Biblioteca OOP

O projeto **Biblioteca OOP** teve como principal objetivo desenvolver um sistema de gerenciamento de uma biblioteca, tornando possível o cadastro, consulta e empréstimo de livros, assim como o gerenciamento de usuários. O projeto foi baseado na Programação Orientada a Objetos.

A **Programação Orientada a Objetos** organiza o código em objetos, que representam elementos do mundo real, facilitando o desenvolvimento e expansão do sistema. Ela torna o código mais reutilizável e de fácil entendimento. Ela possui **quatro pilares** que são essenciais para o desenvolvimento do software de maneira eficiente. São eles:

Encapsulamento: É um dos pilares responsáveis por proteger os dados internos de um objeto, controlando o acesso e assegurando a integridade e segurança do sistema. É importante utilizar o Encapsulamento pois ele protege os dados, facilita a reutilização de código, e é de fácil manutenção.

Está presente nos modificadores de acesso (**public, private e protected**), que controlam os atributos e métodos de uma classe. Os atributos declarados como private, só podem ser acessados dentro da própria classe. Os métodos de uma interface, são públicos, permitindo que os objetos acessem os dados de forma controlada.

Outro exemplo muito comum de encapsulamento são os métodos Getters e Setters, que são responsáveis por obter(**get**) e modificar(**set**) o valor de atributos privados. Permite que o usuário valide os dados antes de atribui-los.

Exemplos de Encapsulamento no código:

Na classe **ItemBiblioteca:**

```
public string Titulo { get; set; }
public string Codigo { get; set; }
```

Aqui os acessos aos atributos estão públicos, mas são validados através do get e set. O mesmo ocorre também na classe **Livro**, e **Usuário**.

Na classe **Livro:**

```
public string Autor { get; set; }
public string ISBN { get; set; }
public string Genero { get; set; }
public int QuantidadeEmEstoque { get; set; }
```

.Na classe **Usuário:**

```
public string Nome { get; set; }
public string NumeroIdentificacao { get; set; }
public string Endereco { get; set; }
public string Contato { get; set; }
```

Herança: Outro pilar importantíssimo na Programação Orientada a Objetos. A herança permite a criação de classes que herdam os atributos e métodos de outras classes, uma relação semelhante a de pai e filho. É fundamental na reutilização do código, evitando a duplicação e facilitando a manutenção e organização.

Exemplos de Herança no código:

```
public class Livro : ItemBiblioteca, IEmprestavel, IPesquisavel
```

Aqui a classe **Livro** está herdando da classe ItemBiblioteca, que contém as propriedades Titulo, Código e os métodos Emprestar e Devolver.

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

No construtor da classe Livro, utilizamos : base(titulo, codigo), para referenciar as propriedades que estão presentes na classe ItemBiblioteca.

Polimorfismo: O polimorfismo(do grego muitas formas) permite que os objetos de classes diferentes responderem a mesma mensagem de diferentes maneiras, e isso fica muito evidente na hora de sobrescrever os métodos.

Exemplos de Polimorfismo no código:

No código, a classe Livro está herdando da classe ItemBiblioteca, como visto acima, dentro dessa classe existem dois métodos abstratos, Emprestar e Devolver:

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

public abstract void Devolver();
```

Dentro da classe **Livro**, os métodos abstratos Emprestar e Devolver são implementados através de sobrescrita:

```
        public override void Emprestar(Usuario usuario)
        {
            if (QuantidadeEmEstoque > 0)
            {
                QuantidadeEmEstoque--;
```

```
                usuario.AdicionarLivroEmprestimo(this);
                        Console.WriteLine($"O livro {Titulo} foi emprestado ao usuário
{usuario.Nome}");
            }
            else
            {
                Console.WriteLine("O livro não está disponível em nosso estoque");
            }
        }

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

Ou seja, o método é escrito de maneira diferente dentro da classe Livro, mas ainda é o mesmo método de ItemBiblioteca..

Abstração: A abstração é responsável por esconder os detalhes internos complexos e mostrar apenas a parte essencial para a utilização de um método ou objeto, simplificando a interação.

No código a classe ItemBiblioteca é declarada como abstrata, além dos métodos Emprestar e Devolver, que é declarado sem uma implementação detalhada, deixando em aberto para a classe que for herdar implementar da maneira que achar melhor, basicamente definindo que os itens da biblioteca devem realizar a ação de emprestar e devolver, mas não deixando claro como se faz isso.

Exemplo de Abstração no código: A **classe ItemBiblioteca** e seus **métodos Emprestar e Devolver**.

```
    public abstract class ItemBiblioteca{

        //propriedades comuns
        public string Titulo { get; set; }

        public string Codigo { get; set; }

        //metodos abstratos;
```

```
        public ItemBiblioteca(string titulo, string codigo)
        {
            Titulo = titulo;
            Codigo = codigo;
        }

        public abstract void Emprestar(Usuario usuario);

        public abstract void Devolver();
    }
```

O desenvolvimento do projeto **Biblioteca OOP** se demonstrou muito importante no quesito de fixação e melhor entendimento dos conceitos de POO, na melhora da prática da sintaxe da linguagem C# e também foi responsável por aumentar a autonomia, já que foi necessário realizar pesquisas para entender melhor os conceitos.

Os quatro pilares da Programação Orientada a Objetos(Encapsulamento, Herança, Polimorfismo e Abstração) foram aplicados, permitindo a reutilização do código, integridade dos dados, permitir a personalização do comportamento dos objetos e também facilitar futuras modificações no sistema.

Referências

https://learn.microsoft.com/pt-br/dotnet/csharp/fundamentals/object-oriented/polymorphism
https://learn.microsoft.com/pt-br/dotnet/csharp/fundamentals/object-oriented/inheritance
https://www.macoratti.net/12/06/c_caip1.htm
https://learn.microsoft.com/pt-br/dotnet/csharp/language-reference/keywords/abstract
https://learn.microsoft.com/pt-br/dotnet/csharp/language-reference/keywords/override
https://learn.microsoft.com/pt-br/dotnet/csharp/language-reference/keywords/virtual
https://learn.microsoft.com/pt-br/dotnet/csharp/language-reference/keywords/static