



Object Oriented Programming

Curso C# Aplicando Princípios SOLID na prática

Aprenda a utilizar os conceitos da programação orientada a objetos na prática usando a linguagem C#

**Fundamentos – Exercícios :
Respostas**

Fundamentos – Exercícios - Resposta

1- Defina o que é classe e o que é objeto.

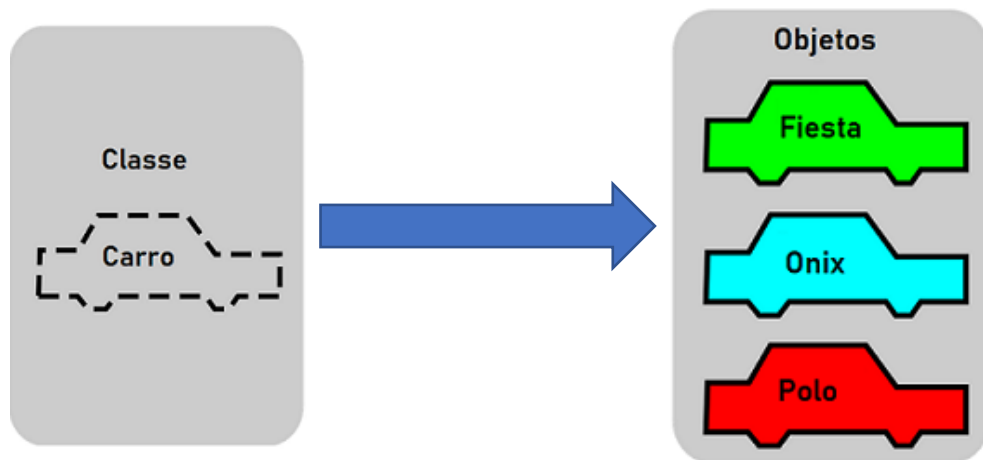
Uma classe é uma estrutura de dados e comportamentos que pode conter campos, constantes, métodos, propriedades, eventos, operadores, construtores e destrutores.

Uma classe é um modelo ou gabarito para a definição de objetos.

Um objeto não existe sem a definição de uma classe.

As classes são tipos e os objetos são instâncias das classes.

A classe funciona como um modelo, uma planta a partir da qual são criados os objetos.



Fundamentos – Exercícios - Resposta

1 - E como podemos criá-los na linguagem C#.

Na linguagem C# criamos uma classe usando a palavra chave **class** seguida do *nome da classe* e do corpo da classe definido entre chaves { } :

```
<modificador de acesso> class <nome>
{
    //membros da classe : campo, propriedade, método, construtor, eventos
}
```

```
public class Cliente
{
    //código
}
```

Criar um objeto de uma classe significa criar uma instância da classe.

Na linguagem C# para criar uma instância de uma classe usamos a palavra-chave **new**.

Cliente macoratti = new Cliente();

Cria uma *instância* da classe **Cliente()** ou seja cria o objeto **macoratti** do tipo **Cliente()**.

Fundamentos – Exercícios - Resposta

2- Como você define o conceito de abstração usado no paradigma da POO ?

A abstração é um dos três pilares da Programação Orientada a Objetos (OOP).

- Significa perceber uma entidade em um sistema ou contexto de uma perspectiva específica.
- Descartar os detalhes desnecessários e focar apenas nos aspectos necessários para esse contexto ou sistema em consideração.

Realizar a abstração é selecionar dados de um pool maior para mostrar apenas os detalhes relevantes do objeto ao usuário.

A abstração ajuda a reduzir a complexidade e o esforço da programação.

Fundamentos – Exercícios - Resposta

3- Dê exemplo de uma abstração do mundo real.

Um notebook é composto de várias partes como processador, RAM, placa mãe, tela LCD, câmera, portas USB, bateria, alto-falantes etc.

Para usá-lo, precisamos apenas saber como operar o notebook ligando o aparelho.

Não precisamos saber detalhes do funcionamento interno da construção do equipamento

Aqui, o notebook é um objeto projetado para expor apenas os recursos necessários ocultando seus detalhes de implementação.

Temos assim no notebook um exemplo de abstração do mundo real.

Fundamentos – Exercícios - Resposta

4- O que é encapsulamento e quais são os seus benefícios ?

O encapsulamento refere-se à capacidade de um objeto para ocultar dados e comportamentos que não são necessários para o usuário.

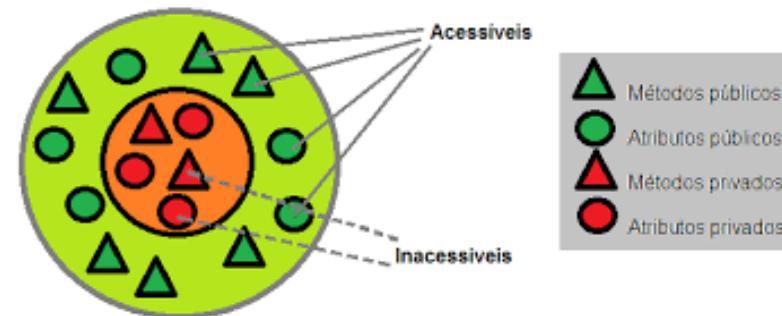
O encapsulamento permite que um grupo de propriedades, métodos e outros membros sejam considerados uma única unidade ou objeto.

No encapsulamento podemos usar os modificadores de acesso, e assim ocultar os detalhes do objeto e expor apenas métodos e propriedades necessários através da referência de um objeto

A linguagem C# suporta os seguintes modificadores de acesso: **Public, Private, Protected, Internal e Protected Internal.**

Benefícios :

- Proteção de dados contra corrupção accidental;
- Especificação da acessibilidade de cada um dos membros de uma classe;
- Flexibilidade e extensibilidade do código e redução na complexidade;
- Menor acoplamento entre objetos e melhoria na manutenção do código;



Fundamentos – Exercícios - Resposta


5- Quando o construtor de uma classe é chamado?

Um construtor é um método especial que é chamado automaticamente sempre que uma instância da classe for criada.

Assim quando criamos um objeto da classe o construtor é chamado.

```
class A
{
    public A() ←
    {
        Console.WriteLine("Executei o construtor da classe A");
    }
    public string Nome { get; set; }
}
```

```
static void Main(string[] args)
{
    var a1 = new A();
    Console.Read();
}
```

A horizontal arrow points from the 'new A()' expression in the Main method to the 'public A()' constructor definition in class A.

Fundamentos – Exercícios - Resposta

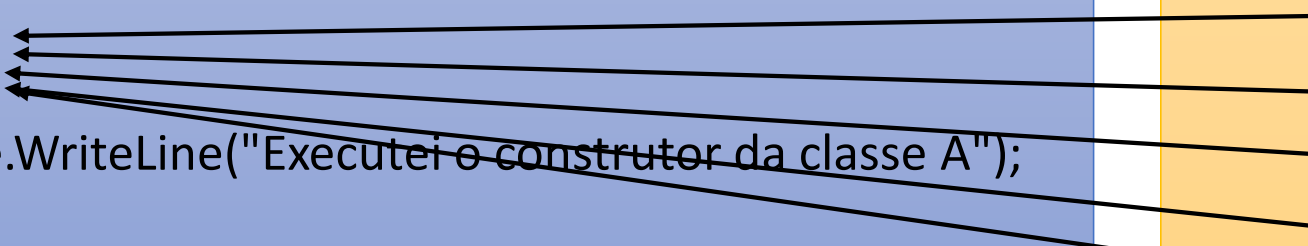
6- Se você criar 5 objetos de uma classe, quantas vezes o construtor da classe será chamado ?

Em toda criação de um objeto o construtor é chamado.

Assim, o construtor será chamado 5 vezes na criação de 5 objetos da classe.

```
class A
{
    public A()
    {
        Console.WriteLine("Executei o construtor da classe A");
    }
    public string Nome { get; set; }
}
```

```
static void Main(string[] args)
{
    var a1 = new A();
    var a2 = new A();
    var a3 = new A();
    var a4 = new A();
    var a5 = new A();
    Console.Read();
}
```



Fundamentos – Exercícios - Resposta

7- Por que você usa um construtor?

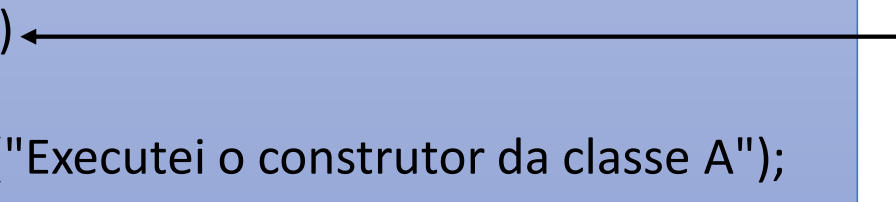
Um construtor é um *método especial da classe* que é chamado automaticamente sempre que uma instância da classe é criada.

Como métodos, um construtor também contém a coleção de instruções que são executadas no momento da criação do objeto.

Assim um construtor é útil *para inicializar e definir valores padrão* para os membros de dados do novo objeto.

```
class A
{
    public A(string nome)
    {
        Console.WriteLine("Executei o construtor da classe A");
        Nome = nome;
    }
    public string Nome { get; set; }
}
```

```
static void Main(string[] args)
{
    var a1 = new A("Teste");
    Console.Read();
}
```

A horizontal arrow points from the line 'var a1 = new A("Teste");' in the Main method to the 'public A(string nome)' constructor in class A. A diagonal arrow points from the 'nome' parameter in the constructor to the 'Nome' property in the constructor body.

Fundamentos – Exercícios - Resposta

8- Crie uma classe para calcular a soma de 2 números inteiros e/ou reais usando o conceito de sobrecarga de métodos. Se nenhum valor for informado retorne a mensagem *“Nenhum valor fornecido”*.

```
public class Calcular
{
    public string Somar()
    {
        return "Nenhum valor fornecido";
    }
    public int Somar(int x, int y)
    {
        return x + y;
    }
    public float Somar(float x, float y)
    {
        return x + y;
    }
    public float Somar(float x, int y)
    {
        return x + y;
    }
}
```

```
static void Main(string[] args)
{
    Calcular calc = new Calcular();

    var a = calc.Somar();
    Console.WriteLine(a);
    var b = calc.Somar(5, 4);
    Console.WriteLine(b);
    var c = calc.Somar(9.3f, 8.6f);
    Console.WriteLine(c);
    var d = calc.Somar(8.56f, 10);
    Console.WriteLine(d);

    Console.Read();
}
```

Fundamentos – Exercícios - Resposta

9- Crie uma classe para calcular a soma de até 4 números inteiros usando o conceito de parâmetros opcionais.

```
class Calculadora
{
    public static int Somar(int x, int y = 0, int z = 0, int w = 0)
    {
        return x + y + z + w;
    }
}
```

```
static void Main(string[] args)
{
    var a = Calculadora.Somar(1);
    Console.WriteLine(a);
    var b = Calculadora.Somar(1,2);
    Console.WriteLine(b);
    var c = Calculadora.Somar(1,2,3);
    Console.WriteLine(c);
    var d = Calculadora.Somar(1,2,3,4);
    Console.WriteLine(d);
    Console.Read();
}
```

Fundamentos – Exercícios - Resposta

10- Qual o conceito de herança ?

Herança é um conceito chave usado na POO para descrever uma relação entre as classes;

Por meio da herança uma classe *copia* ou herda todas as propriedades, atributos e métodos de uma outra classe, podendo assim estender sua funcionalidade;

A classe que cede os membros para a outra classe é chamada **superclasse**, classe **pai** ou classe **base**;

A classe que herda os membros da outra classe é chamada **subclasse** ou classe **derivada**;

A herança permite a reutilização de código e especifica um relacionamento de *especialização/generalização* do tipo **"é um"**;

A herança deve ser usada somente quando necessário pois pode levar a um acoplamento forte entre as classes do seu projeto dificultando a sua *reusabilidade* e a *manutenção*;

Existe a *herança de implementação* onde uma classe derivada herda da classe base e a *herança de interface* onde uma classe pode *herdar de uma interface*, neste caso apenas as assinaturas dos métodos são herdados sendo que os métodos devem ser implementados;

Fundamentos – Exercícios - Resposta

10- Dê um exemplo.

```
class A
{
    public int a;
    public A() { Console.WriteLine("construtor A"); }

    public void M(string nome) { Console.WriteLine("sou o " + nome); }
}
```

Classe Pai ou Base

```
class B : A
{
    int b;
    public B() { Console.WriteLine("construtor B"); }

    public void J() { Console.WriteLine("sou J "); }
}
```

Classe filha ou derivada

Criando uma instância da classe derivada :

B b1 = new B(); ou
var b1 = new B();

O objeto **b1** acessa aos membros públicos da classe A