

LABORATÓRIO DE PROGRAMAÇÃO

ALUNA: Eduarda dos Santos Matos

Turma:3N

Data: 23/10/2024

Avaliação Parcial 2: Projeto C#

O projeto se trata de uma biblioteca que desenvolve as funções de Cadastrar Livros, cadastrar usuários, listar livros, listar usuários, pesquisar livro por título, pesquisar autor, pesquisar gênero.

Foi desenvolvido na linguagem C# e tem alguns requisitos a serem cumpridos, são eles, uma classe abstrata, um método abstrato, propriedades com “get” e “set”, usar os modificadores “public”, “protected” e “private”, classes com herança, propriedades específicas, implementação de métodos abstratos.

Com objetivo de ampliar o conhecimento na linguagem e entender mais sobre os pilares da Programação Orientada a Objetos que é muito importante pois se trata de um modelo de programação essencial que estrutura o código em torno de “objetos”, em vez de se apoiar apenas em funções e lógica.

Encapsulamento

O encapsulamento é um dos preceitos básicos da Programação Orientada a Objetos (POO) e diz respeito ao ato de juntar dados (atributos) e funções que interagem entre si dentro de uma única unidade, chamada de classe. Esta metodologia apresenta vários benefícios, tanto na estruturação do código quanto na segurança e na manutenção.

Uma das características fundamentais do encapsulamento é a proteção dos dados. Ao encapsular os atributos de uma classe, os programadores têm a capacidade de restringir o acesso a esses dados, possibilitando que somente métodos específicos dentro da classe possam interagir com eles. Normalmente, isso é realizado através do uso de modificadores de acesso, tais como private, protected e public. Por exemplo, se um atributo for declarado como private, ele não poderá ser acessado diretamente de fora da classe.

No Código

Na linha 91 da classe Biblioteca temos a função de listar usuário que utiliza do modificador de acesso “protected”, fazendo com que esta lista de usuários esteja protegida de modificações. Para o acesso desta função, foi adicionada outra função, com o modificador de acesso “public”. Confira na imagem a baixo.

```

91     protected void ListarUsuarios()
92     {
93         //puxando da lista usuarios
94         foreach (var usuario in usuarios)
95         {
96             //sera listado cada atributo
97             Console.WriteLine($"Nome:{usuario.Nome}");
98             Console.WriteLine($"Numero de Identificação: {usuario.NumIdentificacao}");
99             Console.WriteLine($"Endereço:{usuario.Endereco}");
100            Console.WriteLine($" Contato:{usuario.Contato}");
101            Console.WriteLine("-----");
102        }
103    }
104    //possibilita acesso a lista de usuarios
105    1 referência
106    public void ExibirUsuarios()
107    {
108        ListarUsuarios();
109    }

```

Na linha 5 da classe biblioteca é usado o “private” para lista de livros e usuários. Isso significa que eles não podem ser acessados diretamente fora da classe Biblioteca.

Abaixo temos o construtor que é responsável por fazer com que as listas estejam prontas para uso. Confira na imagem a baixo.

```

5     private List<Livro> livros;
6     private List<Usuario> usuarios;
7
8     //construtor
9     1 referência
10    public Biblioteca()
11    //sendo iniciada uma lista de livros e usuarios
12    {
13        livros = new List<Livro>();
14        usuarios = new List<Usuario>();
15    }

```

Herança

A Herança é um pilar da POO, ela permite que uma classe (classe derivada) herde propriedades e comportamentos de outra classe (classe base). Isso permite a reutilização de código, a criação de hierarquias de classes e a implementação de polimorfismo.

A herança permite que as classes derivadas reutilizem atributos e métodos da classe base, evitando a duplicação de código. Caso tenha várias classes com comportamentos comuns, é possível colocá-lo na classe base e, em seguida, fazer com que as classes derivadas herdem essas funcionalidades.

No código

Na linha 4 da classe livro temos um exemplo de herança, onde a classe livro herda da classe ItemBiblioteca. Confira.

```
13 referências
public class Livro : ItemBiblioteca
{
```

Na linha 13 da classe livro, temos o método construtor da classe derivada que chama os parâmetros do método construtor da classe base, utilizando o comando, “:base(titulo, codigo)”. Confira.

```
13 | public Livro(string titulo, string codigo, string autor, string isbn,
    | string genero, int estoque) : base(titulo, codigo)
14 | {
15 |
16 |     Autor = autor;
17 |     ISBN = isbn;
18 |     Genero = genero;
19 |     Estoque = estoque;
20 |
21 | }
```

Polimorfismo

O polimorfismo é um princípio essencial na programação orientada a objetos, possibilitando a manipulação de um objeto de diversas maneiras. Dito de outra forma, o polimorfismo possibilita que um objeto atue de formas distintas, dependendo do contexto em que é empregado.

Uma das maiores vantagens é a reutilização de código, já que é viável desenvolver classes genéricas que podem ser empregadas de forma adaptável em variados cenários.

No Código

Na classe IPesquisavel e Iemprestavel temos a implementação da interface que é um exemplo de polimorfismo, ela permite que métodos sejam sobrescritos ou implementados de forma diferente. Confira.

```
0 referências
2 | public interface IPesquisavel
3 | {
4 |     0 referências
    | void PesquisarLivro();
5 | }
6 |
7 | //interface pesquisar usuario, chama
0 referências
8 | public interface IPesquiUsuario
9 | {
10 |     0 referências
    | void PesquisarUsuario();
11 | }
12 |
13 | //interface pesquisar por autor, chama
0 referências
14 | public interface IPesquisaAutor
15 | {
16 |     0 referências
    | void PesquisarAutor();
17 | }
```

```
0 referências
2 | public interface IEmprestavel
3 | {
4 |     0 referências
    | void Emprestar();
5 | }
6 |
7 | //interface devolver, chamando o I
0 referências
8 | public interface IDevolver
9 | {
10 |     0 referências
    | void Devolver();
11 | }
```

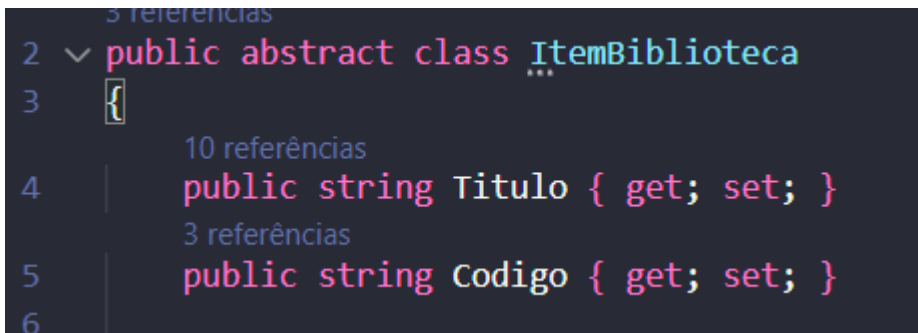
Abstração

A Abstração se refere ao ato de esconder os detalhes de implementação de um objeto, revelando apenas as funcionalidades vitais e pertinentes. A abstração possibilita que os programadores interajam com um objeto em um nível superior, sem se preocupar com os detalhes de como esses objetos são implementados no interior.

A abstração reduz a complexidade e torna o projeto e a implementação mais eficientes.

No Código

Na classe ItemBiblioteca temos métodos abstratos que estão empregados em ItemBiblioteca, no método Emprestar e no método Devolver. Confira.

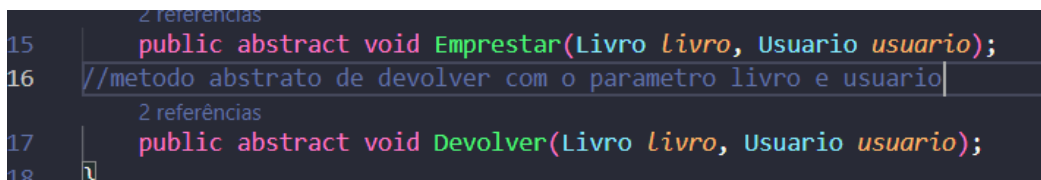


```

2  public abstract class ItemBiblioteca
3  {
4      public string Titulo { get; set; }
5      public string Codigo { get; set; }
6  }

```

Linha 15 e 16.



```

15  public abstract void Emprestar(Livro livro, Usuario usuario);
16  //metodo abstrato de devolver com o parametro livro e usuario
17  public abstract void Devolver(Livro livro, Usuario usuario);
18  }

```

O projeto desenvolvido em C# foi uma ótima oportunidade para aplicar e ampliar o entendimento sobre os fundamentos da Programação Orientada a Objetos (POO), como encapsulamento, herança, polimorfismo e abstração. Cada conceito foi implementado de maneira prática, mostrando a relevância desses princípios para a criação de sistemas sólidos, flexíveis e de fácil manutenção.

Por meio deste projeto foi possível entender como esses conceitos da POO são importantes para resolver problemas do mundo real, permitindo criar sistemas mais eficientes e fáceis de modificar quando necessário.