# ISEL

# Ambientes Virtuais de Execução

2021

Week 7 – Instances and Methods

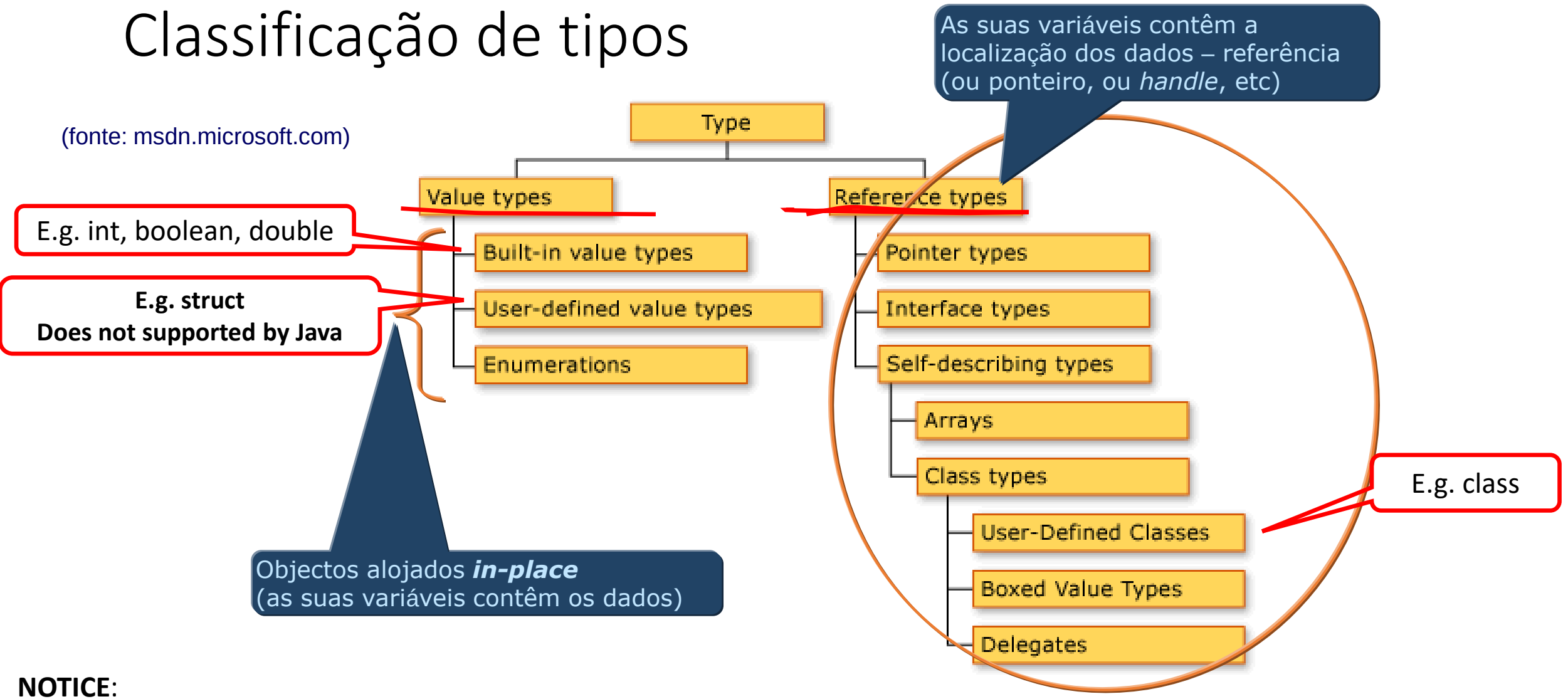
Ref: Essential .Net

Chapter 5 – Instances

Chapter 6 – Methods

# Classificação de tipos

(fonte: msdn.microsoft.com)




**NOTICE**:

- **primitive types** may be both **Value** or **Reference** Types: **int**, **string**, **object** are primitive types (known by the Compiler)
- **NON primitive types** may be both built-in **Value** or **Reference** Types: System.Int32 or System.String

Store local variables

Armazena a instancia **in-place**

Armazena a **referencia** para a instancia

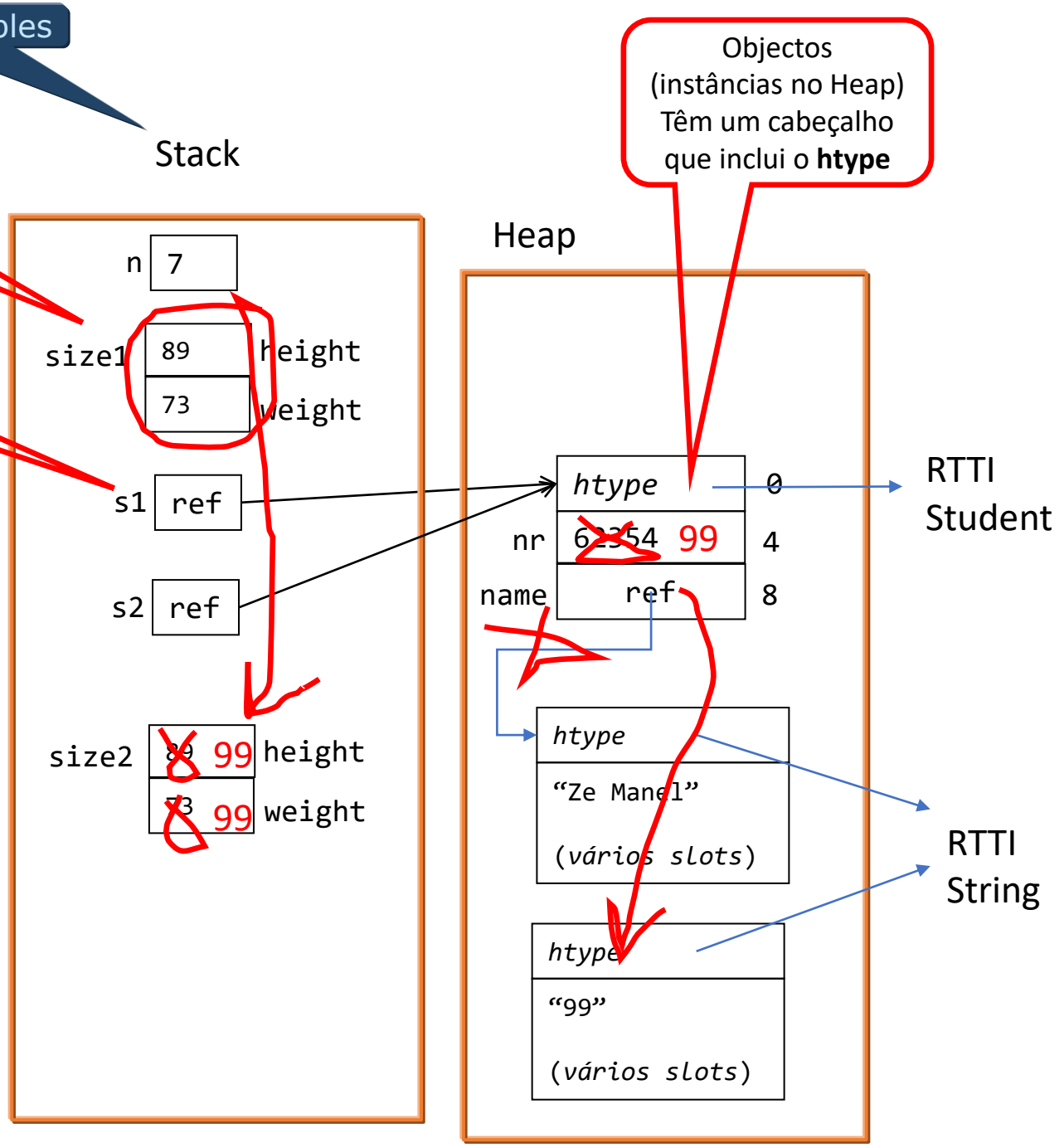
```
public class Student {
  private string name;
  private int nr;
}
```

```
public struct Size{
  public int height;
  public int weight;
}
```

```
int n = 7;
Size size1 = new Size(89, 73);
Student s1 = new Student(62354, "Ze Manel");

Student s2 = s1;
s2.nr = 99;     // !!! s2 and s1 have the same Reference
s2.name = "99";

Size size2 = size1;
size2.height = 99;  // size2 stores other instance
size2.weight = 99;
```



# Objects have a header with htype

- htype = type handle (*ponteiro para o tipo*)
- htype points to RTTI (*runtime type information*) (informação do tipo):
  - Specifies the type of the object
  - NOTICE that is not the same as the class `System.Reflection.Type`
- Objects of the same Type have htype with the same reference.

“*ter varias structs é mais pesado do que varias referencias*” ????

-- passagem de parametros => cópia integral de toda a instancia
E.g. copiar todas as palavras (slots) em vez de apenas a referencia.

++ Acesso mais eficiente aos campos
=> menos uma indirecção

++ As intancias de tipo valor (de struct) não são processadas pelo GC.

++ Tempo de vida

# Life Cycle

??? What is the life cycle of each instance ???

??? When the VM clear those instances???

```
static void Foo()
{

    WrapInt w = new WrapInt();
    int n = 7;

    Student s1 = new Student(62354, "Ze Manel");
    Size size1 = new Size(33, 73);

    Student s2 = s1;
    s2.nr = 99;
    s2.name = "99";

    Size size2 = size1;
    size2.height = 99;
    size2.weight = 99;
}
```

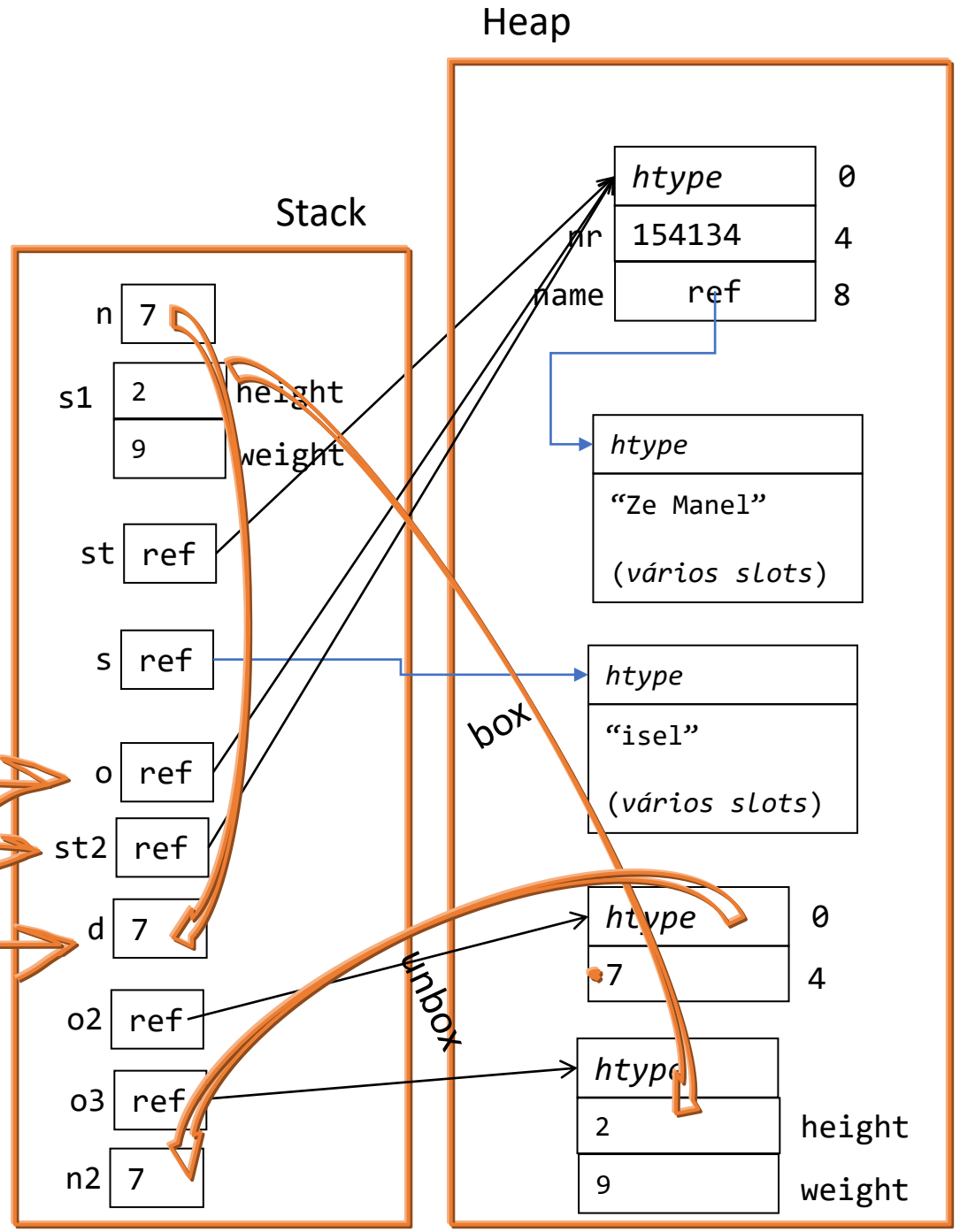
These objects remain in the Heap after completion of Foo() until the GC clear them.

# Conversions

- Reference Type <-> Reference Type --- casting (on references)
  - upcasting (implicit in C#) e.g. `string str = …; object o = str; // ldloc.0; stloc.1;`
  - downcasting e.g. `string str2 = (string) o; // ldloc.1; castclass String; stloc.2`
- Value Type built-in <-> Value Type built-in --- coercion (on values)
  - E.g. `double d = …; int n = (int) d; // ldloc.0; conv.i4; stloc.1;`
  - `IL conv...`
- Value Type user-defined <-> Value Type user-defined
  - Because structs cannot inherit from other structs
- Reference Type -> Value Type --- unboxing (IL: unbox.any)
- Value Type -> Reference Type --- **boxing**
  - NOTICE the overhead ⇔ **newobj** (IL operation to instantiate a class)

# Conversions

```
/*
* Value Types
*/
int n = 7; // Value type built-in (primitive)
Size s1 = new Size(2, 9); // Value Type user defined
/*
* Ref Types
*/
string s = "isel";           // Reference type
Student st = new Student("Ze Manel", 13876);
Object o = st;
Student st2 = (Student) o;
/*
 * Coercion
 */
double d = n; // Coercion: ldloc.0; conv.r8; stloc.6;
/*
 * Boxing and unboxing
 */
Object o2 = n;  // ldloc.0; box Int32; stloc.7;
Object o3 = s1; // ldloc.1; box Size; stloc.8;
int n2 = (int) o2; // ldloc.7; unbox.any; stloc.9
```

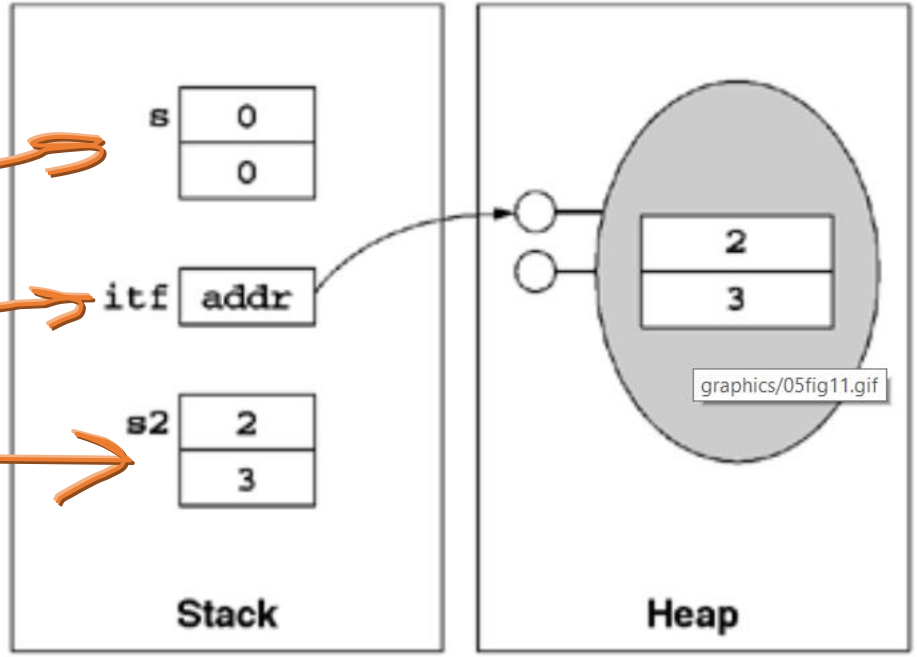


# Instances

Essential .Net (Don Box)

```
public interface IAdjustor {
  void Adjust();
}
public struct Size : IAdjustor {
  public void Adjust() { height+=2; weight+=3; }
  public int height;
  public int weight;
}
static App {
  static void Main() {
    Size s = new Size();
    IAdjustor itf = s; // box
    itf.Adjust();      // operate on boxed copy
    s = (Size)itf;     // unbox
  }
}
```

**Figure 5.11. Boxing and Unboxing**

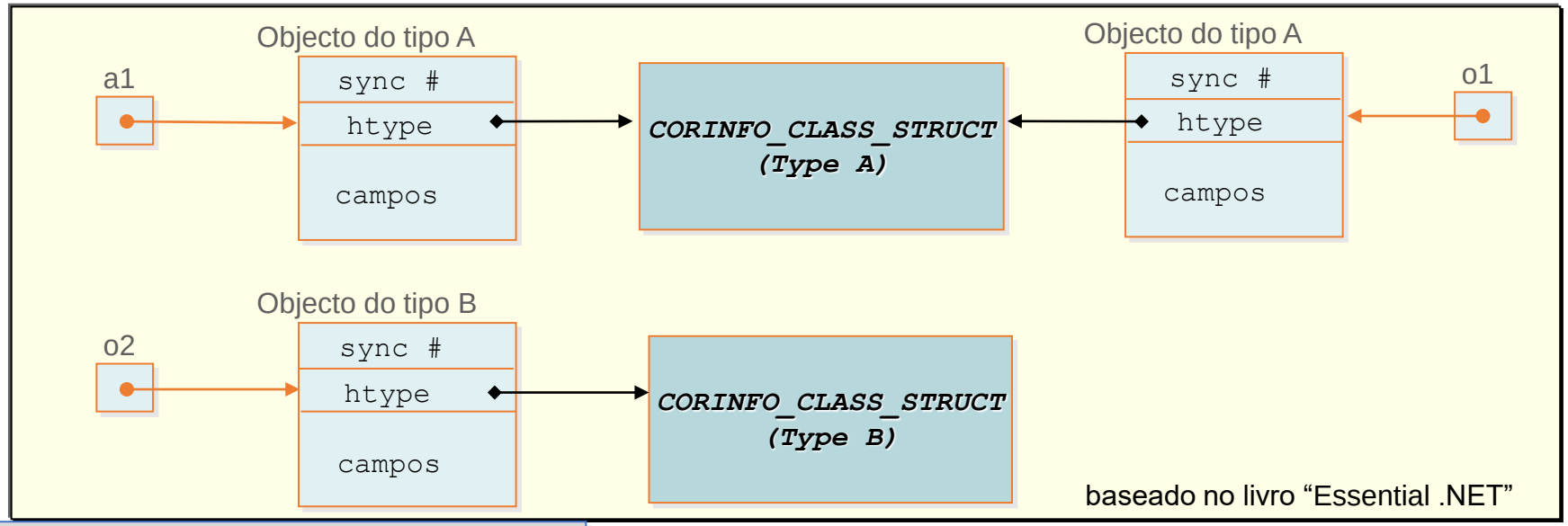



```
// create value type
size s = new Size ();
// box and use
IAdjustor itf = s;
itf.Adjust();
// unbox into s2
Size s2 = (Size)itf;
```

**!!!!! Notice the different result if Size was a class instead !!!!!**

# Informação de tipo em tempo de execução (RTTI)

A cada objecto é associado um cabeçalho (*object header*: sync# e htype) que descreve o tipo do qual ele é instância.




```
class A{}
class B{}
A a1 = new A();
Object o1 = new A();
Object o2 = new B();
```

```
class App {
 static void InstancesOfSameClass(Object o1, Object o2) {
  if(o1.GetType() == o2.GetType())
    Console.WriteLine("Instances of same class.");
  else
    Console.WriteLine("Instances of different classes.");
 }
 static void Main() {
  ...
  InstancesOfSameClass(a1, o1);
  InstancesOfSameClass(o1, o2);
 } }
```

```
> App.exe
Instances of same class.
Instances of different classes.
```

# Instances at runtime

```
public class Person {}
public struct Point {}

class Program
{
    static void Main(string[] args)
    {
        Person p = new Person();
        Point pt = new Point();
    }
}
```

Person is a RT => **instantiated** => stored in Heap =>IL newobj

```
newobj      instance void aula25_methods.Person::.ctor()
stloc.0
```

**newobj**

1. **Alocação de espaço no Heap**
2. **Inicializar o espaço a zeros + Header**
3. **Chamada ao contrutor**

- Retorna uma referencia para a nova instancia

**=> GC !!! overhead**

Point is a VT => **initialized**=> stored in Stack => IL initobj

```
ldloca.s    V_1
initobj     aula25_methods.Point
```

**initobj**

- **Inicializar** um espaço (neste caso no Stack) a zeros
- Recebe como parametro o endereço desse espaço

**=> Epilogo do Método**

**Inicializar != Alocar (i.e. `malloc`)**

Afectar 0 — Reserva de espaço em memória

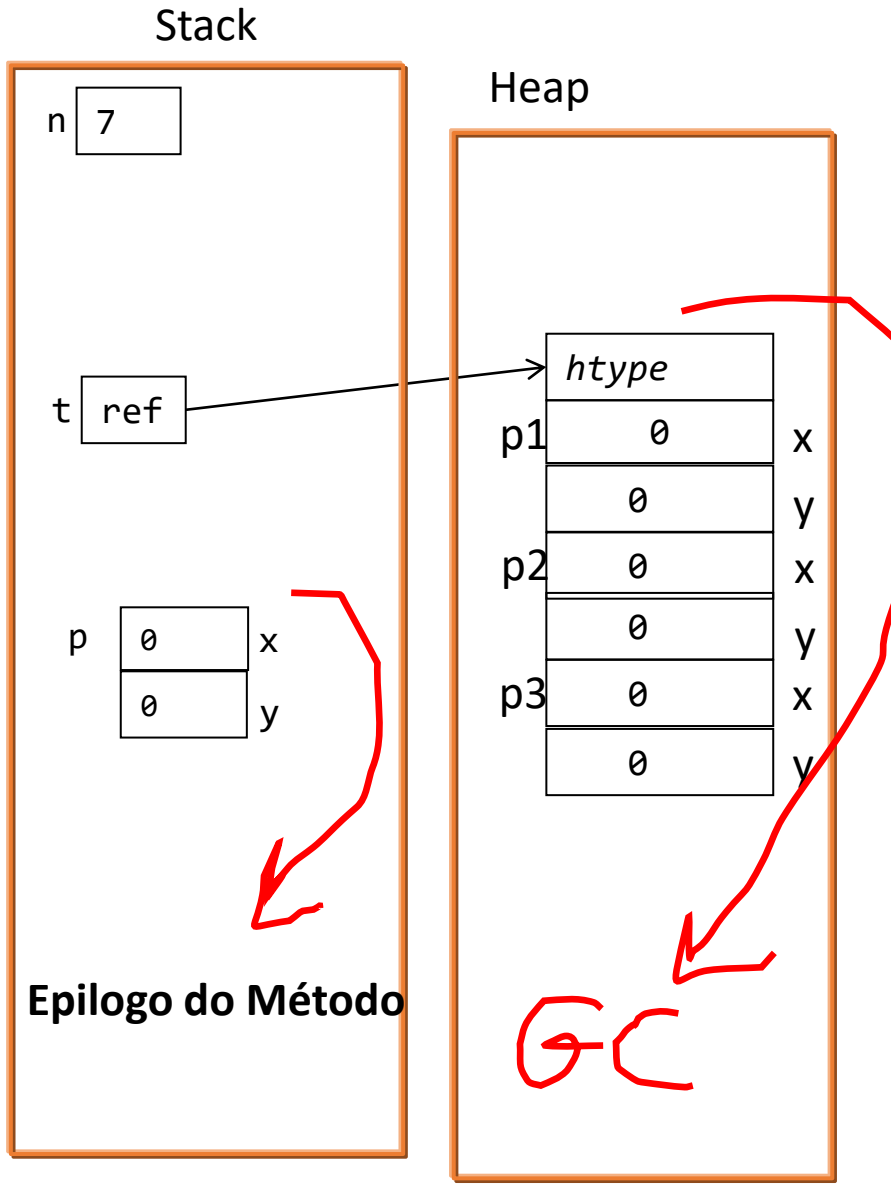
!!!!! Overhead !!!!!

# Value Types are stored in-place

```
public class Person {}
public struct Point { int x; int y;}

class Triangle {
    Point p1, p2, p3; // Stored in-place => Heap
}

class Program
{
    static void Main(string[] args)
    {
        Person p = new Person(); // stored in Heap
        Point pt = new Point();  // stored in Stack
        Triangle t = new Triangle(); // stored in Heap
    }
}
```



# newobj Tasks

**1. Allocates storage on Heap**

**2. Initializes space with zeros + Header (htype pointing to RTTI)**

**3. Calls the constructor**

- Returns the reference of newbie instance

# Initializing Value Types

```
public struct Point { int x; int y;}

Point pt = new Point();
```

```
ldloca.s    V_1
initobj     aula25_methods.Point
```

```
public struct Point {
  int x; int y;
  public Point(int x, int y)
  {
    this.x = x;
    this.y = y;
  }
}
Point pt = new Point();
Point pt2 = new Point(5, 7);
```

```
ldloca.s    V_1
initobj     aula25_methods.Point
```

**Como inicializar Point chamando o constructor?**

```
ldloca.s    V_2
ldc.i4.5
ldc.i4.7
call        void Point::.ctor(int32, int32)
```

# Calling Methods

It does not require a verification of non null target!

| | Methods | | | | |
|---|---|---|---|---|---|
| Types | | Ctor | Static | Instance (***non virtual***) | Virtual (*instance*) |
| | Value Types | `call` | `call` | `call` | NA |
| | Ref. Types | *Implicit in* `newobj` | `call` | `callvirt` | `callvirt` |

**callvirt** has an implicit validation to **check if the target reference (i.e. this) is not null!**