Introduzione a .NET (2)

- Il linguaggio C#
- Implementazioni alternative dell'architettura .NET:
  - Rotor
  - Mono

C#: un nuovo linguaggio

C# : CLR = Java : JVM

- C# è il linguaggio di riferimento per .NET
- Costruito su misura per sfruttare a fondo tutte le funzionalità del framework
- L'eredità dei suoi “genitori” (C++, Java) è notevole:

```
public class hello {
     public static void Main() {
          System.Console.WriteLine(“Hello, world!”);
     }
}
```

C#: fondamenti

- `namespace`: permettono di organizzare logicamente le classi.
  - equivalenti ai `packages` di Java.
- tipi base: `byte`, `int`, `long`, `double`,...
  - Ad ogni tipo base corrisponde un tipo di runtime (ad es. `int` → `System.Int32`)
- costrutti "standard": `if`, `while`. `for`, `switch`.
  - sintassi e semantica (quasi) identica agli omonimi di Java (prevedibile!), `if` e `while` richiedono un `bool`.
- variabili: `static`, `readonly`, `const`.

C#: fondamenti

- Scorciatoie sintattiche:
  - `switch` consente l'utilizzo di una stringa come elemento di confronto.
  - nuovo costrutto `foreach` per esaminare i valori di un array.

```
for (int pos; pos < array.Cont; pos++) {
   oggetto elem = (oggetto) array[pos];
   elem.metodo();
}
```

  - si può riscrivere come:

```
foreach (oggetto elem in array) {
   elem.metodo();
}
```

C#: array e stringhe

- Array: sintassi (e semantica) Java-like:
  - `int [] vettore = new int[10];`
  - `int [] vettore = { 1, 2, 3, 4 };`
  - `int [,] matrice = new int [5,8];`
  - `int [][] matrice = {new int [5], new int [3];}`
- `System.array` contiene metodi utili, tra cui `Sort`, `Reverse`, e `IndexOf`.
- Stringhe: definite tramite il tipo `string`
  - E' possibile accedere direttamente all'i-simo carattere (come in C, ma non in Java).
  - allocate staticamente (come in Java): per utilizzare stringhe dinamiche, si usa la classe `StringBuilder` (analoga a `StringBuffer` di Java)

C#: boxing/unboxing

- Il compilatore provvede a trasformare una variabile in un oggetto, e viceversa.
  - In altre parole. il compilatore provvede ad generare un reference type dove occorre, a partire dal value type (boxing), e viceversa (unboxing).

```
System.Console.WriteLine(15.ToString());
```

```
ArrayList colori;
Color colore;
....
colore.red=(byte)150;
colore.blue=(byte)20;
colore.green=(byte)10;
colori.Add( colore );
```

C#: parametri delle funzioni

- Semantica arricchita per i parametri delle funzioni:
  - in (default): tipi base per valore, oggetti per riferimento (come puntatori in C).
  - ref: parametri sempre per riferimento.
  - out: parametro di output, nessun controllo sull'eventuale inizializzazione.

C#: delegates

- Possibilità di definire puntatori a funzione (o meglio, funzionali – functor) tramite delegates.

.NET: Assembly

- L'evoluzione delle DLL
- autodescrittivi; composti da:
  - Manifest (versione, informazioni generali, dipendenze da altri assembly, tipi, informazioni di sicurezza,...);
  - Codice (MSIL): le classi sono organizzate gerarchicamente tramite namespace;
  - Risorse.
- Goodbye DLL Hell !
  - differenti versioni dello stesso assembly possono convivere;
  - notevole cambiamento rispetto al modello classico di Windows (file LIB/DLL).

C#: classi ed oggetti

- Definizione delle classi: analoga a quella di Java/C++.
- Ereditarietà singola, con interfacce.

```
[attributi] [modificatori] class <nomeClasse>
[:nomeClasseBase] [implements interfaccia1[,interfaccian]]
```

- Modificatori di visibilità
  - per classi: `public`, (`internal`).
  - per classi nested: `public`, (`private`), `internal`, `protected`, `protected internal`.

Visibilità	Significato
public	Accesso non ristretto
protected	Accesso limitato alla classe contenitore o ai tipi da essa derivati
internal	Accesso limitato al progetto corrente (assembly)
protected internal	Accesso limitato al progetto corrente ed ai tipi derivati dalla classe contenitore
private	Accesso limitato al tipo contenitore

C#: ereditarietà e polimorfismo

- maggiore espressività per la ridefinizione dei metodi:
  - aprire una “catena” polimorfa, senza nessun elemento iniziale: `abstract`.
  - aprire una catena polimorfa, con elemento iniziale: `virtual`
  - elementi intermedi della catena: `override`
  - ultimo elemento della catena: `sealed override`
  - aprire una nuova catena, chiudendo la precedente: `new virtual`
  - aprire una catena di un solo elemento (non polimorfo), chiudendo la precedente: `new`
  - bloccare l'ereditarietà: `sealed`.
- Overloading degli operatori.

C#: property

- E' possibile definire metodi che si comportano come un attributo.

```
public class articolo
{
     private int lCosto = 1000;
     public int costo
     {
          set {
               if (value>0)
                    lCosto = value;
          }
          get {
               return lCosto;
          }
     }
}
```

C#: indexer

- Permette di utilizzare un oggetto come se fosse un array.
  - si definisce un metodo di nome `this`.

```
private stringCollection cache;
public object this[string name] {
   get {
      if (cache.Contains( name )) {
         long position = (long)cache[name];
            return getBufferAtPosition( position );
      }
      return “”;
   }
   set {
      if (c.Contains(name)) {
         long position = (long)cache[name];
            saveBufferAtPosition( position, value );
      } else {
         appendBuffer( value );
      }
   }
}
```

C#: altre caratteristiche

- garbage collector
- eccezioni (Java-like)
- codice “unsafe” (possibilità di uso diretto dei puntatori)
- riflessione
- thread
- enum (non corrispondono ad un intero, come avviene in C)
- definizione di attributi (per combinazione condizionale)

E, prossimamente:

- Generics (ma anche in Java 1.5!)

Shared Source Common Language Interface / Rotor

- Implementazione d'esempio delle componenti .NET rese standard dall'ECMA:
  - 334: Il linguaggio C#
  - 335: Common Language Infrastructure (CLI)
- Distribuzione del codice sorgente, *non commerciale*.
  - Cross-platform: Windows XP, FreeBSD, MacOSX.
  - Archivio tgz, poco più grande di 16Mb: contiene implementazione e codice di test.
  - Ampia documentazione: ulteriori 5Mb in formato html.

Rotor vs CLR commerciale

- Entrambi implementazioni complete degli standard ECMA.
- Rotor è un sovrainsieme di ECMA, sottoinsieme del framework .NET completo.
- Rotor è derivato dal codice sorgente della versione commerciale.
  - Le migliorie apportate sono state reintegrate nel “main tree”.
  - Il compilatore C# è praticamente lo stesso!

Rotor vs CLR commerciale

- Differenze tra SSCLI e .NET framework commerciale:
  - il compilatore JIT e il garbage collector sono stati rimpiazzati da implementazioni portabili.
  - sono state rimosse molte funzionalità specifiche di Windows, come l'interoperabilità con oggetti COM, WinForm, ecc.
  - non sono stati incluse molte caratteristiche commerciali, come ADO.NET, servizi enterprise, e ASP.NET.

Mono

- Progetto avviato da Ximian. (ora acquisita da Novell)
- Re-implementazione completa del .NET framework
  - Compilatore C# (self-hosted!) e VB (in sviluppo) (GPL)
  - Ambiente di runtime (interprete normale e JIT) (LGPL)
  - Libreria di classi.
    - si appoggia a varie librerie opensource: libart (2d), gtk (Windows.Forms), gnome-db (ADO.NET), ...
- Cross-platform: Linux/Unix, Windows.

Mono

- Sviluppo molto rapido.
- Roadmap:
  - 1.0 (Q2 2004): core, C#, database, web application.
    - allineato con .NET 1.1, compatibilità con .NET 1.0
  - 1.2 (Q4 2004): Windows.Forms, VB.NET, generics,
    - seguirà .NET 1.2

Rotor vs Mono

- Licenza
  - Rotor è non commerciale: non è possibile (al contrario di Mono) effettuare modifiche e ridistribuire a pagamento il codice. (target: ambiente accademico).
- Completezza
  - Rotor è un'implementazione di “riferimento”, limitata alle componenti rese standard. Manca di tutte le caratteristiche necessarie allo sviluppo di applicazioni.
  - il termine di confronto di Mono è (o sarà) il .NET framework completo.