UNIVERSIDADE LUTERANA DO BRASIL

CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

CAMPUS TORRES

PAULO HENRIQUE DOS SANTOS

Projeto Orientado a Objetos em C#

TORRES

2024

UNIVERSIDADE LUTERANA DO BRASIL

CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

CAMPUS TORRES

PAULO HENRIQUE DOS SANTOS

Avaliação Parcial 2: Projeto Orientado a Objetos em C# apresentado ao Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Luterana do Brasil, abordando o desenvolvimento de um sistema de gerenciamento de pacotes turísticos, utilizando os princípios da Programação Orientada a Objetos.

Prof.º Lucas Rodrigues Schwartzhaupt Fogaça

TORRES

2024

1. INTRODUÇÃO

Neste trabalho, vamos explorar uma Avaliação Parcial 2 do nosso Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas na Universidade Luterana do Brasil. O projeto teve como objetivo desenvolver um sistema de gerenciamento de pacotes turísticos usando C#. O que torna esse projeto tão interessante é a chance de aplicar conceitos fundamentais da Programação Orientada a Objetos (POO) de uma maneira prática e dinâmica. A POO é uma abordagem que não só nos ajuda a organizar o código, mas também a torná-lo mais flexível e fácil de manter.

2. DESENVOLVIMENTO

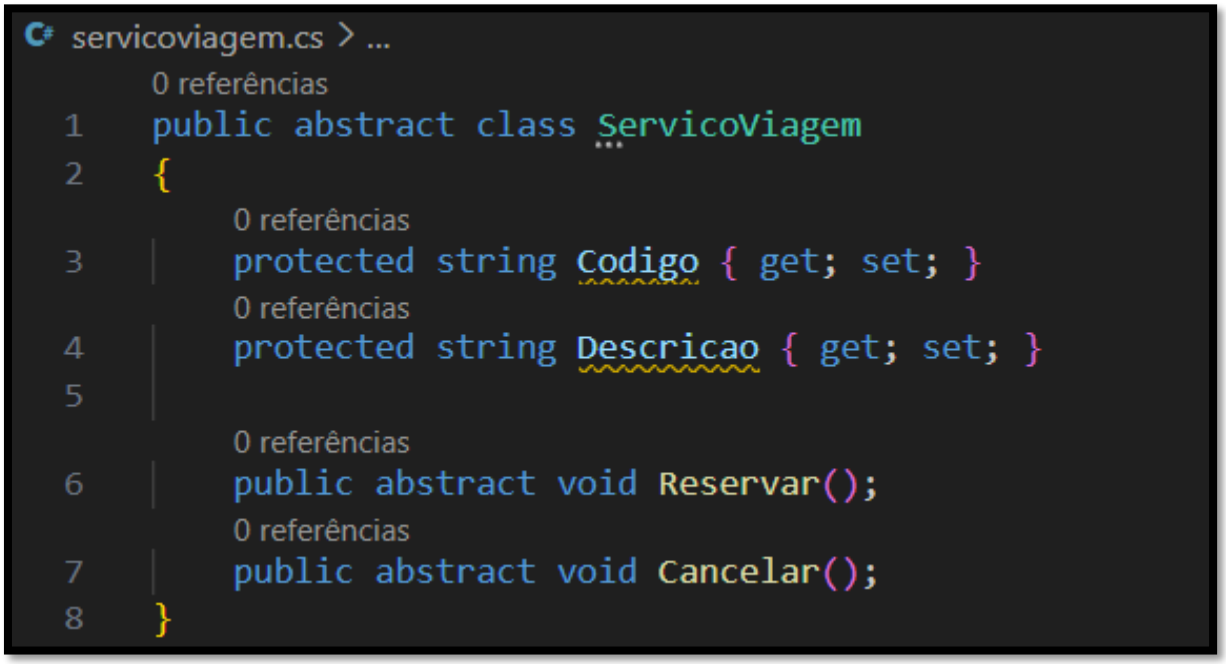
Durante o desenvolvimento do sistema, tivemos a oportunidade de mergulhar nos quatro pilares da POO: encapsulamento, herança, polimorfismo e abstração. Cada um deles desempenha um papel crucial na construção de um software robusto e intuitivo.

1. **Encapsulamento**: Esse conceito nos permite esconder os detalhes internos de uma classe, expondo apenas o que é realmente necessário. Para o nosso sistema, criamos uma classe Cliente que encapsula as informações do cliente, como Nome, NumeroIdentificacao e Contato. Dessa forma, garantimos que os dados sejam manipulados de maneira segura, sem expor diretamente suas informações. O trecho de código abaixo ilustra bem isso:

```
Cliente.cs > ...
      5 referências
1   public class Cliente {
          2 referências
2         public string Nome { get; set; }
          2 referências
3         public string NumeroIdentificacao { get; set; }
          2 referências
4         public string Contato { get; set; }
5
          1 referência
6         public void ExibirInformacoes()
7         {
8             Console.WriteLine("Nome: " + Nome);
9             Console.WriteLine("ID: " + NumeroIdentificacao);
10            Console.WriteLine("Contato: " + Contato);
11        }
12  }
```

Com essa estrutura, podemos adicionar métodos que validam e manipulam esses dados, garantindo que tudo ocorra de forma controlada.

2. **Herança:** A herança nos permite criar uma nova classe com base em uma já existente, o que facilita a reutilização de código. No nosso projeto, implementamos uma classe ServicoViagem que serve como base para classes como PacoteTuristico. Essa abordagem reduz a redundância e torna a manutenção do código mais simples. Funcionamento:



Ao definir métodos abstratos, as subclasses são obrigadas a implementar suas próprias versões, o que promove uma estrutura mais organizada.

3. **Polimorfismo:** Este pilar nos permite utilizar métodos de forma intercambiável, dependendo do contexto. Por exemplo, tanto a classe PacoteTuristico quanto outras que implementam a interface IReservavel podem ter suas próprias implementações dos métodos Reservar() e Cancelar(). Isso traz uma flexibilidade incrível ao sistema. Aqui está um exemplo do código

```
public class PacoteTuristico : IReservavel
{
    2 referências
    public string Codigo { get; set; }
    2 referências
    public string Descricao { get; set; }
    5 referências
    public int VagasDisponiveis { get; set; }

    2 referências
    public void Reservar()
    {
        if (VagasDisponiveis > 0)
        {
            VagasDisponiveis--;
            Console.WriteLine("Reserva realizada com sucesso.");
        }
        else
        {
            Console.WriteLine("Não há vagas disponíveis.");
        }
    }

    2 referências
    public void Cancelar()
    {
        VagasDisponiveis++;
        Console.WriteLine("Reserva cancelada com sucesso.");
    }
}
```

Com essa estrutura, a implementação dos métodos pode variar de acordo com o tipo de serviço, permitindo uma experiência mais personalizada para o usuário

4. **Abstração:** Por fim, a abstração nos permite focar nos aspectos mais importantes de um objeto, sem nos preocuparmos com detalhes desnecessários. Ao trabalharmos com a classe Destino, por exemplo, apenas as informações essenciais são expostas. Isso facilita o uso da classe sem sobrecarregar o usuário com dados excessivos:

```
1  public class Destino : IPesquisavel
2  {
       3 referências
3      public string NomeLocal { get; set; }
       2 referências
4      public string Pais { get; set; }
       3 referências
5      public string Codigo { get; set; }
       2 referências
6      public string Descricao { get; set; }
7
       3 referências
8      public void ExibirInformacoes()
9      {
10         Console.WriteLine($"Destino: {NomeLocal}, País: {Pais}, Código: {Codigo}, Descrição: {Descricao}");
11     }
```

Dessa forma, a interação com a classe se torna mais intuitiva e simples, permitindo que os usuários se concentrem no que realmente importa

3. **CONCLUSÃO**

Ao longo deste projeto, percebi como a Programação Orientada a Objetos não é apenas uma maneira de organizar o código, mas também uma filosofia que traz clareza e eficiência ao desenvolvimento de software. Cada um dos quatro pilares — encapsulamento, herança, polimorfismo e abstração — contribui de maneira significativa para a criação de sistemas mais organizados e fáceis de manter. Aprendi que aplicar esses conceitos na prática não só melhora a qualidade do código, mas também facilita o trabalho em equipe e a evolução do projeto. Estou animado para levar esses aprendizados para futuros desafios e continuar aprimorando minhas habilidades em programação.

4. REFERÊNCIAS

SELEM, Cristiane. Minutos de Desenvolvimento de Sistemas: Abstração e Encapsulamento na Programação Orientada a Objetos. [S. l.], 8 ago. 2023. Disponível em: <https://www.estrategiaconcursos.com.br/blog/abstracao-encapsulamento-programacao-orientada-objetos/>. Acesso em: 24 out. 2024.

OTTIMA, Admin. Polimorfismo, Encapsulamento, Abstração de dados e Herança em Programação Orientada a objetos. [S. l.], 21 nov. 2020. Disponível em: <https://ottima-power.com/pt/polimorfismo-encapsulamento-abstra%C3%A7%C3%A3o-de-dados-e-heran%C3%A7a-em-programa%C3%A7%C3%A3o-orientada-a-objetos/>. Acesso em: 23 out. 2024.

DEVMEDIA, Henrique. Os 4 pilares da Programação Orientada a Objetos. [S. l.], 2014. Disponível em: <https://www.devmedia.com.br/os-4-pilares-da-programacao-orientada-a-objetos/9264>. Acesso em: 24 out. 2024.

SOUZA, Priscila. Os 4 pilares da Programação Orientada a Objetos. [S. l.], 15 mar. 2023. Disponível em: <https://www.dio.me/articles/os-4-pilares-da-programacao-orientada-a-objetos-Y0CN7G>. Acesso em: 24 out. 2024.