

24 - Possibili approfondimenti

Programmazione e analisi di dati
Modulo A: Programmazione in Java

Paolo Milazzo

Dipartimento di Informatica, Università di Pisa
<http://www.di.unipi.it/~milazzo>
milazzo@di.unipi.it

Corso di Laurea Magistrale in Informatica Umanistica
A.A. 2013/2014

Approfondimenti

Il linguaggio Java è estremamente ricco di funzionalità

Lo scopo di questa breve lezione è di fornire **spunti** e **riferimenti** per approfondimenti individuali

Iniziamo con un po' di approfondimenti sulla programmazione a oggetti di cui potrete trovare spiegazione nei libri di testo (alcuni) o sul web.

Altro sulla programmazione a oggetti (1)

Classi interne (o **inner class**). E' possibile definire una classe all'interno di un'altra

```
public class RegistroEta {  
  
    // classe interna (privata) che potra' essere usata solo  
    // nell'ambito della classe RegistroEta  
    private class NomeEta {  
        public String nome;  
        public int eta;  
    }  
  
    private Vector<NomeEta> registro = new Vector<NomeEta>();  
  
    public void inserisci(String nome, int eta) {  
        NomeEta ne = new NomeEta();  
        ne.nome = nome;  
        ne.eta = eta;  
        registro.add(ne);  
    }  
}
```

Altro sulla programmazione a oggetti (2)

Classi anonime. E' possibile definire classi "al volo" per singoli oggetti

```
public void metodo() {
    // crea "al volo" una classe che implementa ActionListener,
    // definendo il metodo actionPerformed. Tale classe viene
    // subito usata per creare un oggetto da passare al metodo
    // addActionListener di button
    ActionListener listener = new ActionListener(){
        public void actionPerformed(ActionEvent evt) {
            JOptionPane.showMessageDialog(null, "Buongiorno!");
        }
    };
    JButton button = new JButton("Saluta");
    button.addActionListener(listener);
}
```

Altro sulla programmazione a oggetti (3)

Classi generiche E' possibile definire **proprie** classi che ricevono tipi come parametri

- Vector è una classe parametrica... in Vector<String> la classe string è un parametro

```
// una classe che rappresenta un coppia di elementi dello stesso
// tipo. Il tipo e' generico (T) e sara' specificato al momento
// della chiamata del costruttore
public class Coppia<T> {
    public T elemento1;
    public T elemento2;

    public Coppia(T elemento1, T elemento2) {
        this.elemento1 = elemento1;
        this.elemento2 = elemento2;
    }
}
```

```
// una coppia di interi
Coppia<Integer> = new Coppia<Integer>(10,20);
// una coppia di stringhe
Coppia<String> = new Coppia<String>("Ciao","tutti");
```

Novità delle ultime versioni di Java

La versione attuale di java è la 7.

Ogni nuova versione porta con se:

- Aggiornamenti nei **costrutti del linguaggio**
- Aggiornamenti nelle **classi della libreria** standard (Java API)

Può essere interessante andare a vedere come è evoluto il linguaggio da una versione all'altra, e come evolverà nella prossima versione.

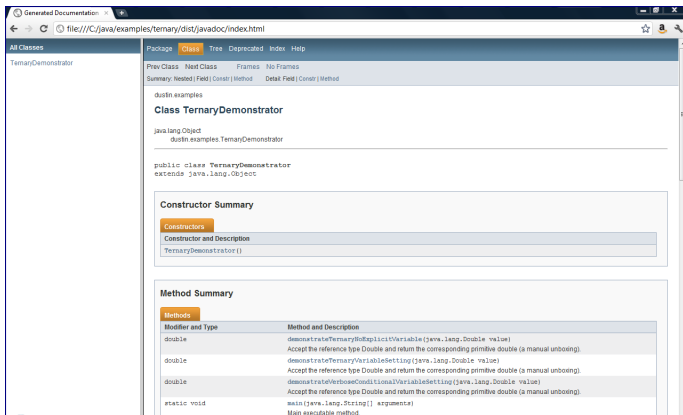
Qualche articolo in merito:

- **Java 5 e 6:** <http://www.html.it/pag/18022/introduzione-a-java-6/>
- **Java 7:**
<http://www.html.it/articoli/project-coin-le-novit-di-java-7-1/>
- **Java 8: (lettura difficile)** <http://www.cosenonjaviste.it/il-java-che-verra-una-breve-introduzione-alle-espressioni-lambda/>

Documentazione con Javadoc (1)

Java mette a disposizione uno strumento (**Javadoc**) per creare semi-automaticamente pagine web di documentazione del proprio codice

Usando speciali commenti all'inizio delle proprie classi e dei propri metodi si potrà ottenere un risultato simile al seguente:



The screenshot shows a web browser window titled "Generated Documentation" displaying the Javadoc for the `TernaryDemonstrator` class. The browser address bar shows the file path `file:///C:/java/examples/ternary/dist/javadoc/index.html`. The page layout includes a navigation sidebar on the left with "All Classes" and "TernaryDemonstrator". The main content area has a top navigation bar with "Package", "Class", "Tree", "Deprecated", "Index", and "Help". Below this is a sub-navigation bar with "Prev Class", "Next Class", "Frames", and "No Frames". The main content is organized into sections: "Summary", "Constructor Summary", and "Method Summary".

Class TernaryDemonstrator

```
public class TernaryDemonstrator
extends java.lang.Object
```

Constructor Summary

Constructor and Description
<code>TernaryDemonstrator()</code>

Method Summary

Modifier and Type	Method and Description
double	<code>demonstrateTernaryToExplicitVariable(java.lang.Double value)</code> Accept the reference type Double and return the corresponding primitive double (a manual unboxing)
double	<code>demonstrateTernaryVariableSetting(java.lang.Double value)</code> Accept the reference type Double and return the corresponding primitive double (a manual unboxing)
double	<code>demonstrateVerboseConditionalVariablesSetting(java.lang.Double value)</code> Accept the reference type Double and return the corresponding primitive double (a manual unboxing)
static void	<code>main(java.lang.String[] arguments)</code> Main executable method

Documentazione con Javadoc (2)

Ad esempio, la classe Rettangolo

```
/**
 *
 * Classe che descrive un rettangolo
 *
 * @author Paolo Milazzo
 *
 */
public class Rettangolo {

    // membri privati
    private int base;
    private int altezza;

    /**
     *
     * Costruttore di un rettangolo
     *
     * @param base Base del rettangolo
     * @param altezza Altezza del rettangolo
     */
    public Rettangolo(int base, int altezza) {
        this.base = base;
        this.altezza = altezza;
    }
}
```

(segue)

Documentazione con Javadoc (3)

(segue Rettangolo)

```
/**
 * Restituisce la base del rettangolo
 *
 * @return la base del rettangolo
 */
public int getBase() { return base; }

/**
 * Imposta una nuova base per il rettangolo
 *
 * @param base la nuova base
 */
public void setBase(int base) { this.base = base; }
```

(segue)

Documentazione con Javadoc (4)

(segue Rettangolo)

```
/**
 * Restituisce l'altezza del rettangolo
 *
 * @return l'altezza del rettangolo
 */
public int getAltezza() { return altezza; }

/**
 * Imposta una nuova altezza per il rettangolo
 *
 * @param altezza la nuova altezza
 */
public void setAltezza(int altezza) { this.altezza = altezza; }
}
```

JavaDoc può essere eseguito tramite Eclipse

- menu Project -> Generate Javadoc...

Per il risultato vedere [Rettangolo.zip](#) sulla pagina web (il file [index.html](#))

Programmazione concorrente (1)

Molto spesso nei programmi si utilizzano aspetti di **programmazione concorrente**:

- parti diverse del programma vengono eseguite **in parallelo**
- mentre una parte è impegnata a fare un calcolo, gestire input/output o altro, le altre possono proseguire l'esecuzione
- il parallelismo può essere
 - ▶ reale (esempio: su processori multicore)
 - ▶ simulato (il processore è unico, ma le varie parti si alternano continuamente per dare la sensazione di essere eseguite contemporaneamente)
- programmi concorrenti in Java possono essere ottenuti tramite i **thread**

Programmazione concorrente (2)

Esempio: una gara di matematica...

```
// partecipa a una gara in cui bisogna fare 10000 calcoli inutili
public class Concorrente implements Runnable {

    int num;

    public Concorrente(int num) {
        this.num = num;
    }

    // metodo eseguito da un thread
    public void run() {
        for (int i=0; i<=10000; i++) {
            double k = 0.0;
            // calcolo inutile ma che richiede un po' di tempo
            k += Math.pow(i/100,i%100)-Math.pow(i/100,i%100);
            // visualizza un messaggio ogni 10% del ciclo
            if (i%1000==0)
                System.out.println("Concorrente["+num+"] -- "+i/100+"%");
        }
    }
}
```

Programmazione concorrente (3)

```
public class Gara {  
    public static void main(String[] args) {  
        Concorrente c1 = new Concorrente(1);  
        Thread t1 = new Thread(c1);  
        Concorrente c2 = new Concorrente(2);  
        Thread t2 = new Thread(c2);  
        Concorrente c3 = new Concorrente(3);  
        Thread t3 = new Thread(c3);  
  
        // fa partire i tre concorrenti  
        t1.start();  
        t2.start();  
        t3.start();  
    }  
}
```

Programmazione concorrente (4)

Una esecuzione (questa volta ha vinto il 2!):

```
Concorrente [1] -- 0%
Concorrente [3] -- 0%
Concorrente [1] -- 10%
Concorrente [3] -- 10%
....
....
Concorrente [2] -- 80%
Concorrente [2] -- 90%
Concorrente [2] -- 100%
Concorrente [3] -- 20%
Concorrente [3] -- 30%
Concorrente [3] -- 40%
Concorrente [3] -- 50%
Concorrente [3] -- 60%
Concorrente [3] -- 70%
Concorrente [3] -- 80%
Concorrente [3] -- 90%
Concorrente [3] -- 100%
Concorrente [1] -- 80%
Concorrente [1] -- 90%
Concorrente [1] -- 100%
```

Programmazione concorrente (5)

Riferimenti:

- alcuni libri di testo
- tutorial Java (<http://docs.oracle.com/javase/tutorial/essential/concurrency/>)

Approfondimenti su algoritmi e complessità computazionale

Un buon programmatore dovrebbe avere una buona cultura di **algoritmica**

Lo studio dell'algoritmica:

- aiuta a comprendere come risolvere i problemi computazionali
- insegna a valutare l'**efficienza** dei propri algoritmi
- insegna a progettare e fare buon uso delle strutture dati

Riferimento: Libro di Crescenzi, Gambosi e Grossi. **Strutture di Dati e Algoritmi**. PEARSON, Addison-wesley

Altre cose...

- Programmazione di rete (interfaccia socket, Remote Methods Invocation (RMI), ...)
- Accesso ai database (JDBC, Java DB,....)
- Il linguaggio XML (linguaggio per lo scambio di dati tra applicazioni)
- Il linguaggio C# (principale antagonista di Java...)
-