Programmazione Orientata agli Oggetti in Linguaggio Java

Sintassi e Semantica:

C#

Parte a

versione 2.4

G. Mecca – Università della Basilicata – mecca@unibas.it

Sommario

- Elementi Comuni
- Il Sistema di Tipi
  - ⇨Boxing e Unboxing
- Dati di un'Applicazione C#
- Terminologia di .NET
- Proprietà e Indicizzatori

Elementi Comuni

- C# è un linguaggio a oggetti
  - ⇨un'applicazione C# è fatta di componenti
  - ⇨i componenti hanno proprietà e metodi
  - ⇨due tipi di componenti: classi e oggetti
  - ⇨gli oggetti sono creati dalle classi usando i costruttori
  - ⇨e si manipolano utilizzando i riferimenti

Elementi Comuni

- Inoltre
  - ⇨C# è un linguaggio basato su Unicode
  - ⇨prevede la "garbage collection" automatica
  - ⇨gli array di C# sono oggetti e si manipolano attraverso i riferimenti
  - ⇨C# è più fortemente tipato del C++ (non ci sono troncamenti espliciti e i numeri non sono booleani)

Elementi Comuni

- Inoltre (continua)
  - ⇨C# richiede l'inizializzazione esplicita dei dati prima dell'uso
  - ⇨le componenti degli array sono inizializzate automaticamente con la regola del v. nullo
  - ⇨le proprietà dei componenti sono inizializzate automaticamente con la regola del v. nullo
  - ⇨attenzione alle variabili locali dei metodi

Elementi Comuni

- Inoltre (continua)
  - ⇨tutte le classi estendono System.Object
  - ⇨ereditano i metodi di System.Object, tra cui bool Equals(Object o) e string ToString()
  - ⇨e sono sostituibili ad oggetti di tipo System.Object
- Leggera differenza sintattica
  - ⇨non esiste la parola chiave "extends"
  - ⇨public class Animale : EssereVivente {...}

Elementi Comuni

ATTENZIONE
all'approccio alla presentazione di C#

- L'impostazione del corso
  - ⇨nel corso verranno illustrate le principali differenze
  - ⇨a meno che non sia detto esplicitamente, le cose in C# funzionano come in Java
  - ⇨possono in alcuni casi esserci piccole differenze sintattiche
  - ⇨risolvibili facendo riferimento alla documentazione

Il Sistema di Tipi

- I tipi di base principali di C#

tipo	valori	costanti
int	interi a 32 bit (complemento a 2)	234 -3 987345
double	virgola mobile a 64 bit	1.12 -1E-2 8.233D
float	virgola mobile a 32 bit	1.12F -1E-2F 8.233F
char	**caratteri Unicode (16 bit)**	'a' '0' '\n' '\t' '\\' '\'' '\"' '\u12de'
bool	**true o false**	true o false
string	**stringhe di caratteri**	"Ciao\n" "Mario Rossi"

Il Sistema di Tipi

- Altri tipi di base di C#

tipo	valori	valori
sbyte	interi a 8 bit con segno	da -128 a 127
byte	interi a 8 bit senza segno	da 0 a 255
short	interi a 16 bit con segno	da -32768 a 32767
ushort	interi a 16 bit senza segno	da 0 a 65535
uint	interi a 32 bit senza segno	da 0 a 4 miliardi circa
long	interi a 64 bit con segno	da -1E18 a 1E18 circa
ulong	interi a 64 bit senza segno	da 0 a 1E19 circa
decimal	reali in virgola mobile a 128 bit	da -1.0E28 a 1.0E28

Il Sistema di Tipi

ATTENZIONE
a bool e string

- Attenzione
  - ⇨il tipo a valori booleani si chiama bool (e non boolean come in Java)
  - ⇨ma ha le stesse caratteristiche del tipo boolean di Java (e non del bool di C++)
  - ⇨il tipo string è considerato un tipo di base (con la s minuscola)
  - ⇨ma esiste anche la classe String
  - ⇨è un esempio di “boxing” automatico

Boxing e Unboxing

- In effetti
  - per ogni tipo di base è prevista in C# una corrispondente classe “wrapper”
  - a seconda dell’utilizzo, in C# è prevista una forma di trasformazione automatica
  - dai valori dei tipi di base ai corrispondenti oggetti (boxing automatico)
  - e viceversa, dagli oggetti ai corrispondenti tipi di base (unboxing automatico)

Boxing e Unboxing

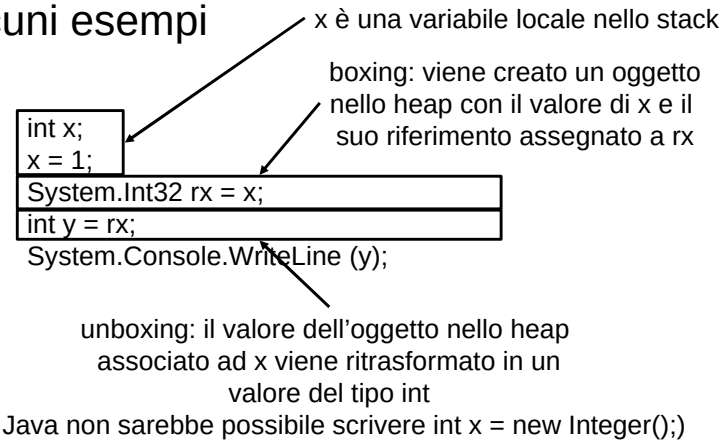
- Nota
  - in Java il boxing e l’unboxing sono normalmente manuali
  - una forma limitata di boxing ed unboxing automatico è stata introdotta in J2SE 5.0
- In C#
  - questo esiste fin dalla prima versione
  - in forma molto più spinta

Boxing e Unboxing

- Le classi wrapper
  - ⇨bool - System.Boolean
  - ⇨char - System.Char
  - ⇨int - System.Int32
  - ⇨float - System.Single
  - ⇨double - System.Double
  - ⇨string - System.String

Boxing e Unboxing

- Alcuni esempi



Boxing e Unboxing

- Nota
  - ⇨in J2SE 5.0 il boxing e l'unboxing automatico si verificano solo in sede di assegnazione di un valore di un tipo primitivo ad un riferimento (es: Double d = 2.5);
  - ⇨in C#, invece, valori dei tipi primitivi e corrispondenti classi wrapper sono del tutto intercambiabili

Boxing e Unboxing

in Java:
int x = 1; int y = 1;
Integer rx = new Integer(x);
Integer ry = new Integer(y)
if (rx.equals(ry)) {...}

- Alcuni esempi

x e y sono variabili del tipo di base int

x e y si comportano da valori di base

```
int x, y;
x = 1;
y = 1;
```

```
if (x == y) {
    System.Console.WriteLine("Uguali");
}
```

```
if (x.Equals(y)) {
    System.Console.WriteLine("Uguali");
}
```

x e y si comportano da riferimenti a oggetti

Boxing e Unboxing

- Per farla breve
  - ⇨i tipi primitivi sono in pratica “alias” delle corrispondenti classi wrapper
  - ⇨è possibile utilizzare i valori dei tipi primitivi come se fossero oggetti e viceversa
- Esempio
  - ⇨utilizzare string e System.String in un programma C# è del tutto indifferente

Dati di un’Applicazione C#

- In un’applicazione C#
  - ⇨valori dei tipi di base
  - ⇨componenti: classi e oggetti
  - ⇨riferimenti agli oggetti
- I dati nei metodi
  - ⇨variabili
  - ⇨costanti
  - ⇨parametri

Dati di un'Applicazione C#

- Costanti
  - ⇨si usa la parola chiave const
  - ⇨equivalente a static e final assieme
- Parametri
  - ⇨più complessi di quelli di Java ma:
  - ⇨per i tipi di base il passaggio è per valore
  - ⇨per i riferimenti il passaggio è identico a quello di Java

Dati di un'Applicazione C#

- Ma...
  - ⇨C# è sensibilmente più complesso di Java
  - ⇨per esempio, oltre alle classi esistono le strutture e le enumerazioni
  - ⇨e molti altri elementi del linguaggio
- Nel seguito
  - ⇨un'indicazione di alcuni di questi

Dati di un'Applicazione C#

- In C# sopravvivono
  - ⇨i record ed i tipi enumerati
- Enum in C#
  - ⇨analoghe ai tipi enumerati del C++
  - ⇨sono semplici classi utilizzate per rappresentare tipi a valori discreti ed enumerabili (>>)

Dati di un'Applicazione C#

- Struct in C#
  - ⇨strutture analoghe a quelle del C++
  - ⇨sono considerate semplici classi con campi e metodi
- La differenza fondamentale
  - ⇨lo spazio in memoria per le istanze delle strutture è allocato nello stack e non nello heap (hanno una "semantica di valore")
  - ⇨non si manipolano utilizzando i riferimenti

Dati di un'Applicazione C#

>> System

- In effetti
  - ⇨i tipi wrapper sono tutti strutture e non classi
  - ⇨es: System.Int32
- Attenzione
  - ⇨si tratta di uno strumento insidioso
  - ⇨perchè sembrano componenti tradizionali ma in realtà non lo sono
  - ⇨è opportuno evitarne per ora l'utilizzo

Terminologia di .NET

ATTENZIONE alla terminologia di C#

- Attenzione alla terminologia di C#
- In Java
  - ⇨si dice che i componenti hanno proprietà
  - ⇨o, in alternativa, attributi o campi
  - ⇨terminologia ispirata a quella dei record
- In C#
  - ⇨i tre termini hanno significati completamente diversi

Terminologia di .NET

- In C#
  - ⇨i componenti hanno campi
  - ⇨equivalente delle proprietà di Java
- Ma esistono anche
  - ⇨attributi: forma di “metainformazione” per descrivere il codice
  - ⇨proprietà: modo rapido per scrivere metodi get e set

Terminologia di .NET

>> circonferenzec\Circonferenza.cs

- Attributi .NET
  - ⇨informazioni aggiuntive sul codice
  - ⇨“metainformazioni”: informazioni sul codice e non codice vero e proprio
  - ⇨possono riferirsi a vari elementi
- Sintassi
  - ⇨stringhe tra parentesi quadre
  - ⇨sulla linea precedente a quella dell’elemento a cui si riferiscono

Terminologia di .NET

>> utilita\AssemblyInfo.cs

- Tipologie di attributi
  - ⇨attributi della singola classe\metodo\istruz.
  - ⇨sono specificati nel file relativo
  - ⇨attributi dell'intera applicazione (assembly)
- Una convenzione di .NET
  - ⇨gli attributi dell'applicazione vengono specificati in un file apposito denominato AssemblyInfo.cs

Proprietà e Indicizzatori

- Un obiettivo di .NET
  - ⇨consentire la migrazione della comunità di programmatori Microsoft verso la programmazione a oggetti
  - ⇨programmatori provenienti dal Visual Basic o dal Visual C++
  - ⇨abituati ad uno stile di programmazione "ibrido", con un taglio fortemente procedurale

Proprietà e Indicizzatori

- Per evitare di disorientarli
  - ⇨il linguaggio .NET fornisce una serie di funzionalità sintattiche che fanno in modo che il programmatore “abbia l’impressione” di scrivere codice procedurale
  - ⇨proprietà >> per rendere gli oggetti simili ai record/strutture della prog. procedurale
  - ⇨indicizzatori >> per rendere gli oggetti che rappresentano collezioni simili agli array

Proprietà e Indicizzatori

- Proprietà di C#
  - ⇨consentono di evitare l’uso dei metodi get e set
- Nella programmazione a oggetti tipica
  - ⇨tipicamente le proprietà sono private
  - ⇨e per consentire di manipolarne si forniscono metodi get e/o set
  - ⇨il programmatore deve chiamare i metodi get e set per utilizzare il componente

Proprietà e Indicizzatori

- In C#
  - ⇨è possibile fare esattamente la stessa cosa
  - ⇨oppure scrivere il codice in modo più compatto usando le proprietà
- Idea
  - ⇨la proprietà è uno strumento per dare l'impressione all'esterno che il componente abbia un campo pubblico
  - ⇨nonostante questo sia protetto da set/get

Proprietà e Indicizzatori

- Esempio

```
class Circonferenza {
  private double raggio;
  public double Raggio {
    get {
      return this.raggio;
    }
    set {
      this.raggio = value;
    }
  }
  ...
}
```

campo privato del componente proprietà: stesso nome del campo (con la maiuscola) + coppia di metodi di accesso

valore dell'argomento del set

```
class Principale {
  ...
  private stampaCirconferenza(
      Circonferenza c)  {
    ...
    Console.WriteLine(c.Raggio);
    c.Raggio = 10;
    ...
  }
  ...
}
```

equivalente a c.getRaggio()

equivalente a c.setRaggio(10)

Proprietà e Indicizzatori

>> circonferenzec
>> segmentib

- Nota
  - si tratta di un fenomeno esclusivamente sintattico
  - nella sostanza non cambia nulla
  - resta il dato (campo) privato e i metodi di accesso pubblici
  - ma la scrittura del codice risulta più compatta
  - e il dato è più protetto che se fosse pubblico

Proprietà e Indicizzatori

- L'effetto finale
  - è quello di dare l'illusione che il componente sia un record, con campi pubblici
- Esempio: Circonferenza, struttura con:
  - AscissaCentro, OrdinataCentro, Raggio
  - Lunghezza (costante)
  - Superficie (costante)

Proprietà e Indicizzatori

- Alcune proprietà notevoli
  - ⇨la proprietà Lenght degli array di C#
  - ⇨es: int [] array = new int [5];
  - ⇨for (int i = 0; i < array.Length; i++) { ... }
  - ⇨la proprietà Lenght di System.String
  - ⇨es: System.String s = “prova”;
  - ⇨System.Console.WriteLine(s.Length);

Proprietà e Indicizzatori

- Indicizzatori di C#
  - ⇨strumento che consente di utilizzare un componente che contiene una collezione di dati come se fosse un array
  - ⇨consentendo di manipolarlo con le [ ]
  - ⇨ma anche in questo caso l’accesso e la modifica è implementata con set e get

Proprietà e Indicizzatori

- Un esempio
  - la classe System.String
  - indicizzatore per accedere ai caratteri della stringa
- Esempio
  - System.String s = “Prova”;
  - System.Console.WriteLine(s[2]); // stampa ‘o’

Proprietà e Indicizzatori

- Uno sguardo (virtuale) dentro String
  - un campo di tipo array di caratteri
  - un indicizzatore per consentire l’accesso
- Esempio

indicizzatore per gli oggetti di tipo MiaStringa; può essercene al più uno e si dichiara con this[int indice]

```
public class MiaString {

    private char[] valori = new char[100];

    public char this[int i] {
        get { return valori[i]; }
        set { valori[i] = value; }
    }
}
```

Proprietà e Indicizzatori

- In sintesi
  - ⇨proprietà e indicizzatori hanno obiettivi simili
  - ⇨nascondono metodi del componente
- Un componente può avere
  - ⇨molte proprietà, che si comportano all'esterno come campi pubblici
  - ⇨un unico indicizzatore (proprietà multivalore), che si comporta all'esterno come un array tradizionale

Riassumendo

- Elementi Comuni
- Il Sistema di Tipi
  - ⇨Boxing e Unboxing
- Dati di un'Applicazione C#
- Campi e Attributi
- Proprietà e Indicizzatori

Termini della Licenza