Programação orientada a objetos

Bernardo Negrini Borba

Introdução

A Programação Orientada a Objetos (POO) permite que os sistemas sejam desenvolvidos de maneira modular, escalável e reutilizável, facilitando a manutenção e a compreensão do código. Os quatro pilares da POO – Encapsulamento, Herança, Polimorfismo e Abstração – fornecem uma estrutura robusta que promove boas práticas de programação, permitindo o desenvolvimento de soluções mais eficazes. Neste texto detalharei cada um desses pilares, juntamente com sua aplicação no projeto de gerenciamento de biblioteca.

Neste projeto, desenvolvi um sistema de gerenciamento de biblioteca utilizando a linguagem C#. O sistema permite o cadastro, consulta e empréstimo de livros e gerenciamento de usuários. A aplicação desses conceitos no desenvolvimento do sistema proporciona um entendimento prático dos benefícios oferecidos pela POO, como a separação de responsabilidades e a flexibilidade para futuras expansões.

Encapsulamento

O encapsulamento refere-se à prática de esconder os detalhes de implementação de um objeto e expor apenas o necessário para o uso externo. Isso é feito através da definição de modificadores de acesso (como private, protected e public), que controlam a visibilidade dos atributos e métodos de uma classe.

No sistema de gerenciamento de biblioteca, o encapsulamento foi aplicado nas classes Livro e Usuario, protegendo suas propriedades com modificadores de acesso adequados. Por exemplo, as propriedades Titulo, Autor, ISBN e QuantidadeEmEstoque da classe Livro são acessadas apenas por meio de métodos públicos, evitando modificações indevidas diretamente:

```
public class Livro
{
    public string Titulo { get; set; }
    public string Autor { get; set; }
    public string ISBN { get; set; }
    public int QuantidadeEmEstoque { get; private set; }
```

```
    public Livro(string titulo, string autor, string isbn, int
quantidade)
    {
        Titulo = titulo;
        Autor = autor;
        ISBN = isbn;
        QuantidadeEmEstoque = quantidade;
    }
}
```

Nesse trecho, a propriedade QuantidadeEmEstoque é encapsulada e só pode ser modificada internamente pela classe, que garante que seu valor só será alterado de acordo com a lógica do sistema, como durante o empréstimo ou devolução de um livro.

Herança

A herança permite que uma classe herde atributos e métodos de outra classe, promovendo o reaproveitamento de código e a redução da redundância. Em C#, a herança é implementada utilizando a palavra-chave “ : ”, que indica que uma classe está herdando de outra.

No projeto, a classe Livro herda da classe abstrata ItemBiblioteca, o que permite que métodos genéricos como Emprestar e Devolver sejam definidos em ItemBiblioteca, mas especializados na classe Livro. Isso pode ser observado no seguinte trecho de código:

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

    public abstract void Emprestar(Usuario usuario);
    public abstract void Devolver();
}

public class Livro : ItemBiblioteca
{
    public string Autor { get; set; }
    public string ISBN { get; set; }
    public int QuantidadeEmEstoque { get; private set; }
```

```
    public Livro(string titulo, string codigo, string autor, string
isbn, int quantidade)
        : base(titulo, codigo)
    {
        Autor = autor;
        ISBN = isbn;
        QuantidadeEmEstoque = quantidade;
    }

    public override void Emprestar(Usuario usuario)
    {
        if (QuantidadeEmEstoque > 0)
        {
            QuantidadeEmEstoque--;
            Console.WriteLine($"Livro '{Titulo}' emprestado a
{usuario.Nome}.");
        }
        else
        {
            Console.WriteLine("Livro indisponível.");
        }
    }

    public override void Devolver()
    {
        QuantidadeEmEstoque++;
        Console.WriteLine($"Livro '{Titulo}' devolvido.");
    }
}
```

Aqui, a classe Livro herda as propriedades Titulo e Codigo da classe ItemBiblioteca, além de implementar os métodos abstratos Emprestar e Devolver, que são obrigatórios devido à herança. Isso exemplifica o uso de herança para compartilhar comportamento comum entre diferentes tipos de itens na biblioteca, como CDs ou DVDs (que poderiam ser adicionados como outras classes derivadas de ItemBiblioteca).

Polimorfismo

O polimorfismo permite que objetos de diferentes classes sejam tratados de forma uniforme, desde que herdem de uma mesma classe base ou implementem a mesma interface. Esse comportamento é extremamente útil para criar sistemas flexíveis e extensíveis.

No sistema de gerenciamento de biblioteca, o polimorfismo é aplicado por meio do uso de métodos sobrescritos, como Emprestar e Devolver, que são definidos na classe base ItemBiblioteca e implementados de maneira diferente na classe Livro. Dessa forma, ao chamar Emprestar para qualquer objeto do tipo ItemBiblioteca, o sistema automaticamente executa a versão correta do método, conforme o tipo do objeto real.

```
public abstract class ItemBiblioteca
{
    public abstract void Emprestar(Usuario usuario);
    public abstract void Devolver();
}
```

O polimorfismo também pode ser aplicado com o uso de interfaces, como IEmprestavel e IPesquisavel, que definem contratos para as classes que as implementam. Isso permite que a biblioteca trate diferentes tipos de objetos de maneira uniforme, contanto que sigam o contrato das interfaces.

```
public interface IEmprestavel
{
    void Emprestar(Usuario usuario);
    void Devolver();
}

public interface IPesquisavel
{
    List<Livro> PesquisarPorTitulo(string titulo);
}
```

Abstração

A abstração envolve a simplificação da complexidade, permitindo que o programador foque nos aspectos essenciais de um objeto sem se preocupar com os detalhes de sua implementação. Em termos práticos, a abstração é frequentemente realizada através de classes abstratas e interfaces, que definem um comportamento genérico que pode ser implementado por classes concretas.

No projeto, a classe abstrata ItemBiblioteca define a estrutura básica para itens que podem ser emprestados na biblioteca, mas não pode ser instanciada diretamente. Somente classes concretas, como Livro, podem implementar essa estrutura e fornecer detalhes específicos para cada tipo de item.

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

    public abstract void Emprestar(Usuario usuario);
    public abstract void Devolver();
}
```

A classe ItemBiblioteca funciona como uma abstração de qualquer item que a biblioteca possa gerenciar, permitindo que futuras expansões adicionem novos tipos de itens (como DVDs ou revistas) sem precisar modificar o sistema principal, respeitando o princípio da responsabilidade única.

Conclusão

O desenvolvimento deste projeto de gerenciamento de biblioteca permitiu aplicar os princípios da programação orientada a objetos de forma prática, utilizando Encapsulamento, Herança, Polimorfismo e Abstração para criar um sistema modular, organizado e flexível. A experiência reforçou a compreensão da importância desses conceitos e de como sua aplicação contribui para a criação de softwares mais robustos, fáceis de manter e escaláveis.