Tecniche di Programmazione avanzata

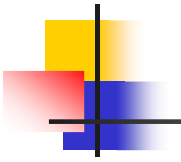
Corso di Laurea Specialistica in Ingegneria Telematica
Università Kore – Enna – A.A. 2009-2010

Alessandro Longheu
http://www.diit.unict.it/users/alongheu
alessandro.longheu@diit.unict.it

Il linguaggio C#
Array e Stringhe

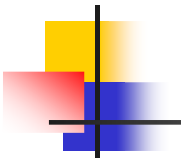
Arrays

- C# fornisce tre tipi di array:
  - Single dimensional
  - Multidimensional
  - Jagged
- Arrays possono essere specificati come in/out
- Un metodo può utilizzare il contenuto di un array
- Arrays derivano da System.Array class



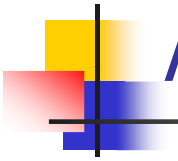
Array

- L'array è un contenitore di dati (elementi) omogenei
- Gli array sono di tipo riferimento e la memoria è allocata nell'heap (managed)
- Gli array sono immutabili
- MSIL offre istruzioni native per la gestione degli array monodimensionali.



Array monodimensionali

```
int [] a;
int[] a = new int[3];
int[] b = new int[3] {3, 4, 5};
int[] b = new int[] {3, 4, 5};
int[] c = {3, 4, 5};
SomeClass[] d = new SomeClass[10];
SomeClass[] d = new SomeClass[10]{
    new SomeClass(),
    new SomeClass(),
    new SomeClass()};
```



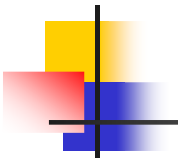
Array initialization

- Gli array vuoti sono creati nel seguente modo:

```
int[] a = new int[10];
```

- Sono inizializzati con "empty values" (per i tipi value) o null (per i tipi riferimento)
- Inizializzazione array

```
string[] a = {"a","b","c"};
a = new string{"a","b","d"};
```

Array multidimensionali

Multidimensional arrays (jagged)

int[][] a = new int[2][]; // array di riferimenti ad un altro array
a[0] = new int[] {1, 2, 3}; // non possono essere inizializzati
a[1] = new int[] {4, 5, 6};
In questo caso la dimensione dell'array non è definita perchè le righe dell'array possono contenere array di differenti dimensioni

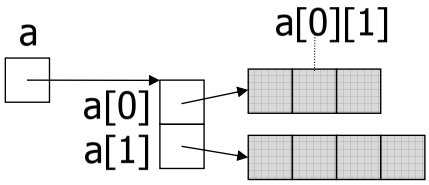
Multidimensional arrays (rectangular)

int[,] a = new int[2, 3]; // le matrici possono essere inizializzate
int[,] b = {{1, 2, 3}, {4, 5, 6}};
int[,,] c = new int[2, 4, 2];

Array multidimensionali

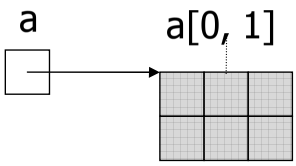
Jagged (come Java, migliore gestione di memoria)

```
int[][] a = new int[2][];
a[0] = new int[3];
a[1] = new int[4];

int x = a[0][1];
```



Rettangolari (più compatti ed efficienti nell'accesso – come in fortran)

```
int[,] a = new int[2, 3];

int x = a[0, 1];
```



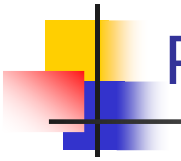
Proprietà degli array

System.Array esporta metodi utili a manipolare gli array

```
int[] a = {7, 2, 5};
int[] b = new int[2];
Array.Copy(a, b, 2);// copies a[0..1] to b
Array.Sort(b);
...
```

Array length

```
int[] a = new int[3];
Console.WriteLine(a.Length);  // 3

int[][] b = new int[3][];
b[0] = new int[4];
Console.WriteLine("{0}, {1}", b.Length, b[0].Length);   // 3, 4

int[,] c = new int[3, 4];
Console.WriteLine(c.Length);   // 12
Console.WriteLine("{0}, {1}", c.GetLength(0), c.GetLength(1));  // 3, 4
```



Passaggio di array come argomento

- Gli array possono essere passati utilizzando out e ref.
- Per esempio:

```
void Fill(out int[] a) {
 a = new int[]{1,2,3};
}
int[] a;
Fill(out a);
```


C# vs JAVA: ARRAY

1. Gli array sono istanze di una classe speciale
2. Il riferimento si dichiara scrivendo: tipo[] w; oppure tipo w[];
3. La lunghezza (non modificabile) è data dall'attributo length
4. Array a più dimensioni sono visti semplicemente come array di array
5. Esiste una sintassi speciale per esprimere array irregolari
6. Non esiste una sintassi per iterare su array o su insiemi di proprietà

1. Gli array sono istanze di una classe speciale
2. Il riferimento si dichiara scrivendo solo tipo[] w;
3. La lunghezza (non modificabile) è data dall'attributo Length (notare la maiuscola iniziale)
4. Esiste una sintassi speciale per esprimere array a più dimensioni
5. Esiste una sintassi speciale per esprimere array irregolari
6. Esiste una sintassi speciale per iterare su array: foreach

Costrutto foreach su Array

- Per iterare su un insieme di oggetti o di proprietà, C# offre anche il costrutto foreach.
- Sintassi:

foreach (tipo nomevar in raccolta) istruzione;

```
class Esempio10 {
        public static void Main(){
                int []v = new int[10];
                foreach(int index in v)
                        Console.WriteLine( v[index] );
        }
}
```

Itera per ogni elemento dell'array (equivale a un for classico da 0 a N-1)
La vera utilità di questo costrutto consiste però nel permettere di iterare su *insiemi di proprietà*

Arrays di lunghezza variabile

```
using System;
using System.Collections;

class Test {

    static void Main() {
     ArrayList a = new ArrayList();
     a.Add("Charly");
     a.Add("Delta");
     a.Add("Alpha");
     a.Sort();
     for (int i = 0; i < a.Count; i++)
         Console.WriteLine(a[i]);
    }
}
```

Array associativi

```
using System;
using System.Collections;

class Test {

    static void Main() {
     Hashtable phone = new Hashtable();
     phone["Anna"] = 7131;
     phone["Pippo"] = 7130;
     phone["Pluto"] = 7132;
     foreach (DictionaryEntry x in phone)
         Console.WriteLine("{0} = {1}", x.Key, x.Value);
    }
}
```


String

- An immutable sequence of Unicode characters
- Reference type
- Special syntax for literals
- string s = "I am a string";

Tipo C#	System Type	Bytes
string	System.String	Minimo 20

C# vs JAVA: STRINGHE

Java

- Le stringhe sono istanze della classe String
- Si creano nuove istanze stringa con new
- Si creano nuove istanze specificando stringhe costanti ("ciao")
- L'operatore == vale fra riferimenti (no aliasing): l'identità di contenuto è verificata dal metodo equals
- L'operatore != vale fra riferimenti (no aliasing): la diversità di contenuto è controllata da !equals

C#

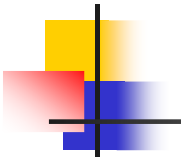
- Le stringhe sono istanze della classe predefinita string
- Non si possono creare nuove istanze stringa con new
- Si creano nuove istanze specificando stringhe costanti ("ciao")
- L'operatore == è overloaded per le stringhe e valuta l'identità di contenuto (non esiste equals)
- L'operatore != è overloaded per le stringhe e valuta la diversità di contenuto (non esiste equals)

Test Uguaglianza In C#

l'operatore **==** è ridefinito per **string** → test verifica l' **identità di contenuto**

```
using System;
        class Esempio5 {
        public static void Main(){
        string s1 = "ciao", s2 = "ciao"; // new string è ERRATO
        Console.WriteLine((s1==s2) ? "uguali" : "diverse");
        Persona p1 = new Persona("John", 23),
        p2 = new Persona("Holly", 18);
        Console.WriteLine((p1==p2) ? "uguali" : "diverse");
        }}
class Persona {
        string nome;
        int eta;
        public Persona(string nome, int eta){
                this.nome=nome;
                this.eta =eta;}
```

l'operatore **==** non è ridefinito per **persona** → test **fra riferimenti** (aliasing)



System.String

Può essere usata come il tipo standard string

```
string s = "Alfonso";
```

Note

- Le Strings non possono essere modificate (tipo StringBuilder per modificare)
- Possono essere concatenate +: "Don " + s
- Possono essere indicizzate: s[i]
- Calcolo della lunghezza: s.Length
- Le stringhe sono dei riferimenti e, quindi, adottano la semantica dei riferimenti nella assegnazioni
- == e != confrontano il contenuto: if (s == "Alfonso") ...
- La classe String definisce I metodi per manipolare le stringhe: CompareTo, IndexOf, StartsWith, Substring, ...