Ambientes Virtuais de Execução

Estrutura de Tipos
(continuação)

Tabela de métodos

- Quando um tipo armazenado no CLR, uma tabela de métodos é inicializada para o tipo.
  - A tabela de métodos no CLR tem entradas para métodos de instância e estáticos.
    - A primeira região é usada para os métodos virtuais (declarados no tipo actual, nos tipos base ou interfaces)
      - *Slots* iniciais irão corresponder aos métodos virtuais declarados pelos tipo base;
      - *Slots* seguintes irão corresponder aos novos métodos virtuais introduzidos pelo tipo;
    - Exemplo:
      - Como `System.Object` é o tipo base de todos os tipos concretos e tem 4 métodos virtuais, os primeiros slots de todas as tabelas de métodos correspondem a esses 4 métodos
    - A segunda região é usada para métodos não virtuais

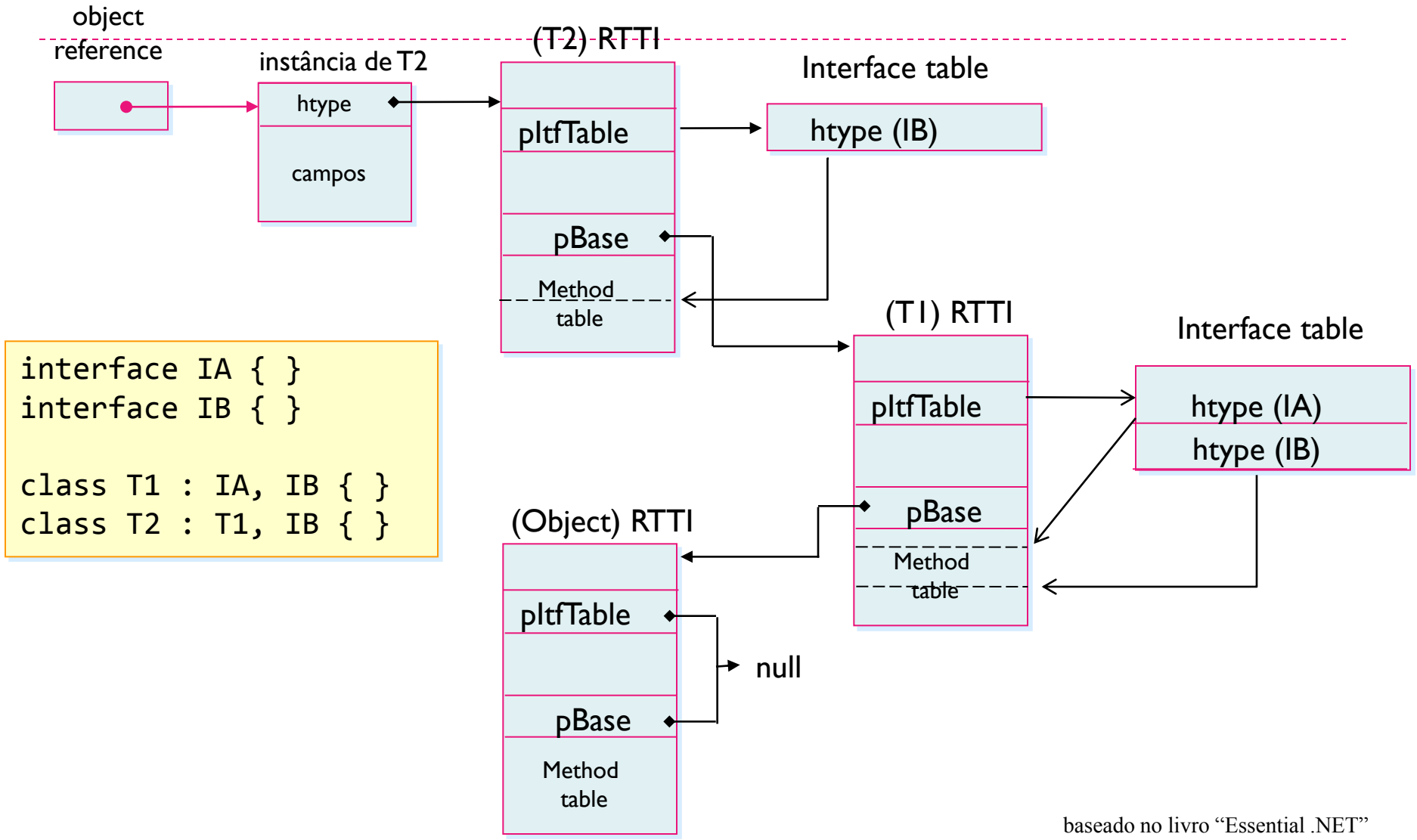
Atributos de metadata e métodos virtuais

Atributos de metadata	Presente	Ausente
virtual	O índice na tabela de métodos está na região dos métodos virtuais.	O índice na tabela de métodos está na região dos métodos não virtuais.
newslot	Aloca um novo índice na tabela de métodos	Re-utiliza o índice do método do tipo base, se possível.
abstract	Requere substituição no tipo derivado.	Permite substituição no tipo derivado
final	Proíbe subtituição no tipo derivado.	Permite substituição no tipo derivado

Tabela de interfaces

- Quando um tipo armazenado no CLR, uma tabela de interfaces é inicializada para o tipo.
  - A tabela de interfaces tem entradas para cada interface que o tipo é compatível

Informação de tipo em tempo de execução (RTTI)



baseado no livro "Essential .NET"

Arrays

- Todos os tipos de arrays derivam implicitamente da classe abstracta `System.Array`.
- Exemplos:
  - `Int32[ ] myIntegers = new Int32[100];`
  - `Point[ ] myPoints = new Point[10];`
  - `Double[ , ] myDoubles = Double[10,20];`
  - `int[] numbers = new int[5] {1, 2, 3, 4, 5};`
  - `int[] numbers = new int[] {1, 2, 3, 4, 5};`
  - `int[] numbers = {1, 2, 3, 4, 5};`
  - `int[,] numbers = { {1, 2}, {3, 4}, {5, 6} };`
  - `//jagged arrays (arrays de arrays)`
  - `Point[ ][ ] myPoligon = new Point[3][ ];`
  - `myPoligon[0] = new Point[10];`
  - `myPoligon[1] = new Point[20];`
  - `myPoligon[2] = new Point[30];`

Conversão de Arrays

- CLR permite a conversão do tipo dos elementos do array origem para o tipo pretendido
  - Ambos os arrays têm de ter as mesmas dimensões
  - Tem de existir uma conversão implícita ou explícita do tipo do elemento do array de origem para o tipo do elemento do array destino
    - `FileStream[ ] s1 = new FileStream[10];`
    - `Object[ ] o1= s1;`
    - `FileStream[ ] s2 = (FileStream[ ]) o1;`
- A conversão dos arrays de um tipo para o outro designa-se por covariância.

covariância em arrays

Seja o seguinte código:

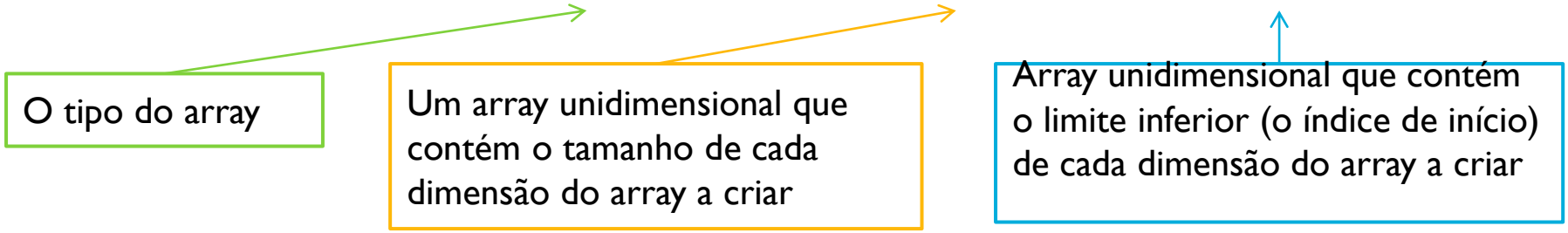
```
string[] tabStrings = new string[] { "str1", "str2", "str3" };
Console.WriteLine("Size={0}", tabStrings.Length);
foreach( string str in tabStrings )
  Console.WriteLine(str);

// E o código seguinte?
object[] tabObjects = tabStrings;  // covariância em arrays
tabObjects[2]= new object();       // problemas?
```

Arrays com limite inferior diferente de zero

- É possível criar este tipo de arrays recorrendo à método:

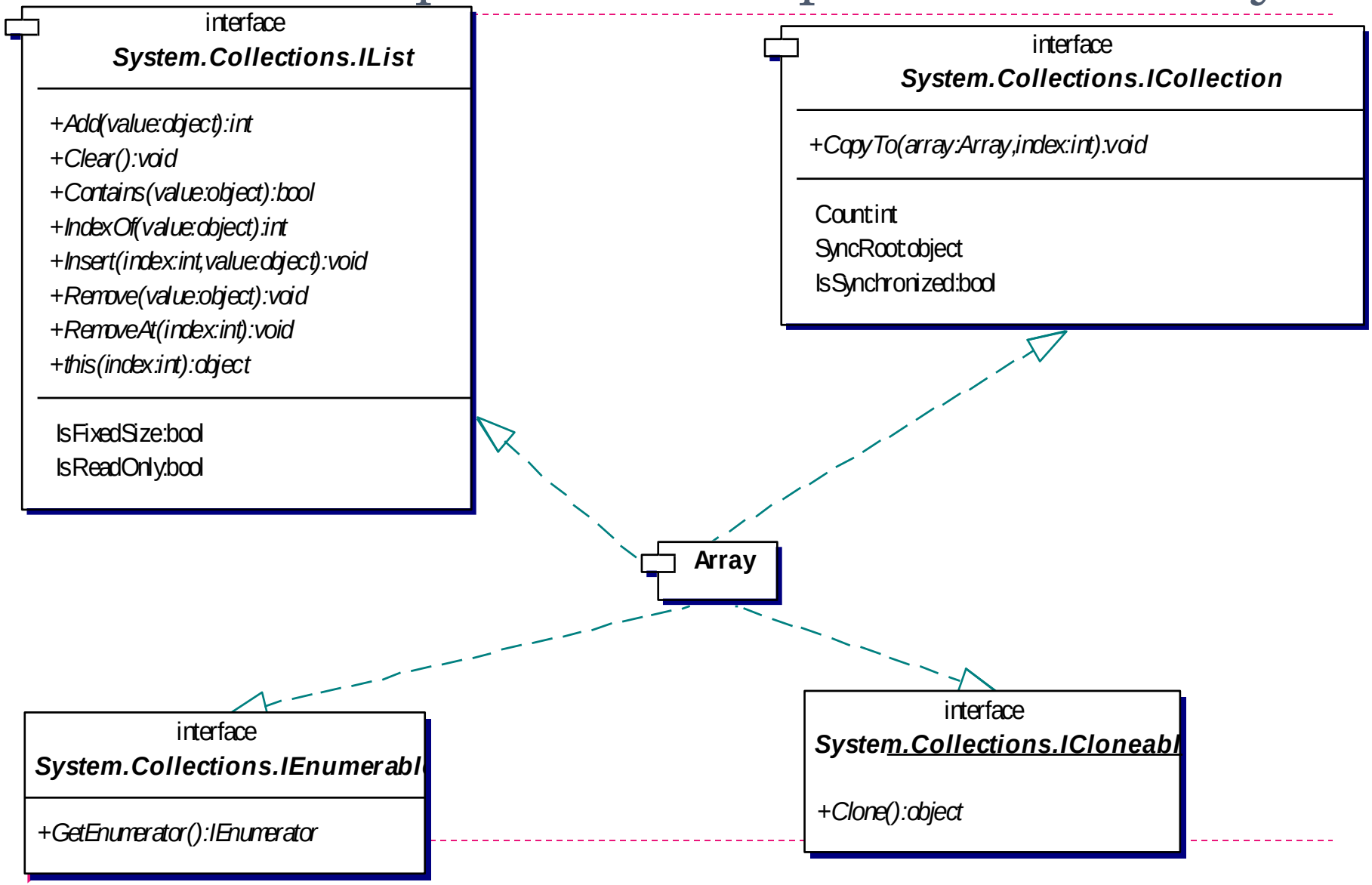
```
public static
 Array CreateInstance(Type elementType, int[] lengths, int[] lowerBounds)
```



Exemplos:

```
Array a = Array.CreateInstance(typeof(String),new Int32[] {0} , new Int32[] {1})
Array b;
 b= Array.CreateInstance(typeof(String),new Int32[] {0,0} , new Int32[] {1,1})
```

Interfaces implementadas pela classe Array



O tipo Array - propriedades

Para além das propriedades relacionadas com a implementação das interfaces `IList, IEnumerable e IClonable`

Length	Gets a 32-bit integer that represents the total number of elements in all the dimensions of the **Array**.
LongLength	Gets a 64-bit integer that represents the total number of elements in all the dimensions of the **Array**.
Rank	Gets the rank (number of dimensions) of the **Array**.

O tipo Array – métodos publicos (I)

Para além dos métodos relacionados com a implementação das interfaces `IList, IEnumerable e IClonable` e dos definidos em object

static int **BinarySearch(**Array, object)	Overloaded. Searches a one-dimensional sorted **Array** for a value, using a binary search algorithm.
static void **Copy**(Array, Array, int)	Overloaded. Copies a section of one **Array** to another **Array** and performs type casting and boxing as required.
static Array **CreateInstance**(Type, int)	Overloaded. Initializes a new instance of the **Array** class.
int **GetLength**(int dimension)	Gets a 32-bit integer that represents the number of elements in the specified dimension of the **Array**.
int **GetLowerBound**(int dimension)	Gets the lower bound of the specified dimension in the **Array**.

O tipo Array – métodos publicos (II)

public int GetUpperBound(int dimension)	Gets the upper bound of the specified dimension in the **Array**.
public object GetValue(int)	Overloaded. Gets the value of the specified element in the current **Array**. The indexes are specified as an array of 32-bit integers.
public void Initialize()	Initializes every element of the value-type **Array** by calling the default constructor of the value type.
public static void Reverse(Array)	Overloaded. Reverses the order of the elements in a one-dimensional **Array** or in a portion of the **Array**.
public void SetValue(object, int)	Overloaded. Sets the specified element in the current **Array** to the specified value.
public static void Sort(Array)	Overloaded. Sorts the elements in one-dimensional **Array** objects.

Acesso a array não seguro

- O acesso a array não seguro permite aceder
  - A elementos de um objecto array managed (alojado no heap)
  - A elementos de um array que está alojado no unmanaged heap
  - A elementos de um array que está alojado na stack
- Para alojar um array na stack utiliza-se a instrução stackalloc
  - Apenas se podem criar arrays unidimensionais, cujo tipo de elementos seja tipo valor, com limite inferior zero (zero-based arrays)
    - O tipo valor poderá conter campos de tipo referência
  - É necessário especificar o swich /unsafe ao compilador de C#

Acesso a array não seguro – Exemplo 1

```
//. . .
public static void Main(){ StackallocDemo(); }

private static void StackallocDemo( ){
  unsafe{
     const Int32 width = 20 ;
     Char * pc = stackalloc Char [ width ];
     String s =“ Ola Mundo “;
     for(Int32 index =  0; index <width; index++){
       pc[width – index – 1] = (index <s.Length) ? s[index] : ‘.’;
   }
   Console.WriteLine(new String(pc,0,width));
 }
//. . .
```

Acesso a array não seguro – Exemplo 2

```
//. . .
public static void Main{  InlineArrayDemo(); }
internal unsafe struct CharArray {
  public fixed Char Characters[ 20 ]; }

private static void InlineArrayDemo( ){
  unsafe{
     CharArray ca;
     const Int32 width = 20 ;
     String s =“Ola Mundo“;
     for(Int32 index =  0; index <width; index++){
      ca.Characters[width – index – 1] = (index <s.Length) ? s[index] : ‘.’;
   }
   Console.WriteLine(new String(ca.Characters,0,width));
 }
//. . .
```