Universidade Luterana do Brasil

Braian Lisandro Siebel

Agencia Viagens OOP
Programação orientada a objetos

Terra de Areia
2024

Introdução ao projeto:

O projeto foi desenvolvido utilizando a linguagem de programação C#, uma linguagem de programação orientada a objetos, o projeto consiste no desenvolvimento de um sistema de gerenciamento para uma agência de viagens.

O projeto conta com inúmeros requisitos, que devem ser implementados de formas variadas utilizando a linguagem, visando o uso dos pilares da programação orientada a objetos.

Importância da programação orientada a objetos:

A importância da programação orientada a objetos (POO), se resume em sua estrutura organizada que permite uma estrutura de código intuitiva, de fácil compreensão. Seus pilares promovem a organização, que facilita que outras pessoas entendam facilmente o código, e que projetos complexos tenham uma execução facilitada.

Os quatro pilares da programação orientada a objetos:

Nesta parte, será explicado como funcionam esses pilares, e como os mesmos foram implementados no projeto.

Encapsulamento:

Visa a proteção de dados numa classe, por meio dos modificadores de acesso, assim podendo garantir um melhor controle sobre quais informações são acessíveis ou não, fazendo com que somente classes com o devido acesso podem estar acessando a informação.

Esse “devido acesso” majoritariamente é difundido de duas formas:

1 - Por meio do modificador de acesso em si, se um método por exemplo, tiver o modificador de acesso *public*, pode ser acessado fora da classe que foi instanciado. Mas caso o modificador de acesso estiver como *private*.

2- Através de getters e setters.

No projeto é possível observar o uso claro de *getters* e *setters* na classe “Reserva.cs”.

```
    public PacoteTuristico Pacote { get; set; }
    public Cliente Cliente { get; set; }
    public bool Status { get; set; }
    private string codigo;

    public string Codigo
```

```
  {
    get { return codigo; }
    set { codigo = value; }
  }
```

Onde a propriedade “codigo” estava registrada como private, então foi necessário estar colocando um getter e setter para conseguir estar acessando essa propriedade fora da classe.

Herança:

Esse pilar consiste na propriedade de herdar características de um ponto A num ponto B, de forma aplicável na linguagem C#, consiste na criação de classes em cima de classes que já existem, seria como se fosse uma árvore genealógica, onde o antecessor passa algumas características para seu sucessor, e esse sucessor herda essas características, e incrementa as suas próprias propriedades, compondo os aspectos ao todo desse sucessor.

No projeto é possível verificar isso nas primeiras linhas da classe “ProjetoTuristico”, onde a mesma herda da classe “ServicoViagem”, da interface “IReservavel”, e também da interface IPesquisavel.

```
public class PacoteTuristico : ServicoViagem, IReservavel, IPesquisavel
{
    public Destino Destino;
    public DateTime Datas;
```

A função de herança no C# não suporta que mais de uma classe seja herdada ao mesmo tempo, porém isso pode ser simulado utilizando interfaces como no exemplo acima.

Na sintaxe, para estar herdando é necessário utilizar dois pontos “:”, e caso queira utilizar interfaces junto a uma classe que está herdando outra classe, é necessário o uso de vírgulas.

Para concluir esse pilar, a herança é o pilar que consiste em herdar características de um ponto A ao B, e no uso prático na linguagem C# é dado por meio de heranças utilizando os dois pontos “:” com classes, e vírgulas “,”, com interfaces.

Concluindo a ideia do pilar, o mesmo implica que características podem ser reutilizadas não precisando reescrever as mesmas novamente.

Polimorfismo:

O pilar polimorfismo promove que um mesmo método possa ser usado para executar funções diferentes. Uma analogia prática disso com algo da vida real seria o vinagre, que pode ser utilizado para condimentar alimentos, limpar superfícies e até mesmo remover ferrugem. Nesse caso na linguagem de programação C#, o método vinagre iria sofrer uma sobrecarga de métodos, dado os devidos parâmetros e funções, o mesmo pode estar atuando tanto como removedor de ferrugem, condimento e produto de limpeza.

No projeto é possível verificar o polimorfismo com o método “PesquisarPorCodigo” (linha 3), na interface “IPesquisavel”.

```
public interface IPesquisavel
```

```
{
    void PesquisarPorCodigo(string codigo);
```

O mesmo método é utilizado na classe “Destino” para retornar um destino. (linha 40 até 47 da classe “Destino.cs”)

```
public void PesquisarPorCodigo(string codigo)
    {
        if (this.Codigo == codigo)
        {
            System.Console.WriteLine("Destino encontrado");
        }
        else
        {
```

E também é utilizado na classe PacoteTuristico para retornar um pacote (linha 55 até 65 da classe “PacoteTuristico.cs”)

```
public void PesquisarPorCodigo(string codigo)
    {
        if (this.Codigo == codigo)
        {
            System.Console.WriteLine("Pacote encontrado");
        }
        else
        {
            System.Console.WriteLine("Pacote não encontrado");
        }
    }
```

O mesmo método foi utilizado para fazer duas coisas diferentes, caracterizando assim um polimorfismo.

Para concluir a explicação, dentro do C#, o pilar consiste em executar funções diferentes utilizando o mesmo método, que no código foi utilizado o método “PesquisaPorCodigo” da interface “IPesquisavel”.

Abstração:

O pilar abstração consiste na simplificação, seja de um objeto ou sistema, é dado apenas o que é necessário sem detalhes complexos, focando unicamente no que esse sistema ou objeto faz, do que como que a função de objeto é realizada, sem detalhes desnecessários visando o funcionamento.
Exemplo disso seria ter um método chamado MandarMensagem(), e outro ReceberMensagem(), com foco apenas nesses métodos, sem precisar se atentar a como as redes funcionam, estudar a fundo o modelo OSI, TCP e UDP, etc.

Exemplo disso no código é a classe ServicoViagem.

```
public abstract class ServicoViagem
```

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


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


}
```

Ela em si já é uma classe abstrata, e conta com dois métodos abstratos, sendo aqueles que herdam esses métodos, terão que dar as implementações que quiserem.

Concluindo o pilar, a abstração permite que processos complexos sejam simplificados focando apenas na função do processo, e não como ele funciona.

Conclusão:

Resumindo o trabalho num todo: O mesmo colaborou muito para o entendimento de como uma linguagem orientada a objetos funciona, principalmente por ser um projeto extenso, colocando a tona todos os principais conceitos de forma constante, e ter que explicar como o código funciona para reforçar e demonstrar domínio sobre o que foi feito, somado também ao fato de ter que após fazer o trabalho, explicar toda a parte teórica do que foi feito, promove o entendimento do paradigma da programação orientada a objetos.

Referências:

COMPUTER WEEKLY. **Abstraktion**. Disponível em: https://www.computerweekly.com/de/definition/Abstraktion.
STACKIFY. **OOP Concepts for Beginners: What is Polymorphism**. Disponível em: https://stackify.com/oop-concept-polymorphism/.
GRAN CURSOS ONLINE. **Herança na Programação Orientada a Objetos**. Disponível em: https://blog.grancursosonline.com.br/heranca-na-programacao-orientada-a-objetos/.
SUMO LOGIC. **Encapsulation - Definition & Overview**. Disponível em: https://www.sumologic.com/glossary/encapsulation/