ORIENTAÇÃO À OBJETOS C# E INSTRUÇÕES BÁSICAS

Autor(es):

Douglas Gomes Ferreira de Moraes

@2TDS-2024

Sumário

1. ORIENTAÇÃO À OBJETOS (C#)

1.1. Introdução

Antes de aprofundamos nos pilares da orientação a objetos no C#, é importante entendermos o que exatamente é o conceito de orientação à objetos.

Orientação à objetos veio com uma proposta de tangibilizar o que estamos desenvolvendo com as características do mundo real. Quando estamos desenvolvendo algo com a programação estruturada, nós acabamos seguindo um fluxo único (seja ele: uma rotina única ou com sub-rotinas) onde caso precisemos replicar um código para um outro trecho do fluxo, basta copiar e colar onde nós desejarmos.

Outra diferença da Programação Estruturada com a Programação Orientada à Objetos está no acesso as variáveis declaradas no decorrer do código. Linguagens estruturadas não possuem muitas regras e restrições. Em linguagens fortemente orientadas a esse paradigma, a restrição do acesso a uma variável se resume a determinar se ela é visível ou não dentro de uma função (ou módulo, como no caso do uso da palavra-chave 'static' na linguagem C). No entanto, nativamente, não é possível especificar que uma variável só pode ser acessada por determinadas rotinas do programa.

Quando falamos que com a Programação Orientada à Objetos estamos nos aproximando no mundo real, estamos trazendo a palavra “objeto” para dar “forma” a algo genérico que, por sua vez, pode representar qualquer coisa.

E COMO FUNCIONA EXATAMENTE ESSE PARADIGMA DE OBJETO????

Quando entendemos que objeto pode ser qualquer coisa, podemos ficar com a dúvida de como classificar ele dentro do nosso código, e é neste ponto

que entramos nos conceitos principais para entender os objetos. Nesse tipo de programação nós vamos ter: **Classes e Objetos**.

Para entendermos melhor como funciona essa definição, note o desenho abaixo:

Quando definimos os objetos, eles estão sempre associados a uma classe. Mas por qual motivo isso acontece? Entenda que uma classe sempre terá um conjunto de características e comportamentos que definem algo. Já o objeto é uma instância ou um molde de uma classe, tornando palpável as características e comportamentos definidos.

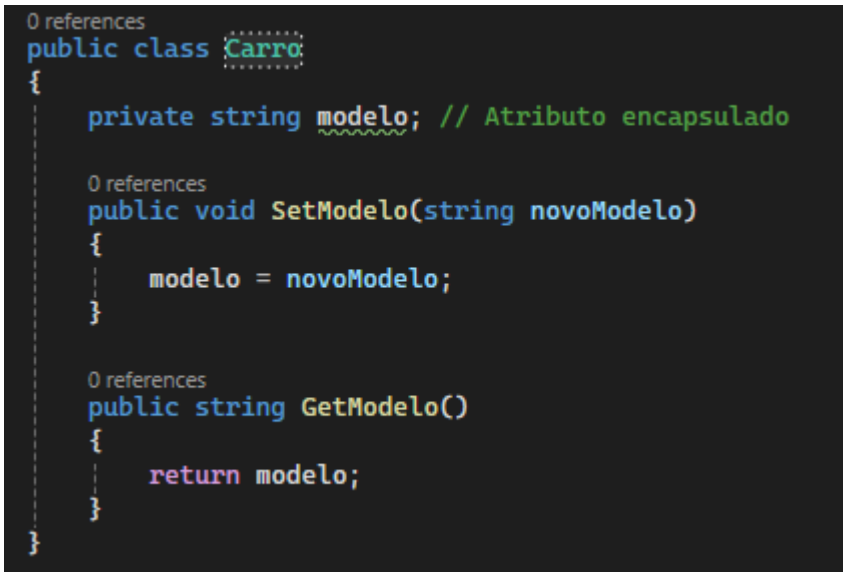
Agora, vamos entender um pouco mais sobre os pilares básicos da Orientação à Objetos!!!

1.2. Encapsulamento

O encapsulamento em C# refere-se à prática de ocultar os detalhes internos de uma classe e expor apenas o que é necessário para o uso externo.

Isso é alcançado através do uso de modificadores de acesso, como **public**, **private** e **protected**, para controlar o acesso aos membros (métodos, propriedades, campos) de uma classe.

Vamos entender como isso funciona em código:



No exemplo que utilizamos, modelo é encapsulado, o que significa que o seu acesso é controlado somente pelos métodos **SetModelo** e **GetModelo**, o que permite que a classe mantenha controle sobre como os dados são manipulados.

1.3. Herança em C#

A herança em C# permite que uma classe herde características (membros e comportamentos) de outra classe. A classe base é chamada de superclasse, e a classe que herda dela é chamada de subclasse.

Mas o que isso quer dizer? Imagine que eu tenho uma superclasse chamada “Animal” e uma subclasse chamada “Cachorro”. Todo animal pode emitir um som aleatório, já o cachorro tem uma característica de latir além dos demais sons que ele pode emitir. Nesse exemplo, eu consigo herdar a

característica de emitir som ao mesmo tempo que eu tenho a característica específica de latir. Veja o exemplo:

```
1 reference
public class Animal
{
    0 references
    public void EmitirSom()
    {
        Console.WriteLine("Som genérico de animal");
    }
}

0 references
public class Cachorro : Animal
{
    0 references
    public void Latir()
    {
        Console.WriteLine("Au Au");
    }
}
```

1.4. Polimorfismo em C#

O polimorfismo em C# refere-se à capacidade de uma classe base ser referenciada como sua classe derivada. Existem dois tipos principais de polimorfismo: polimorfismo de tempo de compilação (ou estático) e polimorfismo de tempo de execução (dinâmico).

```
2 references
public class Animal
{
    2 references
    public virtual void EmitirSom()
    {
        Console.WriteLine("Som genérico de animal");
    }
}

1 reference
public class Cachorro : Animal
{
    2 references
    public override void EmitirSom()
    {
        Console.WriteLine("Au Au");
    }
}

0 references
public class Exemplo
{
    0 references
    public void Teste()
    {
        // Uso do polimorfismo
        Animal meuAnimal = new Cachorro();
        meuAnimal.EmitirSom(); // Chama o método EmitirSom de Cachorro
    }
}
```

Neste exemplo, “**Animal**” possui um método “**EmitirSom**” marcado como “**virtual**”, e “**Cachorro”** sobrescreve esse método com a palavra-chave “**override**”. Ao criar uma instância de “**Cachorro**” e atribuí-la a uma variável do tipo “**Animal**”, podemos chamar o método “**EmitirSom**” da classe derivada (“**Cachorro**”) através da referência da classe base (“**Animal**”). Isso é polimorfismo de tempo de compilação.

2. INSTRUÇÕES BÁSICAS

O C# (C Sharp) é uma linguagem de programação rica em recursos e orientada a objetos. Aqui estão algumas das instruções básicas do C#:

2.1. Declaração de Variáveis:

```
//Declaração de Variáveis no C#
int idade;
string nome;
double salario;
```

2.2. Atribuição de Valores:

```
//Atribuição de Valores
idade = 25;
nome = "João";
salario = 50000.50;
```

2.3. Estruturas de Controle de Fluxo:

2.3.1. IF-ELSE:

```
18    //IF ELSE
19    if (idade >= 18)
20    {
21        //Output no console
22        Console.WriteLine("É maior de idade");
23    }
24    else
25    {
26        //Output no console
27        Console.WriteLine("É menor de idade");
28    }
```

2.3.2. SWITCH:

```
33  //Switch
34  var diaDaSemana = 3;
35
36  switch (diaDaSemana)
37  {
38      case 1:
39          Console.WriteLine("Domingo");
40          break;
41      // ...
42      default:
43          Console.WriteLine("Outro dia");
44          break;
45  }
```

2.3.3. WHILE:

```
//While
var contador = 1;

while (contador < 10)
{
    Console.WriteLine(contador);
    contador++;
}
```

2.3.4. FOR:

```
57
58  //FOR
59  for (int i = 0; i < 5; i++)
60  {
61      Console.WriteLine(i);
62  }
63
64
```

2.3.5. FOR EACH:

```
65  //For Each
66  int[] lista = { 1, 2, 3 };
67
68  foreach (var item in lista)
69  {
70      Console.WriteLine(item);
71  }
72
```

2.4. Métodos e Funções:

```
76  //-----------------------------------------------------------------------------------
77  //-----------------------------------------------------------------------------------
78  //Chamada de Métodos e Funções
79  ExibirMensagem("Teste Método");
80
    1 reference
81  void ExibirMensagem(string mensagem)
82  {
83      Console.WriteLine(mensagem);
84  }
85
```