UNIVERSIDADE LUTERANA DO BRASIL

CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

CAMPUS TORRES

Guilherme Birlem Machado

Introdução

Este projeto visa desenvolver um sistema de gerenciamento para uma Agência de Viagens, utilizando os quatro pilares da Programação Orientada a Objetos (POO): encapsulamento, herança, polimorfismo e abstração. A POO é uma abordagem de programação que se baseia na criação de objetos para modelar conceitos do mundo real. Isso facilita o desenvolvimento de sistemas complexos, tornando o código mais organizado, reutilizável e fácil de manter. Com a crescente demanda por soluções mais estruturadas, a POO se torna uma ferramenta indispensável para o desenvolvimento de software robusto e escalável. Neste texto, será detalhado como cada pilar foi aplicado no projeto da Agência de Viagens, demonstrando sua importância na construção do sistema.

Encapsulamento

O encapsulamento é um dos pilares fundamentais da POO e refere-se à prática de ocultar detalhes internos de uma classe, expondo apenas o que é necessário para o uso externo. Essa prática melhora a segurança do código e evita o acesso indevido aos atributos e métodos, mantendo o controle sobre como os dados são manipulados.

No projeto da Agência de Viagens, o encapsulamento foi aplicado nas classes cliente, destino e PacoteTuristico, que possuem atributos como nome, pais, reco, entre outros. Esses atributos foram declarados como públicos, mas poderiam ser privados ou protegidos, de acordo com a necessidade de controle no sistema.

Exemplo de aplicação de encapsulamento no código:

```
public class Cliente
{
    public string Nome { get; set; }
    public string NumeroIdentificacao { get; set; }
    public string Contato { get; set; }
}
```

Neste trecho, o encapsulamento é implementado através das propriedades públicas, que permitem a leitura e escrita dos atributos de forma controlada.

Herança

A herança é o mecanismo que permite que uma classe herde características de outra, promovendo a reutilização de código e a criação de uma estrutura hierárquica entre as classes. Com isso, classes mais específicas podem herdar comportamentos e atributos de classes mais gerais, adicionando ou sobrescrevendo funcionalidades conforme necessário.

No projeto, a classe ServicoViagem foi definida como uma classe base abstrata, da qual outras classes derivam, como PacoteTuristico. Essa abordagem permite que as subclasses herdem as propriedades e métodos da classe base, como Reservar() e Cancelar(), promovendo a reutilização de código.

Exemplo de herança no código:

```
public abstract class ServicoViagem
{
    public string Codigo { get; set; }
    public string Descricao { get; set; }

    public abstract void Reservar();
    public abstract void Cancelar();
}

public class PacoteTuristico : ServicoViagem
{
    public Destino Destino { get; set; }
    public string Datas { get; set; }
    public decimal Preco { get; set; }
    public int VagasDisponiveis { get; set; }

    public override void Reservar()
```

```
    {
        if (VagasDisponiveis > 0)
        {
            VagasDisponiveis--;
            Console.WriteLine($"Reserva confirmada para o destino: {Destino.NomeLocal}");
        }
    }

    public override void Cancelar()
    {
        VagasDisponiveis++;
        Console.WriteLine($"Reserva cancelada para o destino: {Destino.NomeLocal}");
    }
}
```

Neste exemplo, a classe PacoteTuristico herda da classe ServicoViagem, reutilizando seus atributos e métodos.

Polimorfismo

O polimorfismo permite que um objeto possa ser tratado como um objeto de sua classe base ou de uma interface que ele implementa. Isso proporciona flexibilidade no código, permitindo que métodos sobrescritos tenham diferentes implementações, dependendo da classe específica que os invoca.

No projeto da Agência de Viagens, o polimorfismo é aplicado quando o método Reservar() é sobrescrito na classe PacoteTuristico. Mesmo que o método esteja presente na classe base ServicoViagem, ele é implementado de forma específica na classe derivada, possibilitando diferentes comportamentos para diferentes tipos de pacotes de viagem.

Exemplo de polimorfismo no código:

```
public override void Reservar()
{
    if (VagasDisponiveis > 0)
    {
        VagasDisponiveis--;
        Console.WriteLine($"Reserva confirmada para o destino: {Destino.NomeLocal}");
    }
}
```

Aqui, o método Reservar() é implementado de maneira específica na classe PacoteTuristico, e, caso fosse aplicada em outra classe derivada de ServicoViagem, poderia ser implementada de forma diferente.

Abstração

A abstração envolve a simplificação de conceitos complexos através da criação de classes e métodos que capturam apenas os detalhes essenciais para o funcionamento do sistema. Isso permite que o desenvolvedor se concentre em aspectos importantes da lógica do sistema, sem se preocupar com a implementação interna de cada funcionalidade.

No projeto, a abstração foi aplicada com a criação da classe abstrata ServicoViagem, que define a estrutura básica para os serviços oferecidos pela agência, como a reserva e o cancelamento de pacotes. Essa classe serve como um modelo para outros tipos de serviços que possam ser adicionados ao sistema no futuro.

Exemplo de abstração no código:

```
public abstract class ServicoViagem
{
    public string Codigo { get; set; }
    public string Descricao { get; set; }

    public abstract void Reservar();
    public abstract void Cancelar();
}
```

Neste exemplo, a classe ServicoViagem define a estrutura básica para serviços de viagem, sem se preocupar com os detalhes da implementação, que são tratados pelas classes derivadas.

Conclusão

O desenvolvimento deste projeto foi uma experiência enriquecedora, que permitiu a aplicação prática dos quatro pilares da programação orientada a objetos. A utilização de encapsulamento, herança, polimorfismo e abstração não apenas facilitou a construção do sistema de gerenciamento de viagens, como também proporcionou um código mais organizado, reutilizável e fácil de manter. A POO mostrou-se essencial para criar soluções escaláveis e flexíveis, contribuindo significativamente para o aprendizado de boas práticas no desenvolvimento de software.

Referências

- LARMAN, Craig. "Utilizando UML e Padrões." 3ª ed. Bookman, 2007.
- DEITEL, Paul; DEITEL, Harvey. *C#: Como Programar*. 6ª ed. São Paulo: Pearson, 2017.
- GAMMA, Erich et al. "Padrões de Projeto: Soluções Reutilizáveis de Software Orientado a Objetos." Bookman, 1994.