Programação Orientada à Objetos

Aula 06
Métodos Construtores e Primeiros Relacionamentos

Henrique Poyatos
henrique.poyatos@bandtec.com.br

Métodos Construtores

O que são ?

- Trata-se de métodos que possuem um procedimento de como ***construir*** o objeto da maneira desejada.
- Este tipo de método será chamado automaticamente no momento da instanciação do objeto.
- Na C# (e outras linguagens), os métodos construtores podem ser facilmente identificados por dois fatores:
  - Possuem sempre o mesmo nome da classe;
  - Não possuem tipo de retorno declarado (nem mesmo void);
- Um classe pode possuir um ou mais métodos construtores (diferem pela sua assinatura) – chamamos isso de sobrecarga de método.
- Toda classe C# possui um método construtor - quando não é nenhum declarado um vazio é disponibilizado automaticamente.

Método Construtor
Declaração

Sintaxe

```
public class Carro
{
    public int placa;
    public string marca;
    public string modelo;
    public string cor;

    public Carro()
    {
        this.cor = "Azul";
    }
}
```

Ausência de tipo de dado de retorno → Construtores são procedimentos que não retornam informação.

Nome do método idêntico ao nome da classe → Rigorosamente, letras maiúsculas e minúsculas.

Procedimentos do construtor → Neste exemplo, todos os carros instanciados terão como cor padrão o “Azul”.

Chamada ao método Construtor

Sintaxe

```
class Program
{
    static void main(string args[
    {
      Carro meuCarro = new Carro();
      Console.WriteLine(meuCarro.cor);


    }
}
```

Chamada do Método Construtor → O procedimento será executado neste Momento, definindo automaticamente o atributo "cor" como "Azul".

Acesso externo à um atributo → Para acessar um atributo de um objeto, deve-se informar o nome do objeto, o símbolo "." (ponto), e o nome do atributo (ou mesmo método) desejado. Isso é possível pois o tipo de acesso é público (*public*).

Sobrecarga de método (Overloading)

- Um classe pode possuir métodos declarados uma ou mais vezes (sobrecarga de método).
- São considerados assim os métodos que possuem o mesmo nome.
- Apesar de possuírem o mesmo nome, precisam obrigatoriamente ter assinaturas diferentes (tipos ou quantidades de parâmetros diferentes).

Método Construtor
Declaração

```
public class Carro
{
    public int placa;
    public string marca;
    public string modelo;
    public string cor;

    public Carro()
    {
        this.cor = "Azul";
    }

    public Carro(String cor)
    {
        this.cor = cor;
    }
}
```

Sobrecarga de método (Overloading) → Dois métodos, mesmo nome.

Assinaturas diferentes → Um deles possui um parâmetro; o outro, parâmetro algum.

A importância da instrução "this" → this.cor refere-se ao atributo cor, presente na classe. A variável "cor", sem a palavra "this" refere-se à variável local (declarada como parâmetro).

Chamada de método em sobrecarga

Sintaxe

```
class Program
{
    static void main(String args[]
    {
      Carro meuCarro = new Carro();
      Console.WriteLine(meuCarro.cor);

      Carro seuCarro = new Carro("Vermelho");
      Console.WriteLine(seuCarro.cor);
    }
}
```

Chamada de método construtor → O Java irá procurar por um método Carro() que não possua parâmetro algum. Localiza o mesmo, portanto, “meuCarro” é “Azul”.

Chamada de método construtor → O Java irá procurar por um método Carro() que possua um parâmetro só, do tipo String. Localiza o mesmo, portanto, “seuCarro” é “Vermelho”.

Vale para qualquer método → O exemplo é com construtor, mas a técnica vale para qualquer tipo de método.

Primeiros Relacionamentos entre objetos

- A orientação à objetos incentiva ao desmembramento de códigos em pedaços menores, encapsulados em métodos dentro de classes.
- Entretanto, estes trechos de códigos precisam relacionar-se entre si.
- E se o nosso carro precisar de informações de seu proprietário ?

Primeiros Relacionamentos entre objetos

Sintaxe

```
namespace Projeto
{
    public class Carro
    {
        public int placa;
        public string marca;
        public string modelo;
        public string cor;
        public string proprietario;
    }
}
```

Primeiros Relacionamentos entre objetos

- Ok... Mas se eu precisar de MAIS informações do proprietário?
- Como rg e cpf, por exemplo?

Primeiros Relacionamentos entre objetos

Sintaxe

```
namespace Projeto
{
    public class Carro
    {
        public int placa;
        public string marca;
        public string modelo;
        public string cor;
        public string proprietario;
        public string rgProprietario;
        public string cpfProprietario;
    }
}
```

Parece uma boa abordagem ?

Primeiros Relacionamentos entre objetos

- Não, esta não seria uma boa abordagem.
- Estamos FERINDO a OOP desta forma.. RG e CPF são características do Carro?
- Não .. São características do DONO do carro.
- O exemplo anterior CONTAMINA a Classe Carro com atributos que não pertencem à ele.
- Desta maneira, não seria melhor construir uma Classe que abrigue características do dono do carro e relacioná-las de alguma forma?

Primeiros Relacionamentos entre objetos

Sintaxe

```
namespace Projeto
{
    public class Proprietario
    {
        public int codigo;
        public string nome;
        public string rg;
        public string cpf;
    }
}
```

Respeito ao OOP

Classe Proprietário
com atributos pertinentes
ao mesmo

Primeiros Relacionamentos entre objetos

Sintaxe

```
namespace Projeto
{
    public class Carro
    {
        public int placa;
        public string marca;
        public string modelo;
        public string cor;
        public Proprietario proprietario;
    }
}
```

O "pulo do gato"

O atributo proprietário não será um primitivo, e sim uma classe que criamos para este fim.

Tipo do atributo é a classe Proprietário

Nome do atributo é **p**roprietário (Poderia ser diferente, como "dono")

Primeiros Relacionamentos entre objetos

Sintaxe

Exemplo de Utilização

```
namespace Projeto
{
    Class Program
    {
        static void Main(string[] args)
        {
            Carro meuCarro = new Carro();
            meuCarro.marca = "Ferrari";

            Proprietario eu = new Proprietario();
            eu.nome = "Henrique";
            eu.rg = "12.345.678-X";
            meuCarro.proprietario = eu;

            // Mostrando meu rg
            Console.WriteLine(carro1.proprietario.rg);
        }
    }
}
```