IFRN

PROGRAMAÇÃO ORIENTADA A OBJETOS EM C#

Construtores e ToString

Prof. Gilbert Azevedo

Objetivos

- Utilizar métodos construtores na escrita de classes
- Conhecer o gerenciamento de memória no C#: Stack e Heap
- Utilizar a referência “this” e o valor “null”
- Utilizar o método ToString

Construtores

- São métodos especiais que instanciam um objeto e retornam uma referência para ele
- Os construtores são invocados pelo operador new
- Toda classe tem um construtor padrão, sem parâmetros, definido automaticamente

```
class Triangulo {
  private double b, h;
  ...
}
Triangulo x = new Triangulo(); // Construtor padrão
```

Definição de Construtores

- Construtores com parâmetros podem ser escritos para iniciar os atributos do objeto com valores diferentes do valor padrão

```
class Triangulo {
  private double b, h;
  public Triangulo(double vb, double vh) {
    if (vb > 0) b = vb;
    if (vh > 0) h = vh;
  }
}
```

Construtores com Parâmetros

- Construtores com parâmetros devem necessariamente receber valores quando são chamados
- Quando um construtor é inserido na classe, o construtor padrão deixa de existir

```
class MainClass {
  public static void Main () {
    Triangulo x = new Triangulo(10, 20);
    Triangulo y = new Triangulo(); // ERRO
  }
}
```

```
class Triangulo {
  private double b, h;
  public Triangulo(double vb, double vh) {
    if (vb > 0) b = vb;
    if (vh > 0) h = vh;
  }
}
```

Construtores x Métodos de Acesso

- Construtores alocam objetos na memória; métodos de acesso, não

```
class MainClass {
  public static void Main () {
    Triangulo1 x = new Triangulo1(10, 20);
    Triangulo2 y = new Triangulo2();
    y.SetBase(10);
    y.SetAltura(20);
  }
}
```

```
class Triangulo1 {
  private double b, h;
  public Triangulo1(double vb, double vh) {
    if (vb > 0) b = vb;
    if (vh > 0) h = vh;
  }
}

class Triangulo2 {
  private double b, h;
  public void SetBase(double v) {
    if (v > 0) b = v; }
  public void SetAltura(double v) {
    if (v > 0) h = v; }
}
```

Sobrecarga de Construtores

- Uma classe pode ter vários construtores com parâmetros distintos
- E pode reincluir o construtor padrão

```
class MainClass {
  public static void Main () {
    Triangulo x = new Triangulo();
    x.SetBase(10);
    x.SetAltura(20);
    Triangulo y = new Triangulo(10);
    Triangulo z = new Triangulo(10, 20);
}
```

```
class Triangulo {
  private double b, h;
  public Triangulo() { }
  public Triangulo(double v) {
    if (v > 0) { b = v; h = v; }
  }
  public Triangulo(double vb, double vh) {
    if (vb > 0) b = vb;
    if (vh > 0) h = vh;
  }
}
```

Valor Padrão dos Atributos

- O valor padrão dos atributos é zero para números e caracteres, falso (*false*) para booleanos e nulo (*null*) para referências
- Valores diferentes do padrão podem ser informados nos atributos ou através de um construtor

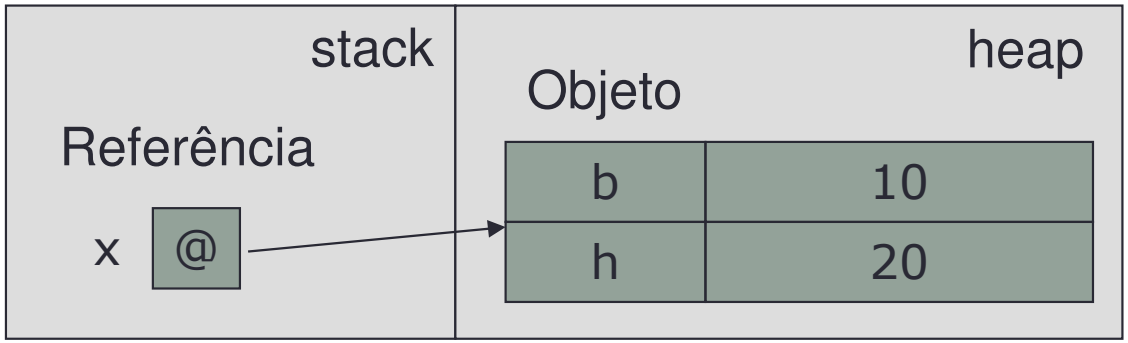
```
class Triangulo {
  private double b = 10, h = 20;
  public Triangulo() { }
}

class Triangulo {
  private double b, h;
  public Triangulo() { b = 10; h = 20; }
}
```

Memória Stack e Heap

- A memória Stack armazena as referências dos objetos
- A memória Heap armazena os objetos instanciados

```
class MainClass {
  public static void Main () {
    Triangulo x = new Triangulo(10, 20);
  }
}
```

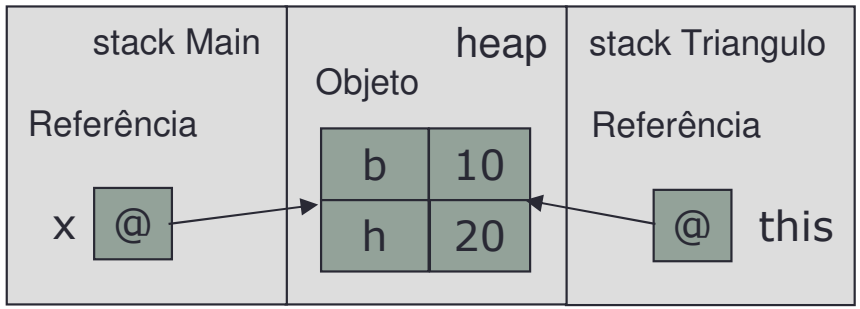
```
class Triangulo {
  private double b, h;
  public Triangulo(double vb, double vh) {
    if (vb > 0) b = vb;
    if (vh > 0) h = vh;
  }
}
```



Referência “this”

- “this” é uma referência especial para um objeto quando um método é invocado, sendo usada normalmente quando atributos e parâmetros possuem o mesmo identificador

```
class MainClass {
  public static void Main () {
    Triangulo x = new Triangulo(10, 20);
    x.SetBase(10);
    x.SetAltura(20);
  }
}
```

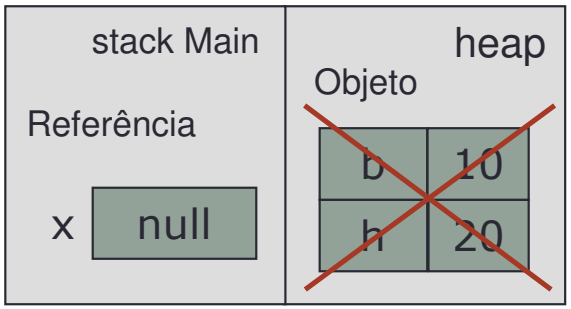



```
class Triangulo {
  private double b, h;
  public Triangulo(double b, double h) {
    if (b > 0) this.b = b;
    if (h > 0) this.h = h;
  }
  public void SetBase(double b) {
    if (b > 0) this.b = b; }
  public void SetAltura(double h) {
    if (h > 0) this.h = h; }
}
```

O valor “null”

- “null” é o valor vazio para referências de quaisquer classes
- Objetos não referenciados são removidos da memória pelo “coletor de lixo” – Garbage Collection

```
class MainClass {
  public static void Main () {
    Triangulo x = new Triangulo(10, 20);
    x.SetBase(10);
    x.SetAltura(20);
    x = null;
  }
}
```




```
class Triangulo {
  private double b, h;
  public Triangulo(double b, double h) {
    if (b > 0) this.b = b;
    if (h > 0) this.h = h;
  }
  public void SetBase(double b) {
    if (b > 0) this.b = b; }
  public void SetAltura(double h) {
    if (h > 0) this.h = h; }
}
```

Método ToString

- O método ToString é normalmente usado para retornar um texto com os dados do objeto
- A implementação padrão deste método informa a classe do objeto

```
class MainClass {
  public static void Main () {
    Triangulo x = new Triangulo(10, 20);
    Console.WriteLine(x);
    Console.WriteLine(x.ToString());
  }
}
```

```
class Triangulo {
  private double b, h;
  public Triangulo(double b, double h) {
    if (b > 0) this.b = b;
    if (h > 0) this.h = h;
  }
}
```

- A informação apresentada nesse caso é: Triangulo

Sobrescrita do Método ToString

- O método pode ser sobrescrito para mostrar alguns dados mais específicos sobre o objeto
- Métodos sobrescritos são assinados com *override*

```
class MainClass {
  public static void Main () {
    Triangulo x = new Triangulo(10, 20);
    Console.WriteLine(x);
    Console.WriteLine(x.ToString());
  }
}
```

- A saída agora é:
  - Base = 10, Altura = 20

```
class Triangulo {
  private double b, h;
  public Triangulo(double b, double h) {
    if (b > 0) this.b = b;
    if (h > 0) this.h = h;
  }
  public override string ToString() {
    return $"Base = {b}, Altura = {h}";
  }
}
```

Modelo Final do Triangulo

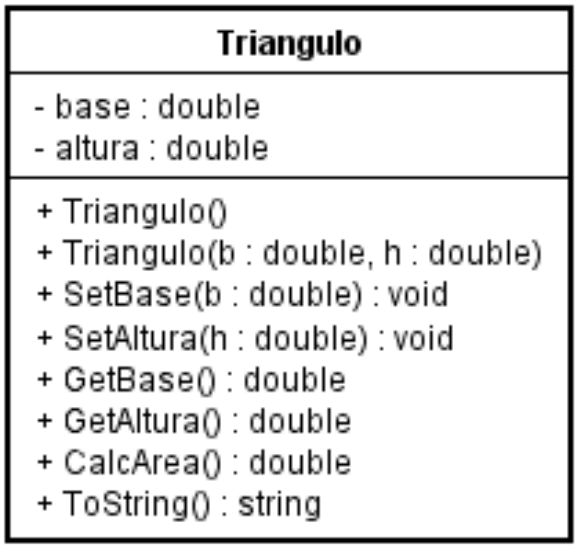
- Classe Triangulo com construtores, métodos de acesso e ToString

```
class MainClass {
  public static void Main () {
    Triangulo x = new Triangulo();
    x.SetBase(10);
    x.SetAltura(20);
    Triangulo y = new Triangulo(10, 20);
    Console.WriteLine(x);
    Console.WriteLine(y.ToString());
  }
}
```

```
class Triangulo {
  private double b, h;
  public Triangulo() { }
  public Triangulo(double b, double h) {
    if (b > 0) this.b = b;
    if (h > 0) this.h = h;
  }
  public void SetBase(double b) {
    if (b > 0) this.b = b; }
  public void SetAltura(double h) {
    if (h > 0) this.h = h; }
  public override string ToString() {
    return $"Base = {b}, Altura = {h}";
  }
}
```

Modelo Final do Triangulo

- Diagrama da classe Triangulo. Métodos Get e CalcArea não estão listados no código.




```
class Triangulo {
  private double b, h;
  public Triangulo() { }
  public Triangulo(double b, double h) {
    if (b > 0) this.b = b;
    if (h > 0) this.h = h;
  }
  public void SetBase(double b) {
    if (b > 0) this.b = b; }
  public void SetAltura(double h) {
    if (h > 0) this.h = h; }
  public override string ToString() {
    return $"Base = {b}, Altura = {h}";
  }
}
```

Referências

- Microsoft Visual C# 2010 – Passo a passo, John Sharp, Bookman, 2010
- Classes (Guia de Programação em C#)
  - https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/classes-and-structs/classes