de objetos 'Usuario'. Em vez de instanciar diretamente as subclasses 'Aluno' ou 'Professor', o sistema utiliza um método para criar o tipo de usuário correto com base na escolha do usuário no momento do cadastro.

```
public void cadastrarUsuario(Scanner scanner) {
    // Código omitido para brevidade
    if (tipoUsuario == 1) {
        usuario = new Aluno(nome); // Instancia um Aluno
    } else if (tipoUsuario == 2) {
        usuario = new Professor(nome); // Instancia um Professor
    }
}
```

Este padrão facilita a criação de novos tipos de usuários no futuro sem alterar a lógica de cadastro no código principal.

3.2 Strategy

O padrão *Strategy* pode ser observado na forma como os limites de empréstimos são tratados. A lógica para obter o limite de empréstimos é diferente para cada tipo de usuário (Aluno ou Professor), mas a interface 'Usuario' define o comportamento comum e delega a implementação específica para as subclasses. Esse padrão facilita a manutenção e a expansão do sistema.

4 Requisitos Não Implementados

O sistema foi desenvolvido com sucesso conforme os requisitos especificados, sem limitações significativas. No entanto, um requisito adicional que não foi implementado foi a funcionalidade de "multiplos livros por empréstimo" para os usuários. A implementação desse requisito não foi concluída devido a limitações de tempo, mas poderia ser facilmente integrada ao sistema, permitindo que um usuário possa pegar vários livros ao mesmo tempo, respeitando seu limite de empréstimos.

5 Conclusão

O sistema de gerenciamento de biblioteca foi desenvolvido com sucesso, utilizando os conceitos fundamentais de Orientação a Objetos, como herança, polimorfismo e encapsulamento. A estrutura modular e reutilizável do código facilita futuras modificações e expansões. O uso de padrões de projeto como Factory Method e Strategy contribuiu para a organização e flexibilidade do código, tornando-o mais robusto e fácil de manter.