UNIVERSIDADE LUTERANA DO BRASIL (ULBRA)
CURSO DE ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

KAILAN DE SCENA PACHECO

DESENVOLVIMENTO DE SISTEMA DE GERENCIAMENTO DE BIBLIOTECA UTILIZANDO PROGRAMAÇÃO ORIENTADA A OBJETOS (POO)

TORRES
2024

1. RESUMO

Este trabalho apresenta o desenvolvimento de um sistema de gerenciamento de biblioteca utilizando os conceitos da Programação Orientada a Objetos (POO) na linguagem C#. O sistema foi implementado para realizar o cadastro, consulta, empréstimo e devolução de livros, além de gerenciar os usuários da biblioteca.

Os quatro pilares da POO — encapsulamento, herança, polimorfismo e abstração — foram aplicados de maneira prática para garantir a modularidade e a escalabilidade do sistema. O encapsulamento protegeu os dados internos das classes, a herança permitiu a reutilização de atributos e métodos comuns, o polimorfismo trouxe flexibilidade ao sistema, e a abstração facilitou a modelagem de objetos complexos.

O resultado foi um sistema robusto, fácil de manter e com a possibilidade de expansão futura. O trabalho demonstrou a importância da POO no desenvolvimento de soluções complexas, proporcionando uma base sólida para a implementação de novos recursos.

Palavras-chave: Programação Orientada a Objetos, encapsulamento, herança, polimorfismo, abstração, C#, sistema de gerenciamento de biblioteca.

2. INTRODUÇÃO

A Programação Orientada a Objetos (POO) é uma abordagem fundamental para o desenvolvimento de sistemas robustos e escaláveis. Ao organizar o código em torno de objetos, que são instâncias de classes, a POO facilita a criação de soluções modulares e adaptáveis. Este trabalho descreve o desenvolvimento de um sistema de gerenciamento de biblioteca utilizando a linguagem C# e os conceitos centrais da POO.

O objetivo deste sistema é gerenciar o cadastro, a consulta, o empréstimo e a devolução de livros, além do controle de usuários. A POO foi aplicada de maneira prática, utilizando os seus quatro pilares — encapsulamento, herança, polimorfismo e abstração — para garantir que o código seja flexível e modular. O projeto visa demonstrar a eficácia dessa metodologia no desenvolvimento de soluções que precisam de fácil manutenção e capacidade de expansão.

3. DESENVOLVIMENTO

Nesta seção, será detalhada a aplicação dos quatro pilares da Programação Orientada a Objetos (POO) encapsulamento, herança, polimorfismo e abstração no desenvolvimento do sistema de gerenciamento de biblioteca. A seguir, cada pilar será explicado com exemplos práticos retirados do código implementado no projeto.

3.1 ENCAPSULAMENTO

Encapsulamento é um dos pilares essenciais da POO, pois protege os dados internos de uma classe e garante que seu acesso e modificação sejam feitos por meio de métodos definidos, controlando a manipulação direta desses dados. O encapsulamento também facilita a manutenção e segurança dos dados.

No sistema de gerenciamento de biblioteca, o encapsulamento foi amplamente utilizado na classe **`Livro`**. Atributos como **`QuantidadeEmEstoque`** foram definidos como privados, permitindo que seu acesso e modificação ocorram somente por meio de métodos específicos, como `Emprestar` e `Devolver`, garantindo que o controle de estoque seja feito de forma adequada.

Exemplo de código:

```
public class Livro : ItemBiblioteca
{
    2 referências
    public string Autor { get; set; }
    1 referência
    public string Isbn { get; set; }
    2 referências
    public string Genero { get; set; }
    6 referências
    public int QuantidadeEmEstoque { get; set; }

    0 referências
    public Livro(string autor, string isbn, string genero, int quantidadeEmEstoque, string titulo, string codigo)
        : base(titulo, codigo)
    {
        Autor = autor;
        Isbn = isbn;
        Genero = genero;
        QuantidadeEmEstoque = quantidadeEmEstoque;
    }

    // Implementa o metodo Emprestar
    2 referências
    public override void Emprestar(Usuario usuario)
    {
        if (QuantidadeEmEstoque > 0)
        {
            QuantidadeEmEstoque--;
            usuario.AdicionarEmprestimo(Titulo);
            Console.WriteLine($"Livro '{Titulo}' emprestado para {usuario.Nome}. Estoque restante: {QuantidadeEmEstoque}");
        }
        else
        {
            Console.WriteLine($"Não há exemplares disponíveis do livro '{Titulo}'.");
        }
    }

    // Implementa o metodo Devolver
    2 referências
    public override void Devolver()
    {
        QuantidadeEmEstoque++;
        Console.WriteLine($"Livro '{Titulo}' devolvido. Estoque atual: {QuantidadeEmEstoque}");
    }
}
```

Neste exemplo, o método **`Emprestar`** reduz a quantidade de exemplares em estoque, enquanto o método **`Devolver`** incrementa o número de exemplares disponíveis, controlando o acesso e garantindo a integridade do sistema.

3.2 HERANÇA

A herança permite que uma classe reutilize os atributos e métodos de outra, promovendo a economia de código e melhorando a organização do sistema. No projeto, a herança foi aplicada com a criação da classe **`ItemBiblioteca`**, que serve como uma classe base para todos os itens da biblioteca.

A classe `Livro` herda de **`ItemBiblioteca`**, reutilizando atributos comuns a todos os itens, como **`Titulo`** e **`Codigo`**, e adicionando novos atributos, como **`Autor`**,

`Genero` e **`QuantidadeEmEstoque`**. Isso evita a duplicação de código e facilita a manutenção.

Exemplo de código:

```
public abstract class ItemBiblioteca
{
    7 referências
    public string Titulo { get; set; }
    2 referências
    public string Codigo { get; set; }

    2 referências
    public abstract void Emprestar(Usuario usuario);

    //Empresta

    2 referências
    public abstract void Devolver();

    //Devolve
}
```

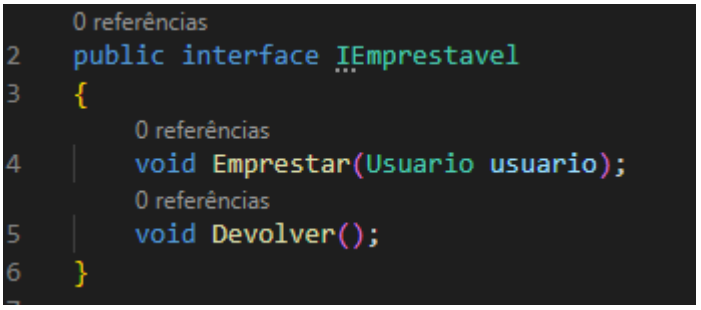
```
7 referências
public class Livro : ItemBiblioteca
{
    2 referências
    public string Autor { get; set; }
    1 referência
    public string Isbn { get; set; }
    2 referências
    public string Genero { get; set; }
    1 referência
    public int QuantidadeEmEstoque { get; set; }

    0 referências
    public Livro(string autor, string isbn, string genero, int quantidadeEmEstoque, string titulo, string codigo)
        : base(titulo, codigo)
    {
        Autor = autor;
        Isbn = isbn;
        Genero = genero;
        QuantidadeEmEstoque = quantidadeEmEstoque;
    }

    // Implementa o metodo Emprestar
    2 referências
    public override void Emprestar(Usuario usuario)
    {
      //Logica para realizar o empréstimo
    }

    // Implementa o metodo Devolver
    2 referências
    public override void Devolver()

    //Logica para Devolver
}
```

Essa estrutura facilita a inclusão de novos tipos de itens no futuro, como DVDs ou revistas, que também podem herdar da classe **`ItemBiblioteca`**, reutilizando os atributos e métodos comuns.

3.3 POLIMORFISMO

O polimorfismo permite que objetos de diferentes classes respondam de maneira distinta ao mesmo método, garantindo a flexibilidade do código. No sistema de gerenciamento de biblioteca, o polimorfismo foi implementado por meio da interface **`IEmprestavel`**, que define os métodos **`Emprestar`** e **`Devolver`**.

Exemplo de código:



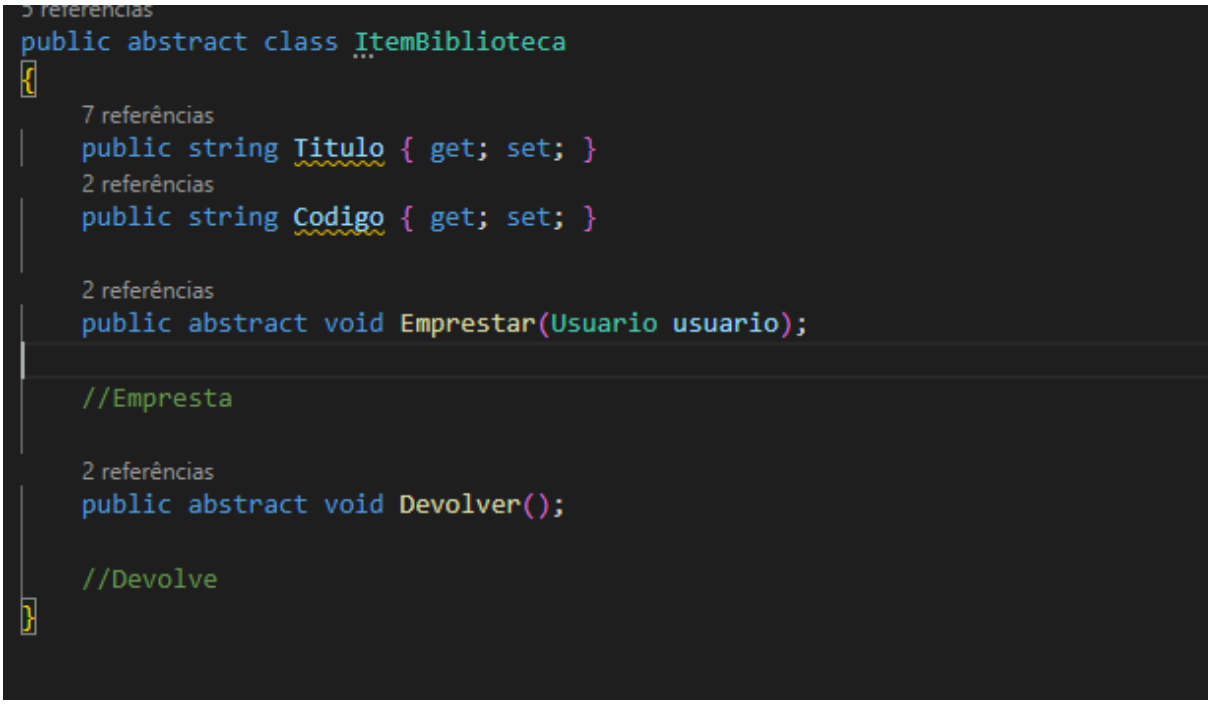
```
public class Livro : ItemBiblioteca
{
    2 referências
    public string Autor { get; set; }
    1 referência
    public string Isbn { get; set; }
    2 referências
    public string Genero { get; set; }
    6 referências
    public int QuantidadeEmEstoque { get; set; }

    0 referências
    public Livro(string autor, string isbn, string genero, int quantidadeEmEstoque, string titulo, string codigo)
        : base(titulo, codigo)
    {
        Autor = autor;
        Isbn = isbn;
        Genero = genero;
        QuantidadeEmEstoque = quantidadeEmEstoque;
    }

    // Implementa o método Emprestar
    2 referências
    public override void Emprestar(Usuario usuario)
    {
        if (QuantidadeEmEstoque > 0)
        {
            QuantidadeEmEstoque--;
            usuario.AdicionarEmprestimo(Titulo);
            Console.WriteLine($"Livro '{Titulo}' emprestado para {usuario.Nome}. Estoque restante: {QuantidadeEmEstoque}");
        }
        else
        {
            Console.WriteLine($"Não há exemplares disponíveis do livro '{Titulo}'.");
        }
    }

    // Implementa o metodo Devolver
    2 referências
    public override void Devolver()
    {
        QuantidadeEmEstoque++;
        Console.WriteLine($"Livro '{Titulo}' devolvido. Estoque atual: {QuantidadeEmEstoque}");
    }
}
```

Neste exemplo, a classe **`Livro`** implementa a interface **`IEmprestavel`**, garantindo que qualquer item da biblioteca que possa ser emprestado siga a mesma estrutura para o processo de empréstimo e devolução.

3.4 ABSTRAÇÃO

A abstração é o processo de simplificação dos objetos, ocultando detalhes desnecessários e focando apenas nas suas características essenciais. No projeto, a classe **`ItemBiblioteca`** é uma abstração para todos os itens da biblioteca, como livros e revistas. Essa classe define atributos como **`Titulo`** e **`Codigo`**, além de métodos abstratos, como **`Emprestar`** e **`Devolver`**.

Exemplo de código:



Essa abstração torna o sistema mais flexível e fácil de expandir, permitindo que novas classes herdem de **`ItemBiblioteca`** e implementem suas próprias lógicas de empréstimo e devolução sem modificar a estrutura principal.

4. CONCLUSÃO

O desenvolvimento deste sistema de gerenciamento de biblioteca proporcionou uma compreensão aprofundada dos quatro pilares da Programação Orientada a Objetos e de como eles podem ser aplicados para criar soluções robustas e escaláveis. A aplicação prática de encapsulamento, herança, polimorfismo e abstração garantiu que o código fosse modular e fácil de manter. Além disso, o uso desses princípios facilitou a implementação de novas funcionalidades e a reutilização de código.

Este projeto mostrou-se um exemplo claro de como a POO pode simplificar a criação de sistemas complexos, permitindo que o código seja adaptável e extensível para atender a novas demandas no futuro.

5. REFERÊNCIAS

LIMA, Leandro dos Santos. **Programação Orientada a Objetos em C#: Conceitos e Aplicações**. 2. ed. São Paulo: Novatec, 2019.

BARBOSA, Maurício Samy Silva. **Dominando a Programação Orientada a Objetos com C#**. 1. ed. São Paulo: Editora Érica, 2017.

XAVIER, Ricardo de Almeida. **Lógica de Programação e Estruturas de Dados em C#: Da Base à Orientação a Objetos**. 1. ed. São Paulo: Novatec, 2020.

SOARES, Paulo Sergio. **Programação Orientada a Objetos com Exemplos Práticos em C#**. 1. ed. Rio de Janeiro: Alta Books, 2016.

FERREIRA, Tiago. **Desenvolvimento de Aplicações com Programação Orientada a Objetos em C#**. São Paulo: Editora Atlas, 2018.