IFRN

PROGRAMAÇÃO ORIENTADA A OBJETOS EM C#

Valores e Referências

Prof. Gilbert Azevedo

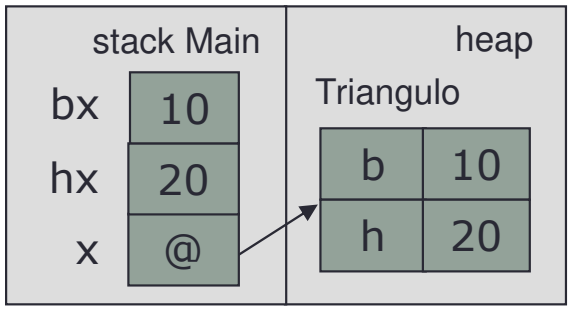
Objetivos

- Conhecer a organização da memória no .NET
- Entender as diferenças entre os tipos por Valor e por Referência
- Utilizar o tipo Object
- Conhecer as operações de Boxing e Unboxing

Organização da Memória

- Variáveis no C# são armazenadas nas memórias *stack* e *heap*
  - Variáveis locais e parâmetros ficam na memória *stack*: bx, hx e x
  - Instâncias de classes ficam na *heap:* objeto Triangulo

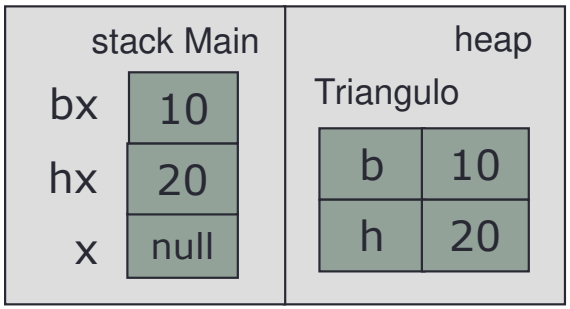
```
class MainClass {
  public static void Main () {
    double bx = 10, hx = 20;
    Triangulo x = new Triangulo(bx, hx);
  }
}
class Triangulo {
  private double b, h;
  public Triangulo(double vb, double vh) {
    if (vb > 0) b = vb; if (vh > 0) h = vh;
  }
}
```



Organização da Memória

- Variáveis na memória *stack* são removidas quando o método encerra
  - bx, hx e x desaparecem quando Main acaba
- Variáveis na *heap* são removidas quando não têm mais referência
  - O coletor de lixo remove o objeto Triangulo quando a referência x recebe o valor *null*

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



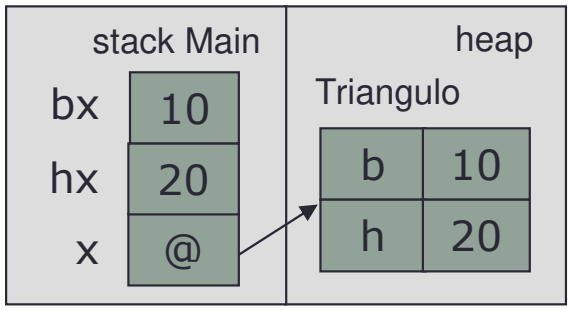
Tipos de Objetos

- O C# possui dois tipos de dados: tipos por valor e tipos por referência
- Tipos por Valor
  - A variável contém uma instância do tipo
  - Estruturas – struct: int, double, char, bool, DateTime, TimeSpan
  - Enumerações – enum
- Tipos por Referência
  - A variável é uma referência para uma instância do tipo
  - class, interface, delegate
- Tipos por Referência Internos
  - dynamic, object, string

Tipos por Valor e por Referência

- Tipos por valor (Value Types)
  - As variáveis são alocadas na memória *stack – (objetos double bx, hx)*
  - Devem ser iniciados antes de utilizados e não recebem *null*
  - Atributos (*b* e *h*) ficam sempre na memória Heap, mas armazenam o valor e não uma referência

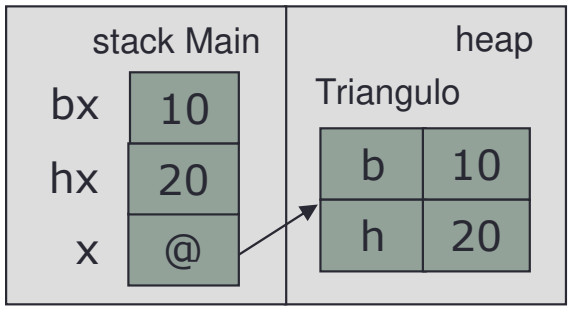
```
class MainClass {
  public static void Main () {
    double bx = 10, hx = 20;
    Triangulo x = new Triangulo(bx, hx);
    x = null;
  }
}
```



Tipos por Valor e por Referência

- Tipos por referência (Reference Types)
  - Os dados são alocados na memória *heap – (objeto Triangulo)*
  - São iniciados com o operador *new e* destruídos pelo coletor de lixo
  - São controlados por uma referência – (x)
  - Um atributo também pode armazenar uma referência

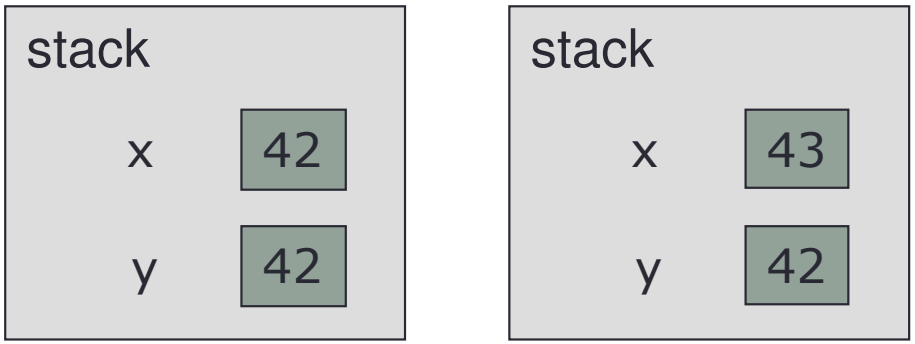
```
class MainClass {
  public static void Main () {
    double bx = 10, hx = 20;
    Triangulo x = new Triangulo(bx, hx);
    x = null;
  }
}
```



Atribuição em Tipos por Valor

- Na atribuição, os objetos são copiados na memória (duplicados)
  - Quando o objeto é alterado o outro permanece com o valor inicial

```
int x = 42;
int y = x;
Console.WriteLine($"{x} {y}");
x = 43;
Console.WriteLine($"{x} {y}");
```

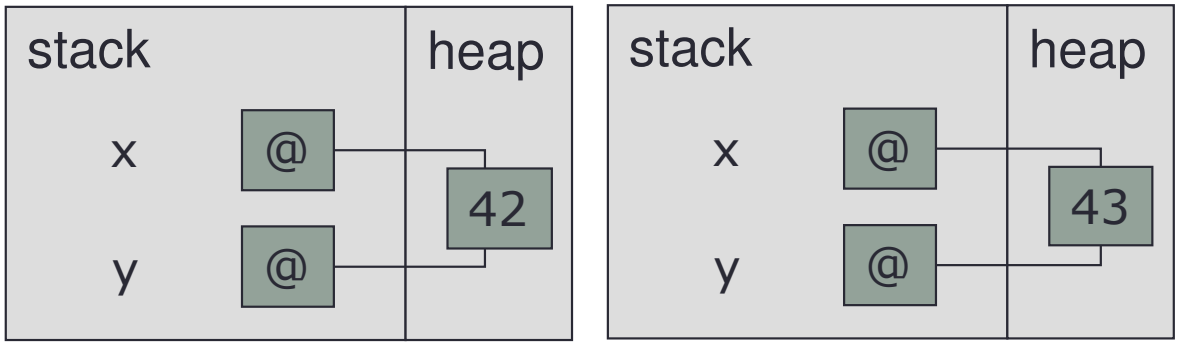


Atribuição em Tipos por Referência

- Na atribuição, as referências são copiadas na memória, o objeto não

```
Inteiro x = new Inteiro();
Inteiro y = x;
x.Num = 42;
Console.WriteLine($"{x.Num} {y.Num}");
x.Num = 43;
Console.WriteLine($"{x.Num} {y.Num}");
```

```
class Inteiro {
  public int Num;
}
```



Resumo de Tipos por Valor

- São tipos por valor
  - Inteiros, reais, caracteres, booleanos (structs)
  - Estruturas (struct)
  - Enumerações (enum)
- Características
  - São descendentes da classe *System.ValueType*
  - Quando uma variável (ou atributo) é declarada, um bloco de memória grande o suficiente para conter seu valor é alocado
  - Quando uma variável (ou atributo) é atribuída a outra, os valores ficam duplicados na memória

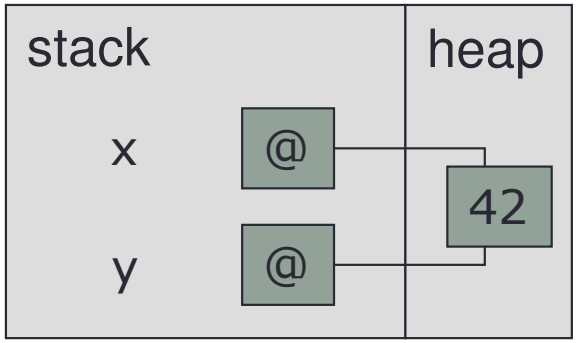
Resumo de Tipos por Referência

- São tipos por referência
  - Todos os tipos definidos com *class, interface* e *delegate*
  - *dynamic* é um tipo usado com funções do sistema operacional
  - *string* é um tipo por referência especial, pois dispensa o uso do *new*
  - *object* permite referenciar tipos por valor e por referência
- Características
  - Quando uma variável é declarada, apenas uma referência é alocada na memória
  - O objeto só é alocado na memória quando o operador *new* é executado
  - Quando uma variável é atribuída a outra, as referências ficam duplicadas na memória e não os objetos

Tipo Object

- Variáveis do tipo *object* podem referenciar todos os tipos de objetos
- Quando um *object* referencia um tipo por referência, ele armazena o endereço do objeto que é alocado na memória *heap*

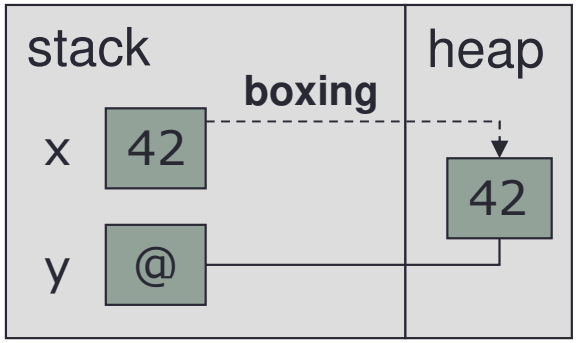
```
Inteiro x = new Inteiro();
x.Num = 42;
object y;
y = x;
```



Boxing

- Quando um *object* referencia um tipo por valor, uma operação *Boxing* é realizada
- A operação realiza uma cópia do tipo por valor para a memória *heap* e a variável *object* armazena o endereço do valor copiado

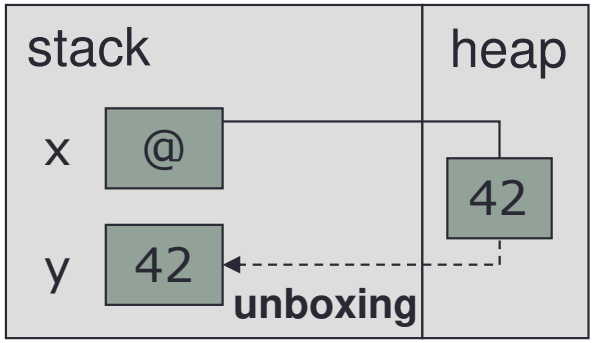
```
int x = 42;
object y = x;
Console.Writeline(y);
```



Unboxing

- Para acessar novamente o valor *boxed* na memória *heap* como um tipo por valor, é necessário fazer uma operação de *unboxing*
- A operação *unboxing* deve ser realizada com um *cast*

```
object x = 42;
int y = (int) x;
y++;
Console.Writeline(y);
```



Referências

- Microsoft Visual C# 2010 – Passo a passo, John Sharp, Bookman, 2010
- Tipos de Valor
  - https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/builtin-types/value-types
- Tipos de referência
  - https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/keywords/reference-types
- Conversões Boxing e Unboxing (C# Programming Guide)
  - https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/types/boxing-and-unboxing