AP2-Fogaça

Introdução

Durante o desenvolvimento do projeto da AP2, tive a oportunidade de fazer pesquisas mais profundas nos conceitos da Programação Orientada a Objetos (POO). Essa programação se destaca por sua capacidade de organizar e modularizar o código, o que facilita a manutenção e maior entendimento dos sistemas. No meu projeto, busquei implementar funcionalidades que permitem gerenciar livros, usuários e operações de empréstimo, aplicando os princípios da POO para criar uma solução eficiente.

Os quatro pilares da POO—encapsulamento, herança, polimorfismo e abstração—são fundamentais para entender como construir o codigo de maneira eficaz. Vou explicar cada um desses conceitos e como os apliquei no meu projeto, trazendo exemplos de código que ajudam a ilustrar esses princípios.

Os Quatro Pilares da Programação Orientada a Objetos

Encapsulamento

O encapsulamento foi um dos conceitos que mais me chamou a atenção. Ele permite que os dados de um objeto sejam protegidos de acessos indesejados, mantendo a integridade das informações. Em meu projeto, a classe Livro exemplifica bem esse conceito. Os atributos dessa classe são definidos como private, o que significa que não podem ser acessados diretamente de fora da classe. Para interagir com esses atributos, criei métodos public.

```
public class Livro
{
    private string titulo;
    private string autor;
    private string isbn;
    private string genero;
    private int quantidadeEstoque;

    public Livro(string titulo, string autor, string isbn, string genero, int quantidadeEstoque)
```

```
    {
        this.titulo = titulo;
        this.autor = autor;
        this.isbn = isbn;
        this.genero = genero;
        this.quantidadeEstoque = quantidadeEstoque;
    }

    public string Titulo => titulo; // Propriedade somente leitura
    public int QuantidadeEstoque
    {
        get => quantidadeEstoque;
        set
        {
            if (value >= 0)
                quantidadeEstoque = value;
        }
    }
}
```

Com isso, consegui garantir que os dados dos livros fossem manipulados de forma segura, evitando que outros códigos pudessem alterá-los sem passar pelos métodos que controlei.

Herança

A herança é um recurso muito interessante, pois permite criar novas classes com base em classes existentes, o que economiza tempo e esforço na programação. No meu sistema, pensei em como poderia usar herança para criar tipos específicos de livros. Embora o exemplo não tenha sido implementado no código final, imaginei uma classe LivroInfantil que herda da classe Livro.

```
public class LivroInfantil : Livro
{
```

```
    public string FaixaEtaria { get; set; }


    public LivroInfantil(string titulo, string autor, string isbn, string genero, int
quantidadeEstoque, string faixaEtaria)
        : base(titulo, autor, isbn, genero, quantidadeEstoque)
    {
        FaixaEtaria = faixaEtaria;
    }
}
```

Dessa forma, LivroInfantil poderia ter atributos e métodos adicionais, mas ainda aproveitando tudo que a classe Livro oferece. Isso me ajudou a entender como a herança pode tornar o código mais organizado e reutilizável.

Polimorfismo

O polimorfismo me ensinou a flexibilidade no código. Ele permite que objetos de diferentes classes respondam a uma mesma chamada de método de maneiras diferentes. No projeto, embora eu não tenha usado achei interessante pesquisar sobre.

Abstração

Por fim, a abstração me ajudou a entender como simplificar a complexidade do código. Ao usar interfaces ou classes abstratas, conseguimos representar conceitos complexos de forma mais acessível. Em meu projeto, criei uma interface chamada IEmprestimo, que define operações básicas relacionadas a empréstimos de livros.

```
public interface IEmprestimo
{
    void RealizarEmprestimo();
    void DevolverLivro();
}
```

Essa interface força as classes que a implementam a fornecer uma implementação específica para os métodos definidos, promovendo uma forma de organizar o código e destacando as operações essenciais do sistema.

Conclusão

Ao longo do desenvolvimento da AP2, aprendi muito sobre os princípios da Programação Orientada a Objetos. O uso do encapsulamento, herança, polimorfismo e abstração me ajudou a criar um código mais estruturado e fácil de entender. Essa experiência não apenas melhorou minhas habilidades de programação, mas também me deu uma visão mais clara de como construir sistemas de forma eficiente.

Com certeza, esse aprendizado será fundamental para projetos futuros e me motivou a continuar explorando as melhores práticas em desenvolvimento de software. Sinto que a POO é uma ferramenta poderosa e espero aplicar esses conceitos em outros desafios que surgirem.