Introdução

O sistema desenvolvido é capaz de gerenciar o cadastro de livros e usuários. O uso da Programação Orientada a Objetos neste projeto permitiu não apenas uma estrutura organizada, mas também a adição de novas funcionalidades. A seguir, serão apresentados os princípios da Programação Orientada a Objetos e como cada um foi aplicado no projeto.

Encapsulamento

O encapsulamento é uma das características mais fundamentais da Programação Orientada a Objetos e se refere à como ocultar os detalhes internos de um objeto, mostrando apenas o que é necessário para o uso externo. Essa prática tem como objetivo proteger os dados e garantir que eles sejam alterados de maneira controlada, evitando erros

No projeto de gerenciamento de biblioteca, o encapsulamento foi aplicado em várias classes. Por exemplo, na classe `Usuario`, as propriedades como `Nome`, `NumeroIdentificacao`, `Endereco` e `Contato` foram encapsuladas usando métodos `get` e `set`:

```
public class Usuario

{

public string Nome { get; set; }

public string NumeroIdentificacao { get; set; }

public string Endereco { get; set; }

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

Herança

A herança é um mecanismo que permite que uma classe herde propriedades e métodos de outra classe. Isso facilita a reutilização de código e a manutenção.

No projeto, a classe `ItemBiblioteca` foi definida como uma classe abstrata que serve de base para diferentes tipos de itens, como livros ou outros possíveis itens no futuro (ex.

revistas, DVDs). A classe `Livro` herda de `ItemBiblioteca`, e adiciona os métodos abstratos da classe base:

```
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; set; }

    public override void Emprestar(Usuario usuario)
    {
        if (QuantidadeEmEstoque > 0)
        {
            QuantidadeEmEstoque--;
            // Registrar o empréstimo
        }
    }
```

```
    public override void Devolver()
    {
        QuantidadeEmEstoque++;
        // Atualizar o registro de devolução
    }
}
```

Essa estrutura de herança permitiu a criação de diferentes tipos de itens na biblioteca, utilizando métodos e atributos comuns. A classe `ItemBiblioteca` contém as propriedades `Titulo` e `Codigo`, que são aplicáveis a todos os itens da biblioteca, enquanto a classe derivada `Livro` torna essas funcionalidades com propriedades específicas, como `Autor` e `ISBN`.

Polimorfismo

O polimorfismo é a capacidade de um objeto assumir diferentes formas, permitindo que um mesmo método possa ser utilizado por diferentes tipos de objetos, com comportamentos diferentes. Isso promove facilidade no código e a manutenção, uma vez que uma mesma interface pode ser utilizada para diferentes implementações.

No projeto, o polimorfismo foi aplicado através da implementação dos métodos abstratos `Emprestar` e `Devolver` na classe `Livro`. Esses métodos são definidos na classe base `ItemBiblioteca` como abstratos, mas possuem comportamentos específicos quando implementados nas subclasses. O sistema poderia ser estendido para outros tipos de itens, como revistas, sem alterar a lógica de empréstimos e devoluções.

```
public override void Emprestar(Usuario usuario)
{
    if (QuantidadeEmEstoque > 0)
    {
        QuantidadeEmEstoque--;
        // Log de empréstimo
    }
    else
    {
        Console.WriteLine("Livro indisponível.");
    }
}
```

Abstração

A abstração envolve a simplificação de conceitos complexos, ocultando os detalhes de implementação e exibindo apenas os aspectos essenciais. O objetivo é facilitar uma interface clara para o usuário ou Dev, sem a necessidade de conhecer os detalhes internos de como um processo é realizado.

No projeto, a abstração foi utilizada na classe `ItemBiblioteca`, que define métodos abstratos que devem ser adicionados pelas classes derivadas. A classe `ItemBiblioteca` abstrai o conceito de um item da biblioteca, sem se preocupar com os detalhes específicos de como cada tipo de item (livro, revista, etc.) deve ser tratado. As subclasses, como `Livro`, implementam esses detalhes de acordo com suas necessidades

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

Essa abstração permitiu que o sistema de gerenciamento de biblioteca fosse projetado de forma simples, possibilitando a adição de novos tipos de itens, sem a necessidade de modificar o código existente.

Conclusão

O desenvolvimento do sistema de gerenciamento de biblioteca proporcionou uma visão prática e aprofundada sobre Programação Orientada a Objetos. Através do encapsulamento, foi possível proteger os dados e garantir uma manipulação controlada. A herança e o polimorfismo permitiram a reutilização de código e a simplicidade na implementação de novos tipos de itens na biblioteca, enquanto a abstração forneceu uma interface simplificada para o gerenciamento de empréstimos e devoluções.

A experiência de utilizar esses conceitos em um projeto real reforçou a importância da Programação Orientada a Objetos no desenvolvimento de sistemas e de fácil manutenção. Este projeto servirá como base para o desenvolvimento de sistemas mais complexos no futuro.