AP2 - **LABORATÓRIO DE PROGRAMAÇÃO.**

NOME: Amanda Justin de Lima.

OS QUATRO PILARES DA PROGRAMAÇÃO ORIENTADA A OBJETOS.

INTRODUÇÃO:

A **Programação Orientada a Objetos** (conhecida, também, pela sigla POO) é um paradigma de linguagem de programação e, portanto, é um dos caminhos para conseguir resolver um problema proposto. Ela auxilia a definir a forma como soluções e sistemas serão estruturados, além de permitir uma maior **aproximação** entre real e virtual. A **programação orientada a objetos** surgiu como uma alternativa a essas características da programação estruturada. Esse novo paradigma se baseia principalmente em dois conceitos chave: **classes** e **objetos**. Todos os outros conceitos, igualmente importantes, são construídos em cima desses dois.Dentro deste conceito, o design de uma solução é pensado em termos de **entidades e objetos** e, também, na relação entre eles:

- **entidades:** dentro da Programação Orientada a Objetos, as entidades são representações digitais de entes reais, ou seja, estão ligadas com a modelagem do mundo real para o ambiente do código;
- **objetos:** são um tipo de entidade que pode reter uma informação e oferece uma série de dados (comportamento). São eles que permitem o processamento das informações ocorrerem em sistemas.

Para entendermos exatamente do que se trata a orientação a objetos, vamos entender quais são os requerimentos de uma linguagem para ser considerada nesse paradigma:

Abstração:

Na orientação a objetos, o conceito de abstração ou abstrair, significa esconder os detalhes de uma implementação, ou seja, quanto menos souberem sobre nossas classes, mais fácil de consumí-las será. Permite aos desenvolvedores concentrar-se no que um objeto faz, em vez de como ele faz.

Conceito de abstração se desdobra em:

- Classes:
  - São conjuntos de objetos que possuem o mesmo tipo;
- Objetos:
  - São instâncias das classes.

Exemplo no código:

```
public abstract class ServicoViagem
{
    // Propriedades da classe abstrata ServicoViagem
    5 referências
    public string Codigo {get; set; } // Código identificador do
    1 referência
    public string Descricao { get; set; } // Descrição do serviç

    // Construtor da classe ServicoViagem que inicializa as prop
    1 referência
    public ServicoViagem(string codigo, string descricao)
    {
        Codigo = codigo;
        Descricao = descricao;
    }

    // Métodos abstratos que devem ser implementados pelas class
    2 referências
    public abstract void Reservar(); // Método para reservar o s
    2 referências
    public abstract void Cancelar(); // Método para cancelar a r
}
```

Encapsulamento:

O encapsulamento protege os dados de um objeto contra acesso direto de fora da classe, expondo apenas métodos seguros para operação sobre esses dados. Usando modificadores de acesso, como “privado”, “público” e “protegido”, que definem a visibilidade desses membros da classe.

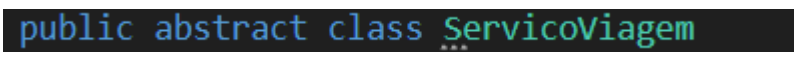
Exemplo do código:

O encapsulamento fortalece a segurança do código e evita que o estado do objeto seja modificado inadvertidamente.

Herança:

A herança é um mecanismo pelo qual uma nova classe, chamada classe derivada ou subclasse, pode adquirir as propriedades de uma classe existente, conhecida como classe base ou superclasse. Esse pilar permite a reutilização de código, a extensibilidade e a organização hierárquica de classes.

Exemplo do código:



Classe abstrata ServicoViagem

```
public class PacoteTuristico : ServicoViagem
```

Classe PacoteTuristico que herda de ServicoViagem

Polimorfismo:

O polimorfismo, que significa “muitas formas”, é a capacidade de um método para fazer coisas diferentes com base no objeto que o invoca. É propriedade de duas ou mais classes derivadas de uma mesma superclasse responderem a mesma mensagem, mas cada subclasse de uma forma diferente da outra classe derivada.

```
public override void Reservar()
{
    if (VagasDisponiveis > 0)
    {
        VagasDisponiveis--; // Reduz o número de vagas disponíveis
        Console.WriteLine($"Vagas diponíveis: {VagasDisponiveis}");
    }
    else
    {
        Console.WriteLine("Não há vagas disponíveis");
    }
}
```

```
public override void Cancelar()
{
    VagasDisponiveis++;
    Console.WriteLine($"Reserva cancelada! Vagas disponíveis: {VagasDisponiveis}");
}

// Implementação do método para pesquisar o pacote por código
// Retorna o pacote se o código coincidir
0 referências
public object? PesquisarPorCodigo(string codigo)
{
    return Codigo == codigo ? this : null;
}

// Implementação do método para pesquisar o pacote por nome do destino
// Retorna o pacote se o nome do destino coincidir
0 referências
public object? PesquisarPorNome(string nome)
{
    return Destino.NomeLocal == nome ? this : null;
}
```

O método Reservar() e Cancelar() são polimorfismo.

Conclusão:

A abstração e o encapsulamento ajudam a esconder a complexidade e a proteger os dados, respectivamente, enquanto a herança e o polimorfismo promovem a reutilização de código e a flexibilidade. Juntos, esses pilares permitem aos desenvolvedores construir sistemas de software mais robustos, eficientes e fáceis de manter.

Referências bibliográficas utilizadas:

https://academiatech.blog.br/o-que-e-programacao-orientada-a-objetos/
https://blog.grancursosonline.com.br/pilares-da-poo/