INSTITUTO INFNET

LÍVIA FARIA BRAZ

TESTE DE PERFORMANCE 3

Disciplina Regular 2: Fundamentos de Desenvolvimento com C#

Professor: Luiz Paulo Maia

ITAÚNA - MG

17 de março de 2025

Fundamentos de Desenvolvimento com C# - TP3

➢ **Exercício 01:** Conceitos de Classe, Objeto, Campos e Métodos (C#)

A Programação Orientada a Objetos (POO) é um jeito de organizar o código pensando em objetos do mundo real. Aqui estão os conceitos básicos:

➔ **Classe:** É como um molde ou um plano para criar objetos. Define quais características (atributos) e comportamentos (métodos) os objetos vão ter.

```
class Pirata {
```

➔ **Objeto:** É a instância da classe, ou seja, a versão real baseada no molde. Se a classe for "Pirata", um objeto seria "Luffy" ou "Zoro".

➔ **Campos (ou atributos):** São as informações que o objeto guarda, como nome e recompensa no caso dos piratas.

```
public String Nome { get; set; }

public double Recompensa { get; set; }
```

➔ **Métodos:** São as ações que o objeto pode realizar. No nosso código, um pirata pode atacar ou aumentar sua recompensa.

```
public override string ToString(){

    return "--- Pirata ---\nNome: " + Nome + "\nRecompensa: " + Recompensa + "
berries\n";

}


public void Atacar() {

   Console.WriteLine(Nome + " partiu para o ataque!");

}


public void AumentarRecompensa(double valor) {

   Recompensa += valor;

    Console.WriteLine("Recompensa de " + Nome + " aumentou para " + Recompensa + "
berries.");

}
```

Criamos uma classe chamada Pirata, que define como um pirata de One Piece deve ser. Ele tem dois atributos:

nome: armazena o nome do pirata.

recompensa: guarda o valor da recompensa dele.

Temos dois construtores: um vazio (caso a gente queira criar o pirata e definir os valores depois) e outro que já recebe nome e recompensa logo na criação.

```
public Pirata() { }


public Pirata(string nome, double recompensa){
   Nome = nome;
   Recompensa = recompensa;
}
```

Os métodos são as ações que um pirata pode fazer:

atacar(): exibe uma mensagem dizendo que o pirata partiu para o ataque.
aumentarRecompensa(double valor): adiciona um valor à recompensa atual e mostra a nova quantia.
toString(): sobrescreve o toString() padrão do Java para que, quando tentarmos imprimir um objeto com System.out.println(), ele exiba os dados formatados do jeito que queremos.
Também tem os métodos get e set, que servem para acessar e modificar os atributos de um jeito controlado.

Agora, na Main, criamos dois piratas:

```
Pirata p1 = new Pirata();
p1.Nome = "Zoro";
p1.Recompensa = 32000;
Pirata p2 = new Pirata("Luffy", 100000);
```

Aqui, criamos p1 sem passar valores no construtor e depois usamos os métodos set para definir nome e recompensa. Já p2 (Luffy) é criado diretamente com nome e recompensa.

```
p2.Atacar();
p1.AumentarRecompensa(10);
Console.WriteLine(p1);
Console.WriteLine(p2);
```

Ele parte para o ataque e, em seguida, aumentamos a recompensa de p1 (Zoro) em 10 berries.

Por fim, usamos System.out.println(p1) e System.out.println(p2). Isso chama o método toString() da classe Pirata, que formata as informações para exibição. No retorno temos:

```
Luffy partiu para o ataque!
Recompensa de Zoro aumentou para 32010.0 berries.
---- Pirata ----
Nome: Zoro
Recompensa: 32010.0 berries

---- Pirata ----
Nome: Luffy
Recompensa: 100000.0 berries
```

➢ **Exercício 02:** Criando a Classe "Ingresso"

A classe Ingresso é essencial para representar qualquer bilhete que pode ser vendido ou gerenciado em um sistema de eventos. Os atributos principais são:

- NomeDoShow (string): guarda o nome do show ou evento para o qual o ingresso é válido.
- Preco (double): representa o valor do ingresso. Como os preços podem conter casas decimais (R$150,50, por exemplo), usamos o tipo double para armazená-los corretamente.
- QuantidadeDisponivel (int): Indica quantos ingressos ainda estão disponíveis para venda.

Ou seja, cada atributo tem seu papel para garantir que o sistema consiga gerenciar os produtos direitinho.

```
namespace exs02_03_04_05_06.Models;

public class Ingresso {

   public string NomeDoShow;

   public double Preco;

   public int QuantidadeDisponivel;

 }
```

➢ **Exercício 03:** Métodos Básicos da Classe "Ingresso"

```
public double AlterarPreco(double novoPreco) {

   Preco = novoPreco;

   return Preco;

}


public int AlterarQuantidade(int novaQuantidade) {

   QuantidadeDisponivel = novaQuantidade;

   return QuantidadeDisponivel;
```

```
}

public void ExibirInformacoes() {

   Console.WriteLine("---- Ingresso ----");

   Console.WriteLine("Nome do show: " + NomeDoShow);

   Console.WriteLine("Preço: " + Preco);

   Console.WriteLine("Quantidade disponível: " + QuantidadeDisponivel);

}
```

➢ **Exercício 04:** Testando a Classe "Ingresso"

```
using exs02_03_04_05_06.Models;

namespace exs02_03_04_05_06;

class Program {
   static void Main(string[] args) {
        Ingresso ingresso = new Ingresso();
        ingresso.NomeDoShow = "Show do Slipknot";
        ingresso.Preco = 200;
        ingresso.QuantidadeDisponivel = 1000;

        ingresso.ExibirInformacoes();

        ingresso.AlterarPreco(150);
        ingresso.AlterarQuantidade(500);

        ingresso.ExibirInformacoes();
   }
}
```

```
---- Ingresso ----
Nome do show: Show do Slipknot
Preço: 200
Quantidade disponível: 1000
---- Ingresso ----
Nome do show: Show do Slipknot
Preço: 150
Quantidade disponível: 500
```

➢ **Exercício 05:** Criando Métodos de Propriedade (Getters e Setters)

Os getters e setters trazem algumas vantagens:

- Deixam o código mais organizado e evitam acesso direto às variáveis, o que pode prevenir modificações indesejadas.
- Facilitam a validação de dados. Por exemplo, no setter de preco, poderíamos garantir que o preço nunca seja negativo.
- Melhoram a flexibilidade do código. Se um dia for necessário mudar a lógica interna de um atributo (como calcular o preço com desconto automaticamente), dá pra fazer isso dentro do getter sem impactar outras partes do código.

Ou seja, os getters e setters ajudam a manter o controle sobre os dados e evitam bagunça no código. E no C# temos uma forma mais fácil de colocar no código, apenas utilizando: `“{ get; set; }”.`

```
public string NomeDoShow { get; set; }
public double Preco { get; set; }
public int QuantidadeDisponivel { get; set; }


static void Main(string[] args) {
    Ingresso ingresso = new Ingresso();
    ingresso.NomeDoShow = "Slipknot - We Are Not Your Kind Tour";
    ingresso.Preco = 200;
    ingresso.QuantidadeDisponivel = 1000;

    ingresso.AlterarPreco(150);
    ingresso.AlterarQuantidade(500);

    ingresso.ExibirInformacoes();

    Console.WriteLine("Alterando utilizando o setter:");
    ingresso.NomeDoShow = "Metallica - WorldWired Tour";
    ingresso.Preco = 450.00;

    ingresso.ExibirInformacoes();
}
```

```
---- Ingresso ----
Nome do show: Slipknot - We Are Not Your Kind Tour
Preço: 150
Quantidade disponível: 500
Alterando utilizando o setter:
---- Ingresso ----
Nome do show: Metallica - WorldWired Tour
Preço: 450
Quantidade disponível: 500
```

➢ **Exercício 06:** Adicionando Construtores à Classe "Ingresso"

O construtor serve para inicializar o objeto no momento em que ele é criado. Normalmente como padrão, assim como no Java, o C# já cria um construtor vazio para usarmos, mas quando queremos iniciar a classe com os atributos, precisamos construí-lo, caso a gente precise deixar as duas opções (de setar depois ou na criação) podemos criar um vazio e outro com os atributos.

```
public Ingresso() {}

public Ingresso(string nomeDoShow, double preco, int
quantidadeDisponivel) {
    NomeDoShow = nomeDoShow;
    Preco = preco;
    QuantidadeDisponivel = quantidadeDisponivel;
}
```

```
Console.WriteLine("\nDeclarando com o construtor completo:");
Ingresso ingresso2 = new Ingresso("Iron Maiden - Legacy of the
Beast Tour", 300, 800);
ingresso2.ExibirInformacoes();

Console.WriteLine("Declarando com o construtor vazio:");
Ingresso ingresso = new Ingresso();
ingresso.NomeDoShow = "Slipknot - We Are Not Your Kind Tour";
ingresso.Preco = 200;
ingresso.QuantidadeDisponivel = 1000;

ingresso.ExibirInformacoes();
```

```
---- Ingresso ----
Nome do show: Slipknot - We Are Not Your Kind Tour
Preço: 200
Quantidade disponível: 1000
```

```
Declarando com o construtor:
---- Ingresso ----
Nome do show: Iron Maiden - Legacy of the Beast Tour
Preço: 300
Quantidade disponível: 800
```

➢ **Exercício 07:** Modelando uma Matrícula

```
public class Matricula {
    string NomeDoAluno { get; set; }
    string Curso { get; set; }
    int NumeroMatricula { get; set; }
    string Situacao { get; set; }
    string DataInicial { get; set; }

        public Matricula(string nomeDoAluno, string curso, int
numeroMatricula, string situacao, string dataInicial) {
        NomeDoAluno = nomeDoAluno;
        Curso = curso;
        NumeroMatricula = numeroMatricula;
        Situacao = situacao;
        DataInicial = dataInicial;
    }

}
```

➢ **Exercício 08:** Criando Métodos na Classe de Matrícula

```
public void Trancar() {
    if (Situacao == "Ativa") {
        Situacao = "Trancada";
        Console.WriteLine("Matrícula trancada com sucesso!");
    } else {
        Console.WriteLine("A matrícula já está trancada ou concluída.");
    }
}
public void Reativar() {
    if (Situacao == "Trancada") {
        Situacao = "Ativa";
        Console.WriteLine("Matrícula reativada com sucesso!");
    } else {
        Console.WriteLine("A matrícula precisa estar trancada para ser
reativada.");
    }
}
public void ExibirInformacoes() {
    Console.WriteLine("---- Matrícula ----");
    Console.WriteLine("Nome do aluno: " + NomeDoAluno);
    Console.WriteLine("Curso: " + Curso);
    Console.WriteLine("Número da matrícula: " + NumeroMatricula);
    Console.WriteLine("Situação: " + Situacao);
    Console.WriteLine("Data inicial: " + DataInicial);
}
```

➢ **Exercício 09:** Testando a Classe de Matrícula

```
using ex07_08_09.models;

namespace ex07_08_09;

class Program {
   static void Main(string[] args) {
         Matricula matricula = new Matricula("Zoro", "Swordsmanship",
123456, "Ativa", "01/01/2021");

       matricula.ExibirInformacoes();

       matricula.Trancar();
       matricula.Trancar();
       matricula.ExibirInformacoes();

       matricula.Reativar();
       matricula.Reativar();
       matricula.ExibirInformacoes();
   }
}
```

```
---- Matrícula ----
Nome do aluno: Zoro
Curso: Swordsmanship
Número da matrícula: 123456
Situação: Ativa
Data inicial: 01/01/2021
Matrícula trancada com sucesso!
A matrícula já está trancada ou concluída.
---- Matrícula ----
Nome do aluno: Zoro
Curso: Swordsmanship
Número da matrícula: 123456
Situação: Trancada
Data inicial: 01/01/2021
Matrícula reativada com sucesso!
A matrícula precisa estar trancada para ser reativada.
---- Matrícula ----
Nome do aluno: Zoro
Curso: Swordsmanship
Número da matrícula: 123456
Situação: Ativa
Data inicial: 01/01/2021
```

➢ **Exercício 10:** Definindo Classes de Formas Geométricas

O atributo raio é essencial porque ele é a base para qualquer cálculo envolvendo um círculo ou uma esfera.

- No caso do círculo, a área é calculada com π * raio². Ou seja, sem o raio, nem daria pra definir a figura corretamente.
- Já na esfera, o volume depende de 4/3 * π * raio³. Então, o raio não só define o tamanho da esfera, mas também seu volume total.

Basicamente, o raio é o que dá "proporção" à forma. Sem ele, não daria pra calcular nada nem saber o tamanho do objeto.

```
namespace exs10_11_12.models;

public class Circulo {
    double Raio { get; set; }
}
```

```
namespace exs10_11_12.models;

public class Esfera {
    double Raio { get; set; }
}
```

➢ **Exercício 11:** Criando Métodos de Cálculo

```
public double CalcularArea() {
    return Math.PI * (Raio * Raio);
}
public double CalcularVolume() {
    return (4.0 / 3.0) * Math.PI * (Raio * Raio * Raio);
}
```

➢ **Exercício 12:** Testando as Classes de Figuras

```
using exs10_11_12.models;

namespace exs10_11_12;

class Program {
    static void Main(string[] args) {
        Circulo circulo = new Circulo(10);
        double area = circulo.CalcularArea();
        Console.WriteLine($"Área do círculo:  {area:F2}");

        Esfera esfera = new Esfera(5);
        double volume = esfera.CalcularVolume();
        Console.WriteLine($"Volume da esfera: {volume:F2}");
    }
}
```

```
Área do círculo:  314,16
Volume da esfera: 523,60
```