Introdução

A Programação Orientada a Objetos (POO) é um dos paradigmas de desenvolvimento de software mais difundidos e amplamente utilizados. Ela organiza o código em torno de objetos, que podem ser instâncias de classes, permitindo uma melhor modelagem de problemas do mundo real e promovendo a reutilização de código, modularidade e facilidade de manutenção. Este paradigma é baseado em quatro pilares fundamentais: encapsulamento, herança, polimorfismo e abstração.

Neste trabalho, exploraremos o desenvolvimento de um sistema de gerenciamento de uma biblioteca, destacando como cada um desses pilares foi aplicado. O projeto envolve classes como `Biblioteca`, `Livro`, `Usuario`, além de interfaces como `IEmprestavel` e `IPesquisavel`, demonstrando os conceitos fundamentais da POO.

1. Encapsulamento

O encapsulamento é o princípio que permite esconder os detalhes internos da implementação de uma classe, expondo apenas o que é necessário para seu uso. Isso garante que os atributos e métodos internos de um objeto sejam acessíveis apenas por meio de interfaces controladas, protegendo-os de modificações indevidas e facilitando a manutenção.

No projeto, o encapsulamento pode ser visto na classe `Usuario`, onde os atributos são declarados como `public` ou `private`. Por exemplo, o atributo `HistoricoEmprestimos`, que armazena os livros emprestados por um usuário, é privado, garantindo que apenas a própria classe `Usuario` possa manipulá-lo diretamente:

```
private List<Livro> HistoricoEmprestimos = new List<Livro>();
```

Além disso, métodos públicos como `ExibirInformacoes` são expostos para permitir o acesso seguro às informações do usuário:

```
public void ExibirInformacoes(string numeroIdentificador)
{
    Console.WriteLine($"Nome: {Nome} \\n Identificador:
{NumeroIdentificador} \\n Endereço: {Endereco} \\n Contato:
{Contato}");
}
```

Esse controle de acesso protege os dados e garante que o sistema opere de maneira consistente.

2. Herança

A herança permite que uma classe herde as características (atributos e métodos) de outra, promovendo a reutilização de código e a criação de hierarquias de classes. No projeto, a classe `Livro` herda de uma classe base chamada `ItemBiblioteca`. Embora o código de `ItemBiblioteca` não tenha sido fornecido, o conceito de herança é evidente no fato de que `Livro` estende essa classe, provavelmente herdando atributos como `Titulo` e `Codigo`.

```
public class Livro : ItemBiblioteca, IEmprestavel, IPesquisavel
{
    public string Autor { get; set; }
    public string ISBN { get; set; }
    public string Genero { get; set; }
    public int QuantidadeEmEstoque { get; set; }
}
```

Através da herança, a classe `Livro` pode adicionar novos atributos, como `Autor` e `ISBN`, além de implementar as interfaces `IEmprestavel` e `IPesquisavel`, o que demonstra também a flexibilidade da herança em conjunto com outros pilares.

3. Polimorfismo

O polimorfismo permite que objetos de diferentes classes sejam tratados de maneira uniforme, desde que implementem os mesmos métodos ou pertençam à mesma hierarquia de classes. Isso é especialmente útil quando se trabalha com interfaces, pois classes diferentes podem fornecer implementações específicas para os métodos definidos por uma interface comum.

No projeto, o polimorfismo é utilizado por meio das interfaces `IEmprestavel` e `IPesquisavel`. A classe `Livro` implementa essas duas interfaces, o que significa que ela pode ser tratada como um objeto de tipo `IEmprestavel` ou `IPesquisavel`, dependendo do contexto. Por exemplo, ao emprestar um livro, o sistema invoca o método `Emprestar`:

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

Essa implementação permite que o mesmo método `Emprestar` possa ser aplicado a diferentes tipos de objetos que implementem a interface, sem a necessidade de conhecer os detalhes da implementação específica de cada um.

4. Abstração

A abstração é o princípio de ocultar os detalhes complexos de um sistema, fornecendo uma interface simples para o uso das funcionalidades. Ela permite que os desenvolvedores se concentrem nas operações importantes sem se preocupar com a implementação detalhada de cada uma delas.

No projeto, a abstração é vista no uso de interfaces, como `IPesquisavel`, que define métodos genéricos para pesquisa sem se preocupar com a implementação exata. A classe `Livro` implementa essa interface, mas a lógica de como a pesquisa é feita é abstraída:

```
public interface IPesquisavel
{
    List<Livro> PesquisarPorTitulo(string titulo, List<Livro>
livros);
    List<Livro> PesquisarPorAutor(string autor, List<Livro> livros);
    List<Livro> PesquisarPorGenero(string genero, List<Livro>
livros);
}
```

Ao utilizar a interface, o sistema pode realizar pesquisas de livros com base em diferentes critérios (título, autor, gênero) sem precisar modificar o código que invoca esses métodos, tornando o sistema mais flexível e fácil de expandir.

Conclusão

O desenvolvimento deste projeto permitiu uma aplicação prática dos quatro pilares da Programação Orientada a Objetos. Através da implementação de classes, interfaces e a interação entre elas, foi possível construir um sistema modular, extensível e fácil de manter. A utilização de conceitos como encapsulamento, herança, polimorfismo e abstração não só melhorou a organização do código, mas também facilitou a sua reutilização e a adição de novas funcionalidades.

Referências Bibliográficas

- DEITEL, H. M.; DEITEL, P. J. **Java: Como Programar**. 10ª ed. São Paulo: Pearson Prentice Hall, 2015.
- SOMMERVILLE, I. **Engenharia de Software**. 10ª ed. São Paulo: Pearson, 2011.