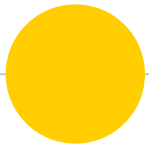
Introdução à Plataforma .NET

Profa. Luciana Balieiro Cosme
Ciência da Computação/IFNMG Montes Claros

Template SlidesCarnival



Classes e objetos

Métodos locais

```
static void CountDown(){
        int x = 10;
        Recursion(x);
System.Console.WriteLine("Terminou");
        void Recursion(int i){
            if (i <= 0) return;
            System.Console.WriteLine(i);
            Recursion(i - 1);
        }
```

Métodos de extensão

```
using ExtCostumizada;
…
string s = "Minha string
personalizada.";
int i = s.Contador();
System.Console.WriteLine(i);
```

```
namespace ExtCostumizada;
public static class StringExt{
    public static int Contador(this string str){
        char [] c = {' ', '.','?'};
        return str.Split(c ,
StringSplitOptions.RemoveEmptyEntries).Length;
    }
}
```

Herança

- Um dos principais conceitos em OO
- Reusar, estender e modificar o comportamento definido em outra classe
- Classe base e classe derivada

Herança

- Transitivo: classe A => B => C
- Classe derivada herda todos os membros, exceto construtores e destrutores

Exemplo (1/3)

```
public class Item{
  private static int id_atual;
  protected int ID{ get; set; }
  protected string Titulo{get;
set;}
  protected string Descricao{
get;set;}

  public Item(){
    ID = 0;
    Titulo="Default";
    Descricao="Default";
  }
```

```
public Item(string t,string desc){
      this.ID = GetNextID();
      this.Titulo = t;
      this.Descricao = desc;
    }
    static Item() {id_atual = 0;}
protected int GetNextID() {
    return ++id_atual;
    }
public override string ToString(){
    return
    $"{this.ID}-{this.Titulo}";
    }
}
```

Exemplo (2/3)

```
public class Requisicao : Item{
  public int outroID { get; set; }
  public Requisicao() { }
  public Requisicao(string t,string desc,int
ID){
        this.ID = GetNextID();
        this.Titulo = t;
        this.Descricao = desc;
        this.outroID  = ID;
    }
}
```

Exemplo (3/3)

```
Item item = new (t:"Fix Bugs", desc:"Corrigir
todos os bugs do branch");

Requisicao change = new ("Mudar a classe base",
                         "Adicionar membros",1);
Console.WriteLine(item.ToString());
Console.WriteLine(change.ToString());
```

1 - Fix Bugs

2 - Mudar a classe base

Downcast

```
Item outroitem = new();
outroitem = change; //obj de requisição
Console.WriteLine(outroitem.ToString());
```

2 - Mudar a classe base

```
//existe change.outroID
//mas não existe  outroitem.outroID = 0;
```

Upcast

```
Requisicao outrareq = new();
outrareq = (Requisicao) outroitem;
Console.WriteLine(outrareq.ToString());
```

Palavras-chave is

```
Item i = new Requisicao("Outra requisicao",
"Adicionar membros", 1);
if (i is Item) // if (i is Requisicao)
   Console.WriteLine("Classes compatíveis
por cast");
```

Palavras-chave as

```
object[] object_array = {item, "string", change,
"teste"};

for (int i = 0;i<object_array.Length;i++) {
    string? str=object_array[i] as string;
     if (str!=null)  Console.WriteLine(str);
     else  Console.WriteLine("Valor Null");
 }
```

```
Valor null
string
Valor null
teste
```

Pattern Matching

```
void Test(object? o){
      if (o is 5)
          System.Console.WriteLine("5");
       else if (o is int i)
     System.Console.WriteLine("int:"+i);
        else if (o is null)
       System.Console.WriteLine("null");
}
```

```
Test(5);
Test(1);
Test(null);
```

Exercício 5.2

A tarefa é determinar se uma sentença é um **pangrama**. Um pangrama é uma sentença que utiliza cada letra do alfabeto pelo menos uma vez. É insensível a maiúsculas e minúsculas.

O programa deve solicitar ao usuário se ele deseja jogar em **Inglês** ou **Português**. Deverá ter uma classe base com os atributos/métodos comuns e duas classes derivadas especializadas para a tarefa em cada idioma. Poderá ser em dupla.

Métodos abstratos e virtuais

- virtual: a classe derivada pode sobrescrever (override)
  - classe base pode implementar
- abstract: a classe derivada deve sobrescrever
  - a classe deve ser abstrata
- Base para o polimorfismo

Classe abstrata

- Previne de ser instanciada
- Usada apenas se uma classe for derivada
- Pode ter métodos abstratos
- Mas não precisa ter membros abstratos

Polimorfismo

- Objetos de uma classe derivada podem ser tratados como objetos de uma classe base
- o tipo declarado do objeto não é mais idêntico ao seu tipo em tempo de execução

Exemplo

```
public class Shape{
    public int X { get; set; }
    public int Y { get; set; }
    public virtual void Draw(){
        Console.WriteLine("Classe base");
    }
}
```

Exemplo

```
public class Circle : Shape{
    public override void Draw(){
        Console.WriteLine("Circulo");
        base.Draw();
    }
}
```

Exemplo

```
public class Rectangle : Shape{
    public override void Draw()     {
        Console.WriteLine("Rectangulo");
    }
}
public class Triangle : Shape{
    public override void Draw() {
        Console.WriteLine("Triangulo");
        base.Draw();
    }
}
```

Exemplo

```
var shapes = new List<Shape>{
new Rectangle(), new Triangle(),
new Circle() };
foreach (var shape in shapes){
            shape.Draw();
        }
```

Retangulo
Triangulo
Classe
Circulo
Classe

Interface

- Tipo que define um conjunto de membros
- Toda classe e estrutura que implementa uma interface deve ter o mesmo conj. de membros
- Pode ter uma implementação padrão
- Deriva-se de uma classe apenas, mas implementa-se de uma ou mais interfaces

Exemplo

```
interface ILogger{
    void Info(string message);
}
class ConsoleLogger : ILogger{
     public void Info(string message){
          System.Console.WriteLine(message);
     }
}
```

Exemplo

```
using System.Runtime.InteropServices;
class WindowsLogger: ILogger{
   [DllImport("User32.dll", CharSet=CharSet.Unicode)]
   public static extern int MessageBox(IntPtr h,
string m, string c, int type);

   public void Info(string message){
      MessageBox((IntPtr)0, message, "MessageBox", 0);
   }
}
```

Exemplo

```
using System.Runtime.InteropServices;

new ConsoleLogger().Info("teste console");
new WindowsLogger().Info("teste windows");
```

Exercício 6

Você deve implementar um programa C# que realiza a ordenação de uma lista de nomes utilizando diferentes estratégias de ordenação.

Exercício 6

Deve ter:

- Um interface IOrdenação que defina um contrato para as estratégias de ordenação. Essa interface deve conter um método Sort que recebe uma lista de strings como parâmetro.

Exercício 6

Deve ter:

- Duas classes concretas, que implementam a interface, e representem diferentes estratégias de ordenação: QuickSort (padrão) e MergeSort.

Exercício 6

Deve ter:

- Uma classe ListaOrdenada que atuará como o contexto as estratégias de ordenação.
  - Atributos: uma lista de strings e um do tipo IOrdenação

Exercício 6

Deve ter:

- No método Sort, da classe ListaOrdenada, chame o método Sort da estratégia de ordenação atual e, em seguida, itere sobre a lista de nomes e exiba-os.

Main

```
ListaOrdenada estudantes = new();
estudades.Add("Jose");
...
estudantes.SetEstrategia(new QuickSort());
estudantes.Sort();
estudantes.SetEstrategia(new ShellSort());
estudantes.Sort();
```

Referências

- Hello World => https://learn.microsoft.com/pt-br/dotnet/core/tutorials/with-visual-studio-code?pivots=dotnet-7-0
- Sistemas de tipos = > https://learn.microsoft.com/pt-br/dotnet/csharp/fundamentals/types/
- Tipos primitivos => https://learn.microsoft.com/pt-br/dotnet/csharp/fundamentals/types/https://learn.microsoft.com/pt-br/dotnet/csharp/language-reference/builtin-types/built-in-types

Referências

- Estruturas de controle => https://github.com/csharpfritz/csharp_with_csharpfritz/blob/main/notebooks/0104-LoopsAndConditionals.ipynb https://learn.microsoft.com/pt-br/dotnet/csharp/language-reference/statements/selection-statements?WT.mc_id=visualstudio-twitch-jefritz
- Switch = > https://learn.microsoft.com/pt-br/dotnet/csharp/language-reference/language-specification/statements#1383-the-switch-statement

Referências

- Operadores => https://learn.microsoft.com/pt-pt/dotnet/csharp/language-reference/operators/
- Expressões lambda => https://learn.microsoft.com/pt-pt/dotnet/csharp/language-reference/operators/lambda-expressions
- Operador de coalescência nula => https://learn.microsoft.com/pt-br/dotnet/csharp/language-reference/operators/null-coalescing-operator?WT.mc_id=visualstudio-twitch-jefritz
- Calculadora => https://learn.microsoft.com/pt-br/visualstudio/get-started/csharp/tutorial-console?view=vs-2022

Referências

- Parâmetros nomeados e opcionais => https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/named-and-optional-arguments
- Expressões lambda => https://learn.microsoft.com/pt-pt/dotnet/csharp/language-reference/operators/lambda-expressions
- Formas de arredondamento https://learn.microsoft.com/en-us/dotnet/api/system.midpointrounding?view=net-7.0

Referências

- Métodos de extensão => https://learn.microsoft.com/pt-br/dotnet/csharp/programming-guide/classes-and-structs/how-to-implement-and-call-a-custom-extension-method
- Classes
  https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/types/classes
- Herança
  https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/object-oriented/inheritance

Referências

- Polimorfismo

  https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/object-oriented/polymorphism

- Message Box

  https://learn.microsoft.com/pt-br/windows/win32/api/winuser/nf-winuser-messagebox?redirectedfrom=MSDN

Referências

Olsson, M. (2022). Class. In: C# 10 Quick Syntax Reference. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-7981-6_10