


Fundamentos de C#

Objetos e Elementos Estáticos




Softblue
cursos online

Tópicos Abordados




- Inicialização dos fields
- Construtores de classes e estruturas
 - Construtores com parâmetros
 - Sobrecarga de construtores
 - Construtor padrão
- O operador *this*
- Encadeando construtores

Tópicos Abordados



- Modificador *static*
 - Fields estáticos
 - Métodos estáticos
 - Construtores estáticos
 - Classes estáticas
 - O modificador *static* em estruturas
- Constantes
- Read-only fields

Inicialização dos Fields



- No C#, variáveis locais não são inicializadas automaticamente

```
double valor;  
double valorMaisUm = valor + 1;
```


Erro de compilação

- No entanto, fields de uma classe ou estrutura são inicializados

```
class Livro  
{  
    string titulo;  
    int numPaginas;  
    double preco;  
}
```

Inicializados automaticamente


Inicialização dos Fields



- O valor padrão assumido pelo field é determinado pelo tipo do dado

Tipo	Valor padrão
Tipos numéricos (<i>int, short, byte, long, float, double, etc.</i>)	0
<i>bool</i>	<i>false</i>
<i>char</i>	<i>'\0'</i>
Reference Types (classes)	<i>null</i>
Estruturas	<i>Value types</i> assumem o valor padrão. <i>Reference types</i> assumem <i>null</i> .

Inicialização dos Fields



- Um field pode ser inicializado ao ser declarado

```
class Livro  
{  
    string titulo;  
    int numPaginas = 300;  
    double preco = 80.0;  
}
```

Todo objeto da classe *Livro* assume inicialmente:
titulo = null
numPaginas = 300
preco = 80.0

Construtores

Softblue

- É um “método especial” chamado em um objeto quando ele é criado
- Utilizado para inicializar o estado do objeto

```
class Livro
{
    public string isbn;
    public string titulo;
    public string autor;
    public int numPaginas;
    public double preco;

    public Livro()
    {
        //...
    }
}
```

Regras:
- Nome igual ao da classe
- Não define tipo de retorno

Construtor

Construtores

Softblue

- O construtor é chamado no objeto assim que ele é instanciado
– Operador **new()**

```
Livro l = new Livro();
```

```
class Livro
{
    public Livro()
    {
        //...
    }
}
```

Um construtor normalmente tem o papel de definir valores iniciais para os fields do objeto

```
public Livro()
{
    numPaginas = 300;
    preco = 80.0;
}
```

Construtores com Parâmetros

Softblue

- Assim como métodos, construtores podem receber parâmetros

```
class Livro
{
    public Livro(int numPaginas, double preco)
    {
        //...
    }
}
```

Construtor que recebe dois parâmetros (int, double)

- Os parâmetros são fornecidos no momento de instanciar o objeto

```
Livro l = new Livro(300, 80.0);
```

Chama o construtor com os parâmetros (int, double)

Sobrecarga de Construtores

Softblue

- Assim como métodos, construtores também podem ser sobrecarregados

```
Livro()
{
}

Livro(int numPaginas, double preco)
{
}

Livro(string isbn)
{
}

Livro(string titulo, string autor)
{
}
```

O construtor que vai executar depende de como o operador `new()` é chamado

Construtor Padrão

Softblue

- Uma classe sempre deve ter um construtor definido
- Se você não definir um construtor explicitamente, o compilador faz isso

```
class Livro
{
    public Livro()
    {
    }
}
```

Construtor padrão: sem parâmetros e sem implementação

Construtor Padrão

Softblue

- A partir do momento que você define algum construtor para a classe, o construtor padrão não é mais gerado

```
class Livro
{
    public Livro(string isbn)
    {
    }
}
```

Este é o único construtor da classe

```
Livro l = new Livro();
```

Não funciona, pois não existe construtor correspondente

O Operador *this*



- O **this** é uma forma do objeto referenciar ele próprio
- O uso do *this* é comum na resolução de problemas de ambiguidade entre variáveis

```
class Livro
{
    int numPaginas;
    public Livro(int numPaginas)
    {
        numPaginas = numPaginas;
    }
}
```

Field

Parâmetro

?

O Operador *this*



- O *this* permite diferenciar um field de um parâmetro quando ambos tem o mesmo nome

```
class Livro
{
    int numPaginas;
    public Livro(int numPaginas)
    {
        this.numPaginas = numPaginas;
    }
}
```

Field

Parâmetro

O Operador *this*



- O *this* pode ser usado em situações onde referencia fields ou métodos do próprio objeto
- O uso não é obrigatório

```
class Livro
{
    string isbn;
    public void ImprimirIsbn()
    {
        Console.WriteLine(this.isbn);
    }
    public void ImprimirInfo()
    {
        this.ImprimirIsbn();
        Console.WriteLine(numPaginas);
    }
}
```

O *this* é opcional

O *this* é opcional

Encadeando Construtores



- Em algumas situações pode ser interessante fazer com que um construtor chame outro
- O operador **this()** é utilizado nestes casos

```
class Livro
{
    public Livro(string titulo) : this(titulo, "Desconhecido")
    {
        numPaginas = 100;
    }

    public Livro(string titulo, String autor)
    {
        this.titulo = titulo;
        this.autor = autor;
    }
}
```

Chama o construtor com os parâmetros (string, string)

Construtores em Estruturas



- As mesmas regras de construtores de classes funcionam para estruturas
- Algumas diferenças
 - O compilador sempre define um construtor padrão, independente de existirem outros construtores
 - Ele inicializa os fields da estrutura com o valor padrão
 - Ele não pode ser redefinido
 - Ao definir construtores com parâmetros, é preciso inicializar todos os fields manualmente

O Modificador *static*



- O modificador *static* pode ser utilizado em diversos locais
 - Fields
 - Métodos
 - Construtores
 - Classes

Fields Estáticos

- Fields declarados como *static* pertencem à classe, e não mais a objetos da classe

```
class Numero
{
    public static int maxValor = 10000;
}
```

Não é preciso criar um objeto para acessá-lo

Field associado à classe

```
int v = Numero.maxValor;
```

O acesso ao field é feito diretamente na classe

```
Numero n = new Numero();
n.maxValor;
```

Erro de compilação

Fields Estáticos

- Um field declarado como *static* é compartilhado entre todos os objetos criados a partir da classe

```
Classe Livro
static maxPaginas = 2000
```

```
Objeto Livro maxPaginas = 2000
```

```
Objeto Livro maxPaginas = 2000
```

```
Objeto Livro maxPaginas = 2000
```

Qualquer alteração em *maxPaginas* é percebida por todos os objetos

Métodos Estáticos

- Métodos declarados como *static* pertencem à classe, e não mais a objetos da classe

```
class Numero
{
    public static void Imprimir(int n)
    {
        Console.WriteLine(n);
    }
}
```

Método associado à classe


```
Numero.Imprimir(25);
```

O acesso ao método é feito diretamente na classe

```
Numero n = new Numero();
n.Imprimir(25);
```

Erro de compilação

Métodos Estáticos



- **Console** é um exemplo de classe que possui métodos estáticos



```
Console.WriteLine("Texto");
Console.ReadLine();
```

Não existe um objeto da classe **Console**
- **Main()** também é um método estático


```
static void Main()
{
    //...
}
```


Chamado pelo CLR sem criação de objeto

Elementos Estáticos e Não Estáticos



- Uma classe pode definir fields e/ou métodos estáticos e não-estáticos ao mesmo tempo
 - Não estáticos
 - Pertencem a cada objeto da classe
 - Estáticos
 - Pertencem à classe
- Regra para acesso
 - Elementos não-estáticos podem acessar elementos estáticos
 - Elementos estáticos não podem acessar elementos não-estáticos

Construtores Estáticos



- Construtores estáticos são chamados pelo CLR na primeira vez que a classe é utilizada
- Têm o objetivo de inicializar fields estáticos da classe

```
class Livro
{
    public static int maxPaginas;

    static Livro()
    {
        maxPaginas = 10000;
    }
}
```

Construtor estático

Construtores Estáticos



- Detalhes importantes sobre o construtor estático
 - Não recebe parâmetros
 - Não pode ser sobrecarregado
 - Não possui modificador de acesso (ex: *public*)
 - Executa apenas uma vez, independentemente do número de instâncias criadas
 - Executa antes dos construtores associados à instâncias
 - É chamado pelo CLR antes de criar a primeira instância da classe ou de acessar um elemento estático pela primeira vez

Classes Estáticas



- Uma classe definida com o modificador *static* não pode ter instâncias

```
static class Numero
{
    public static double pi = 3.1416;

    public static void Imprimir(int n)
    {
        Console.WriteLine(n);
    }
}
```

A classe só pode ser *static* se contiver apenas elementos *static*

```
Numero n = new Numero();
```

Erro de compilação

Constantes



- O modificador **const** pode ser aplicado a fields também

```
class Numero
{
    public const double Pi = 3.1416;
}
```

Fields declarados como *const* são implicitamente *static*


O valor de **Pi** não pode ser alterado

```
double d = Numero.Pi;
```

O acesso é feito pela classe, não por um objeto

O valor atribuído ao field deve ser conhecido na compilação

Read-Only Fields




- Read-only fields são uma alternativa às constantes
 - O valor não precisa ser atribuído no momento da declaração
 - Uma vez definido, o valor não pode ser alterado

```
class Numero
{
    public readonly double Pi;
}
```

Cria um field não inicializado

Diferente do `const`, o modificador `readonly` não define o elemento como `static`

Read-Only Fields



- A inicialização de um read-only field só pode ser feita em um construtor


```
class Numero
{
    public readonly double Pi;

    public Numero()
    {
        Pi = 3.1416;
    }
}
```

Depois de definido, o valor não pode ser alterado

- Se o read-only field for definido como `static`, a inicialização só pode ocorrer em um construtor estático

Elementos Estáticos em Estruturas



- Estruturas também podem ter elementos `static`
 - Fields
 - Métodos
 - Construtores
- O funcionamento é igual se comparado às classes
- O `struct` em si não pode ser definido como `static`

10

