UNIVERSIDADE LUTERANA DO BRASIL

Lucas Martins Tarachuck

PILARES DA PROGRAMAÇÃO ORIENTADA A OBJETOS

Torres
2024

1. **INTRODUÇÃO**

Dentro da Programação Orientada a Objetos existem alguns pilares que são fundamentais, sendo eles: Herança, Encapsulamento, Polimorfismo e Abstração. Este trabalho será desenvolvido em base de apresentá-los, explicar suas funções e onde são utilizados, também serão mostrados exemplos de cada um dentro do código onde será aprofundada a utilização dos mesmo dentro do contexto do projeto realizado.

2. **DESENVOLVIMENTO**

2.1 **HERANÇA**

A herança é um recurso da programação orientada a objetos que permite a criação de uma classe base, assim outras classes podem herdar métodos, variáveis e propriedades declaradas na classe base. Uma classe passa a ser utilizada como base, quando outra classe a recebe, por exemplo: “public class PacoteTuristico : ServicoViagem” na primeira linha da classe PacoteTuristico, os dois-pontos servem para indicar qual classe se tornará a base sendo essa “ServicoViagem” que está a sua frente. No momento que uma classe herda os métodos e propriedades de outra classe, ela é obrigado a possuir os mesmo os mesmos atributos que foram designados dentro do método construtor de sua base, exemplificando, a classe PacoteTuristico possui como propriedades: Destino, DataInicio, DataFim, Preco e VagasDisponíveis (terceira, quarta, quinta, sexta e sétima linha de PacoteTuristico respectivamente); no momento em que serão passados os parâmetros ao criar o método construtor “public PacoteTuristico(string destino, DateTime datainicio, DateTime datafim, decimal preco, int vagasdisponiveis)”(nona linha de PAcoteTuristico), as propriedades da classe base também terão que passar por parâmetro, assim ficando como “public PacoteTuristico(string destino, DateTime datainicio, DateTime datafim, decimal preco, int vagasdisponiveis, string codigo, string descricao)”(nona linha de PacoteTuristico), também tendo que ser acrescentada a palavra reservada “base” mais as propriedades que estão sendo referenciadas da classe base, logo antes de definir cada propriedade dentro do método, o método fica “public PacoteTuristico(string destino, DateTime datainicio, DateTime datafim, decimal preco, int vagasdisponiveis, string codigo, string descricao):base(codigo, descricao)”(nona linha de PacoteTuristico), sendo “codigo” e “descricao” as propriedades da classe base, porém elas não precisam ser definidas dentro do método construtor pois já estão definidas no método da classe base. No caso de ser uma classe abstrata como base, além de todas as propriedades, todos os métodos também deverão ser herdados, caso o método seja abstrato, ele é herdado da seguinte maneira “public override void Reservar()”(décima oitava linha de PacoteTuristico) sendo utilizado o “override” como se fosse sobrescrever um método, depois declarando o tipo de retorno e o nome do método que deverá ser idêntico ao método abstrato declarado na classe base.

2.2 **ENCAPSULAMENTO**

O encapsulamento é um meio de ocultar um método, uma propriedade, uma variável ou lista de uma classe específica, para assim o “proteger” de acessos indevidos a tais itens da classe e são feitos através de modificadores de

acesso, existem o total de cinco deles: public, protected, private, internal e protected internal.

2.2.1 PUBLIC

O modificador de acesso public, permite que o método, propriedade, variável ou lista seja público, permitindo o acesso de outras classe e do “Program.cs”. um item se torna público no momento que há um public declarado antes de tudo, sejam métodos, listas ou propriedades. Exemplo:

Public string Destino { get; set; }(terceira linha PacoteTuristico), nesse caso, está declarando a propriedade “Destino” como pública.

public void ExibirInformacoes()(décima sétima linha de Destinos), aqui está declarando o método ExibirInformacoes como público.

2.2.2 PROTECTED

O protected permite o acesso de itens da classe apenas pela própria classe ou por classes que herdam da classe base, se declara um item protegido com um “protected” antes de declarar o tipo de retorno de um método, todo tipo de uma proprieda ou de uma lista. Exemplos:

protected string Codigo { get; set; }(terceira linha de ServicoViagem), onde a propriedade código está sendo protegida, assim só tendo a classe ServicoViagem e classes derivadas da mesma podendo acessá-lo.

2.2.3 PRIVATE

O modificador de acesso private é utilizado para permitir acesso dos membros da classe apenas pela própria classe, sendo identificado um “private” antes de declarar algum tipo ou lista.

2.2.4 INTERNAL

O internal é modificador interno padrão quando não é especificado nenhum outro tipo de modificador de acesso.

2.2.5 PROTECTED INTERNAL

O protected internal é como um protected é um modificador de acesso que torna um membro da classe acessível apenas para a própria classe e suas classes derivadas.

2.3 POLIMORFISMO

O polimorfismo tem duas funcionalidades, uma que é permitir que objetos de uma classe derivada sejam tratados como objetos da classe base podendo ser utilizados como parâmetros de métodos, o outro caso, entra no caso de sobrescrever métodos de classe base, assim podendo os alterar dentro da

classe derivada, e esses métodos são chamados de métodos virtuais. Um exemplo utilizado no código é “public override void Cancelar()”(quadragésima quarta linha de PacoteTuristico), onde é sobrescrito o método Cancelar() da classe abstrata ServicoViagem, um método virtual tem sua sintaxe com um “override” após o modificador de acesso e antes do tipo de retorno.

2.4 **ABSTRAÇÃO**

A abstração se dá a partir de uma classe abstrata que por conta própria não pode ser alterada sozinho e nem se pode criar um objeto a partir dela, entretanto ela pode ser herdada e se tornar uma classe base para outra classe. Uma classe abstrata pode conter propriedades e métodos abstratos ou não. Os métodos abstratos são métodos que não possuem nenhuma implementação, por exemplo “public abstract void Reservar();”(décima segunda linha de ServicoViagem) que se tornou uma método abstrato ao acrescentar a palavra reservada “abstract” após o modificador de acesso, e eles são apenas métodos virtuais, só podendo ser implementados serem sobrescritos em uma classe derivada de classe abstrata.

3 **CONCLUSÃO**

Esse trabalho foi extremamente importante para ampliação de conhecimento e aprendizado especificamente dadas as pesquisas feitas sobre as funcionalidades de cada pilar da programação orientada a objetos especificamente em C# no caso deste, principalmente na parte do encapsulamento com os modificadores “Internal” que é por “default” quando nenhum modificador de acesso é declarado e o “Protected Internal” que é basicamente equivalente ao “Protected”.

4 REFERÊNCIAS BIBLIOGRÁFICAS

https://learn.microsoft.com/pt-br/dotnet/csharp/fundamentals/tutorials/inheritance

https://www.macoratti.net/18/01/c_encaps1.htm

https://learn.microsoft.com/pt-br/dotnet/csharp/fundamentals/object-oriented/polymorphism

https://pt.stackoverflow.com/questions/45386/qual-a-utilidade-de-m%C3%A9todos-com-modificador-internal

https://learn.microsoft.com/pt-br/dotnet/csharp/language-reference/keywords/protected-internal

https://www.macoratti.net/12/06/c_caip1.htm