Curso C# Completo Programação Orientada a Objetos + Projetos

Capítulo: Construtores, palavra this, sobrecarga, encapsulamento

http://educandoweb.com.br

Prof. Dr. Nelio Alves

Construtores

http://educandoweb.com.br

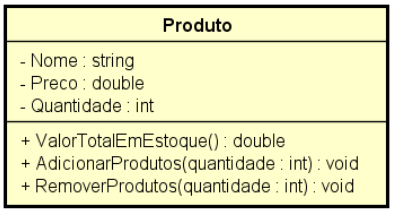
Prof. Dr. Nelio Alves

Construtor Força quem está programando a atribuir valor ao um ou mais atributos

- É uma operação especial da classe, que executa no momento da instanciação do objeto
- Usos comuns:
  - Iniciar valores dos atributos
  - Permitir ou obrigar que o objeto receba dados / dependências no momento de sua instanciação (injeção de dependência)
- Se um construtor customizado não for especificado, a classe disponibiliza o construtor padrão:
  - Produto p = new Produto();
- É possível especificar mais de um construtor na mesma classe (sobrecarga)

Exemplo:

```
Entre os dados do produto:
Nome: TV
Preço: 900.00
Quantidade no estoque: 10

Dados do produto: TV, $ 900.00, 10 unidades, Total: $ 9000.00

Digite o número de produtos a ser adicionado ao estoque: 5

Dados atualizados: TV, $ 900.00, 15 unidades, Total: $ 13500.00

Digite o número de produtos a ser removido do estoque: 3

Dados atualizados: TV, $ 900.00, 12 unidades, Total: $ 10800.00
```



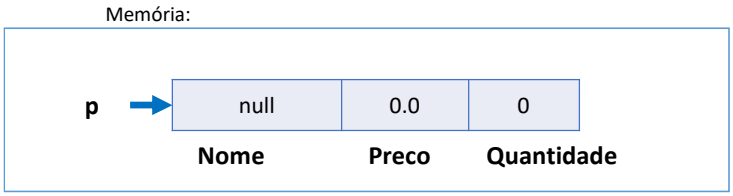
```
using System.Globalization;

namespace Course {
    class Produto {

        public string Nome;
        public double Preco;
        public int Quantidade;

        public double ValorTotalEmEstoque() {
            return Preco * Quantidade;
        }

        public void AdicionarProdutos(int quantidade) {
            Quantidade += quantidade;
        }

        public void RemoverProdutos(int quantidade) {
            Quantidade -= quantidade;
        }

        public override string ToString() {
            return Nome
                + ", $ "
                + Preco.ToString("F2", CultureInfo.InvariantCulture)
                + ", "
                + Quantidade
                + " unidades, Total: $ "
                + ValorTotalEmEstoque().ToString("F2", CultureInfo.InvariantCulture);
        }
    }
}
```

```
using System;
using System.Globalization;

namespace Course {
    class Program {
        static void Main(string[] args) {

            Produto p = new Produto();

            Console.WriteLine("Entre os dados do produto:");
            Console.Write("Nome: ");
            p.Nome = Console.ReadLine();
            Console.Write("Preço: ");
            p.Preco = double.Parse(Console.ReadLine(), CultureInfo.InvariantCulture);
            Console.Write("Quantidade no estoque: ");
            p.Quantidade = int.Parse(Console.ReadLine());

            Console.WriteLine();
            Console.WriteLine("Dados do produto: " + p);

            Console.WriteLine();
            Console.Write("Digite o número de produtos a ser adicionado ao estoque: ");
            int qte = int.Parse(Console.ReadLine());
            p.AdicionarProdutos(qte);

            Console.WriteLine();
            Console.WriteLine("Dados atualizados: " + p);

            Console.WriteLine();
            Console.Write("Digite o número de produtos a ser removido do estoque: ");
            qte = int.Parse(Console.ReadLine());
            p.RemoverProdutos(qte);

            Console.WriteLine();
            Console.WriteLine("Dados atualizados: " + p);
        }
    }
}
```

Proposta de melhoria

Quando executamos o comando abaixo, instanciamos um produto "**p**" com seus atributos “vazios”:

p = new Produto();



Entretanto, faz sentido um produto que não tem nome? Faz sentido um produto que não tem preço?

Com o intuito de evitar a existência de produtos sem nome e sem preço, é possível fazer com que seja “obrigatória” a iniciação desses valores?

```
using System.Globalization;

namespace Course {
    class Produto {

        public string Nome;
        public double Preco;
        public int Quantidade;

        public Produto(string nome, double preco, int quantidade) {
            Nome = nome;
            Preco = preco;
            Quantidade = quantidade;
        }

        public double ValorTotalEmEstoque() {
            return Preco * Quantidade;
        }

        public void AdicionarProdutos(int quantidade) {
            Quantidade += quantidade;
        }

        public void RemoverProdutos(int quantidade) {
            Quantidade -= quantidade;
        }

        public override string ToString() {
            return Nome
                + ", $ "
                + Preco.ToString("F2", CultureInfo.InvariantCulture)
                + ", "
                + Quantidade
                + " unidades, Total: $ "
                + ValorTotalEmEstoque().ToString("F2", CultureInfo.InvariantCulture);
        }
    }
}
```

Construtor

```
using System;
using System.Globalization;

namespace Course {
    class Program {
        static void Main(string[] args) {

            Console.WriteLine("Entre os dados do produto:");
            Console.Write("Nome: ");
            string nome = Console.ReadLine();
            Console.Write("Preço: ");
            double preco = double.Parse(Console.ReadLine(), CultureInfo.InvariantCulture);
            Console.Write("Quantidade no estoque: ");
            int quantidade = int.Parse(Console.ReadLine());

            Produto p = new Produto(nome, preco, quantidade);

            Console.WriteLine();
            Console.WriteLine("Dados do produto: " + p);

            Console.WriteLine();
            Console.Write("Digite o número de produtos a ser adicionado ao estoque: ");
            int qte = int.Parse(Console.ReadLine());
            p.AdicionarProdutos(qte);

            Console.WriteLine();
            Console.WriteLine("Dados atualizados: " + p);

            Console.WriteLine();
            Console.Write("Digite o número de produtos a ser removido do estoque: ");
            qte = int.Parse(Console.ReadLine());
            p.RemoverProdutos(qte);

            Console.WriteLine();
            Console.WriteLine("Dados atualizados: " + p);
        }
    }
}
```

variáveis auxiliares

Sobrecarga

http://educandoweb.com.br

Prof. Dr. Nelio Alves

Sobrecarga

- É um recurso que uma classe possui de oferecer mais de uma operação com o mesmo nome, porém com diferentes listas de parâmetros.

Proposta de melhoria

- Vamos criar um construtor opcional, o qual recebe apenas nome e preço do produto. A quantidade em estoque deste novo produto, por padrão, deverá então ser iniciada com o valor zero.

- Nota: é possível também incluir um **construtor padrão** (sem parâmetros)

```
public Produto() {
}

public Produto(string nome, double preco, int quantidade) {
    Nome = nome;
    Preco = preco;
    Quantidade = quantidade;
}

public Produto(string nome, double preco) {
    Nome = nome;
    Preco = preco;
    Quantidade = 0;
}
```

Sintaxe alternativa para inicializar valores

http://educandoweb.com.br

Prof. Dr. Nelio Alves

```
using System.Globalization;

namespace Course {
    class Produto {

        public string Nome;
        public double Preco;
        public int Quantidade;

        public Produto() {
        }

        public Produto(string nome, double preco, int quantidade) {
            Nome = nome;
            Preco = preco;
            Quantidade = quantidade;
        }

        (...)
```

```
Produto p = new Produto("TV", 900.00, 10);
```

```
Produto p = new Produto {
    Nome = "TV",
    Preco = 900.0,
    Quantidade = 0
};

Produto p2 = new Produto() {
    Nome = "TV",
    Preco = 900.0,
    Quantidade = 0
};
```

Isso funciona mesmo se a classe não possuir construtores implementados

Palavra this

http://educandoweb.com.br

Prof. Dr. Nelio Alves

Palavra this

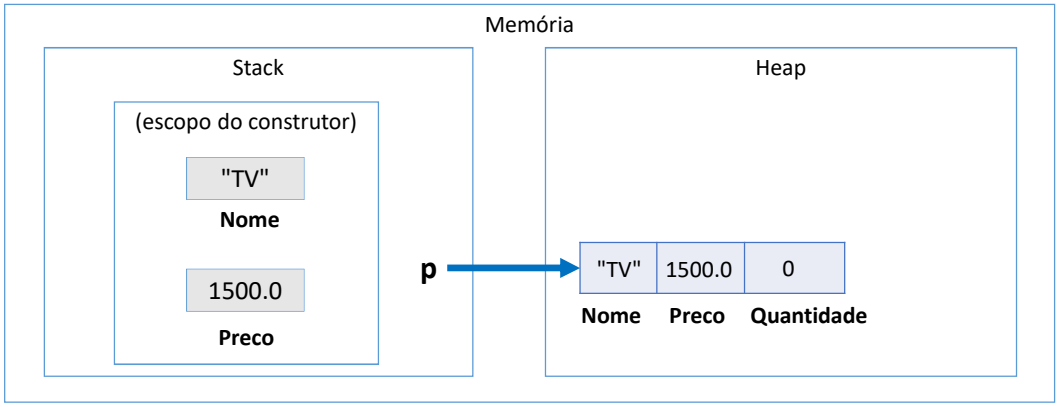
- É uma referência para o próprio objeto

- Usos comuns:
  - Diferenciar atributos de variáveis locais (Java)
  - Referenciar outro construtor em um construtor
  - Passar o próprio objeto como argumento na chamada de um método ou construtor

Diferenciar atributos de variáveis locais

```
Produto p = new Produto("TV", 1500.0);
```

```
public Produto(string Nome, double Preco) {
    this.Nome = Nome;
    this.Preco = Preco;
    Quantidade = 0;
}
```



Referenciar outro construtor em um construtor

```
using System.Globalization;

namespace Course {
    class Produto {

        public string Nome;
        public double Preco;
        public int Quantidade;

        public Produto() {
            Quantidade = 0;
        }

        public Produto(string nome, double preco) : this() {
            Nome = nome;
            Preco = preco;
        }

        public Produto(string nome, double preco, int quantidade) : this(nome, preco) {
            Quantidade = quantidade;
        }

        (...)
```

Passar o próprio objeto como argumento na chamada de um método ou construtor

```
class ChessMatch {

    (...)

    PlaceNewPiece('e', 1, new King(board, Color.White, this));

    (...)
```

Encapsulamento

http://educandoweb.com.br

Prof. Dr. Nelio Alves

Controle de acesso com properties;

Moderar o acesso externo a um atributo utilizando o "private";

Na orientação a objetos, esconder os detalhes de implementação de uma classe é um conceito conhecido como encapsulamento. Como os detalhes de implementação da classe estão escondidos, todo o acesso deve ser feito através de seus métodos públicos. Não permitimos aos outros saber COMO a classe faz o trabalho dela, mostrando apenas O QUÊ ela faz.

Encapsulamento

- É um princípio que consiste em esconder detalhes de implementação de um componente, expondo apenas operações seguras e que o mantenha em um estado consistente.
- Regra de ouro: o objeto deve sempre estar em um estado consistente, e a própria classe deve garantir isso.

Analogia:

Opção 1: implementação manual

- Todo atributo é definido como private

obtém o valor de um atributo

- Implementa-se métodos Get e Set para cada atributo, conforme regras de negócio

definir/alterar valor de um atributo

- Nota: não é usual na plataforma C#

```
using System.Globalization;

namespace Course {
    class Produto {

        private string _nome;
        private double _preco;
        private int _quantidade;

        public Produto() {
        }

        public Produto(string nome, double preco, int quantidade) {
            _nome = nome;
            _preco = preco;
            _quantidade = quantidade;
        }

        public string GetNome() {
            return _nome;
        }

        public void SetNome(string nome) {
            if (nome != null && nome.Length > 1) {
                _nome = nome;
            }
        }

        public double GetPreco() {
            return _preco;
        }
```

Padrão de nome para atributos privativos.

Função que permite que outros arquivos acessem o valor do atributo privativo "nome".

Atributos privativos permitem que você defina condições para acessá-los e/ou alterá-los.

```
        public int GetQuantidade() {
            return _quantidade;
        }

        public double ValorTotalEmEstoque() {
            return _preco * _quantidade;
        }

        public void AdicionarProdutos(int quantidade) {
            _quantidade += quantidade;
        }

        public void RemoverProdutos(int quantidade) {
            _quantidade -= quantidade;
        }

        public override string ToString() {
            return _nome
                + ", $ "
                + _preco.ToString("F2", CultureInfo.InvariantCulture)
                + ", "
                + _quantidade
                + " unidades, Total: $ "
                + ValorTotalEmEstoque().ToString("F2", CultureInfo.InvariantCulture);
        }
    }
}
```

Properties

http://educandoweb.com.br

Prof. Dr. Nelio Alves

Propriedades

- São definições de métodos encapsulados, porém expondo uma sintaxe similar à de atributos e não de métodos

- https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/classes-and-structs/properties
  - Uma propriedade é um membro que oferece um mecanismo flexível para ler, gravar ou calcular o valor de um campo particular. As propriedades podem ser usadas como se fossem atributos públicos, mas na verdade elas são métodos especiais chamados "acessadores". Isso permite que os dados sejam acessados facilmente e ainda ajuda a promover a segurança e a flexibilidade dos métodos.

Facilitam o acesso a atributos

```
using System.Globalization;

namespace Course {
    class Produto {

        private string _nome;
        private double _preco;
        private int _quantidade;

        public Produto() {
        }

        public Produto(string nome, double preco, int quantidade) {
            _nome = nome;
            _preco = preco;
            _quantidade = quantidade;
        }

        public string Nome {
            get { return _nome; }
            set {
                if (value != null && value.Length > 1) {
                    _nome = value;
                }
            }
        }

        public double Preco {
            get { return _preco; }
        }

        public int Quantidade {
            get { return _quantidade; }
        }
```

Apesar de _nome, _preço e _quantidade serem atributos privativos, as properties facilitam o acesso a eles a partir do get e do set.

Graças às propriedades, para alterar o nome de um produto, não é mais necessário chamar um método como no programa anterior, basta declarar "p.Nome" e a propriedade irá alterá-lo de acordo com as condições.

Permite que o preço seja lido apenas, a partir do comando Console.WriteLine (p.Preco); no programa principal.

```
        public double ValorTotalEmEstoque {
            get { return _preco * _quantidade; }
        }

        public void AdicionarProdutos(int quantidade) {
            _quantidade += quantidade;
        }

        public void RemoverProdutos(int quantidade) {
            _quantidade -= quantidade;
        }

        public override string ToString() {
            return _nome
                + ", $ "
                + _preco.ToString("F2", CultureInfo.InvariantCulture)
                + ", "
                + _quantidade
                + " unidades, Total: $ "
                + ValorTotalEmEstoque.ToString("F2", CultureInfo.InvariantCulture);
        }
    }
}
```

Auto Properties

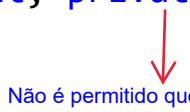
http://educandoweb.com.br

Prof. Dr. Nelio Alves

Propriedades autoimplementadas

- É uma forma simplificada de se declarar propriedades que não necessitam lógicas particulares para as operações get e set.

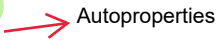
```
public double Preco { get; private set; }
```



Não é permitido que outros arquivos alterem o valor do preço.

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/auto-implemented-properties

```
using System.Globalization;

namespace Course {
    class Produto {

        private string _nome;
        public double Preco { get; private set; }
        public double Quantidade { get; set; }

        public Produto() {
        }

        public Produto(string nome, double preco, int quantidade) {
            _nome = nome;
            Preco = preco;
            Quantidade = quantidade;
        }

        public string Nome {
            get { return _nome; }
            set {
                if (value != null && value.Length > 1) {
                    _nome = value;
                }
            }
        }
```




```
        public double ValorTotalEmEstoque {
            get { return Preco * Quantidade; }
        }

        public void AdicionarProdutos(int quantidade) {
            Quantidade += quantidade;
        }

        public void RemoverProdutos(int quantidade) {
            Quantidade -= quantidade;
        }

        public override string ToString() {
            return _nome
                + ", $ "
                + Preco.ToString("F2", CultureInfo.InvariantCulture)
                + ", "
                + Quantidade
                + " unidades, Total: $ "
                + ValorTotalEmEstoque.ToString("F2", CultureInfo.InvariantCulture);
        }
    }
}
```

Ordem sugerida para implementação de membros

http://educandoweb.com.br
Prof. Dr. Nelio Alves

Ordem sugerida

- Atributos privados
- Propriedades autoimplementadas
- Construtores
- Propriedades customizadas
- Outros métodos da classe

Modificadores e acesso

http://educandoweb.com.br

Prof. Dr. Nelio Alves

Modificadores de acesso

- https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/access-modifiers

Membros

QUEM TEM ACESSO

	própria classe	subclasses no assembly	classes do assembly	subclasses fora do assembly	classes fora do assembly
public	x	x	x	x	x
protected internal	x	x	x	x	
internal	x	x	x		
protected	x	x		x	
private protected	x	x			
private	x				

Classes

- Acesso por qualquer classe
  - public class Product
- Acesso somente dentro do assembly
  - internal class Product
  - class Product

Apenas outras classes do mesmo projeto podem acessar esse tipo de classe.

- Acesso somente pela classe-mãe
  - private class Product
  - Nota: classe aninhada, por padrão, é private

Uma classe dentro da outra

Exercício de fixação

http://educandoweb.com.br

Prof. Dr. Nelio Alves

Em um banco, para se cadastrar uma conta bancária, é necessário informar o número da conta, o nome do titular da conta, e o valor de depósito inicial que o titular depositou ao abrir a conta. Este valor de depósito inicial, entretanto, é opcional, ou seja: se o titular não tiver dinheiro a depositar no momento de abrir sua conta, o depósito inicial não será feito e o saldo inicial da conta será, naturalmente, zero.

Importante: uma vez que uma conta bancária foi aberta, o número da conta nunca poderá ser alterado. Já o nome do titular pode ser alterado (pois uma pessoa pode mudar de nome por ocasião de casamento, por exemplo).

Por fim, o saldo da conta não pode ser alterado livremente. É preciso haver um mecanismo para proteger isso. O saldo só aumenta por meio de depósitos, e só diminui por meio de saques. Para cada saque realizado, o banco cobra uma taxa de $ 5.00. Nota: a conta pode ficar com saldo negativo se o saldo não for suficiente para realizar o saque e/ou pagar a taxa.

Você deve fazer um programa que realize o cadastro de uma conta, dando opção para que seja ou não informado o valor de depósito inicial. Em seguida, realizar um depósito e depois um saque, sempre mostrando os dados da conta após cada operação.

(exemplos nas próximas páginas)

EXEMPLO 1

```
Entre o número da conta: 8532
Entre o titular da conta: Alex Green
Haverá depósito inicial (s/n)? s
Entre o valor de depósito inicial: 500.00

Dados da conta:
Conta 8532, Titular: Alex Green, Saldo: $ 500.00

Entre um valor para depósito: 200.00
Dados da conta atualizados:
Conta 8532, Titular: Alex Green, Saldo: $ 700.00

Entre um valor para saque: 300.00
Dados da conta atualizados:
Conta 8532, Titular: Alex Green, Saldo: $ 395.00
```

EXEMPLO 2

```
Entre o número da conta: 7801
Entre o titular da conta: Maria Brown
Haverá depósito inicial (s/n)? n

Dados da conta:
Conta 7801, Titular: Maria Brown, Saldo: $ 0.00

Entre um valor para depósito: 200.00
Dados da conta atualizados:
Conta 7801, Titular: Maria Brown, Saldo: $ 200.00

Entre um valor para saque: 198.00
Dados da conta atualizados:
Conta 7801, Titular: Maria Brown, Saldo: $ -3.00
```

Correção do exercício de fixação

http://educandoweb.com.br

Prof. Dr. Nelio Alves

Código fonte no Github

https://github.com/acenelio/encapsulamento1-csharp