

Tutorato Informatica II

a.a. 2016-2017

Dott. Marco Radavelli

Prof. Angelo Gargantini

Java conventions and deployment

Convenzioni

La leggibilità del codice e il rispetto di alcuni standard nella sua stesura è di fondamentale importanza nella realizzazione di un buon prodotto software.

E' importante **rispettare le convenzioni** univocamente riconosciute i quanto facilita la comprensibilità, il riuso e la manutenzione del codice.

Elementi interessati dalle convenzioni

- FILE
- Package
- Blocchi
- Convenzioni sui nomi
- Commenti
- JAVA DOC

Organizzazione dei File

- Ogni file sorgente Java contiene una singola classe pubblica o un'interfaccia.
- Se ci sono classi private o interfacce associate, esse possono essere inserite nello stesso file dopo la classe pubblica principale che dà il nome al file.
- Tutti i file sorgenti Java devono avere la seguente struttura:
 - Commenti iniziali (autore, versione, data copyright,...)
 - Package e import
 - Dichiarazione della classe (o dell'interfaccia)

Package e Import

- La definizione del codice occupa la prima riga del codice:

package java.awt;

- Le righe successive prevedono l'import dei package necessari per la classe:

import java.awt.peer.CanvasPeer;

- Il primo campo del nome di un package univoco deve essere minuscolo e deve essere uno dei nomi dei domini di livello più alto:

edu, gov, mil, net, org

- Oppure le due lettere che descrivono una nazione come specificato nello standard ISO 3166, 1981 (it, uk, ch, fr, de,...)

Classi e Interfacce

1. Documentazione della classe o dell'interfaccia (`/** ... */`)
2. Dichiarazione della classe o dell'interfaccia
3. Commento generale all'implementazione, se necessario (`/* ... */`)
4. Class variables: prima quelle pubbliche, poi quelle protette, poi quelle a livello package (senza modificatori) e per ultime le private
5. Instance variables: seguono lo stesso ordine delle class variables
6. Costruttori
7. Metodi: Dovrebbero essere raggruppati per funzionalità e non per scope o visibilità. L'obiettivo è quello di far comprendere il codice nel modo più facile e rapido

Dichiarazione delle variabili

- È consigliato dichiarare una variabile per linea, questo incoraggia i commenti.
- Inizializzare le variabili dove vengono dichiarate
- Posizionare le dichiarazioni all'inizio dei blocchi.

ESEMPIO:

SI:

```
for (int i=0; i<8; i++) {  
    double angolo = 2*Math.PI/8*i;  
    [...]  
}
```

EVITARE:

```
for (i=0;i<8;i++) {  
    [...] // some instructions  
    double angolo;  
    [...]  
    angolo=  
        2*Math.PI/8*i;  
    [...]  
}
```

Java Blocks

```
if (testScore >= 70) {  
    if (studentAge < 10) {  
        System.out.println("You did a great job");  
    } else {  
        System.out.println("You did pass"); //test score >= 70  
                                              //and age >= 10  
    }  
} else { //test score < 70  
    System.out.println("You did not pass");  
}
```


Names

- I package hanno sempre il primo campo tutto minuscolo, i successivi campi sono a scelta.
- Classi e interfacce dovrebbero avere nomi che rappresentano sostantivi, in maiuscolo e possibilmente espressivi delle operazioni fornite. Ogni parola deve iniziare con la lettera maiuscola:

```
class Raster  
class VectorialImage
```

- I nomi dei metodi dovrebbero essere verbi con la prima lettera minuscola e le lettere interne maiuscole (secondo la cosiddetta “camel case notation”):

```
run() ;  
runFast() ;
```

- Le variabili devono avere nomi significativi (eccetto per le temporanee: i, j, k, m, n...) e seguire la stessa convenzione dei metodi.

```
int size;  
myCalc myCalculator;
```

- Le variabili costanti devono essere espresse completamente in maiuscolo usando il carattere ‘_’ come separatore:

```
static final int MIN_WIDTH = 4;  
static final int MAX_WIDTH = 999;
```

Commenti al codice

E' importantissimo che il codice sia **leggibile anche a distanza di tempo** e da parte di persone diverse dall'autore (per essere più facilmente comprensibile e modificabile).

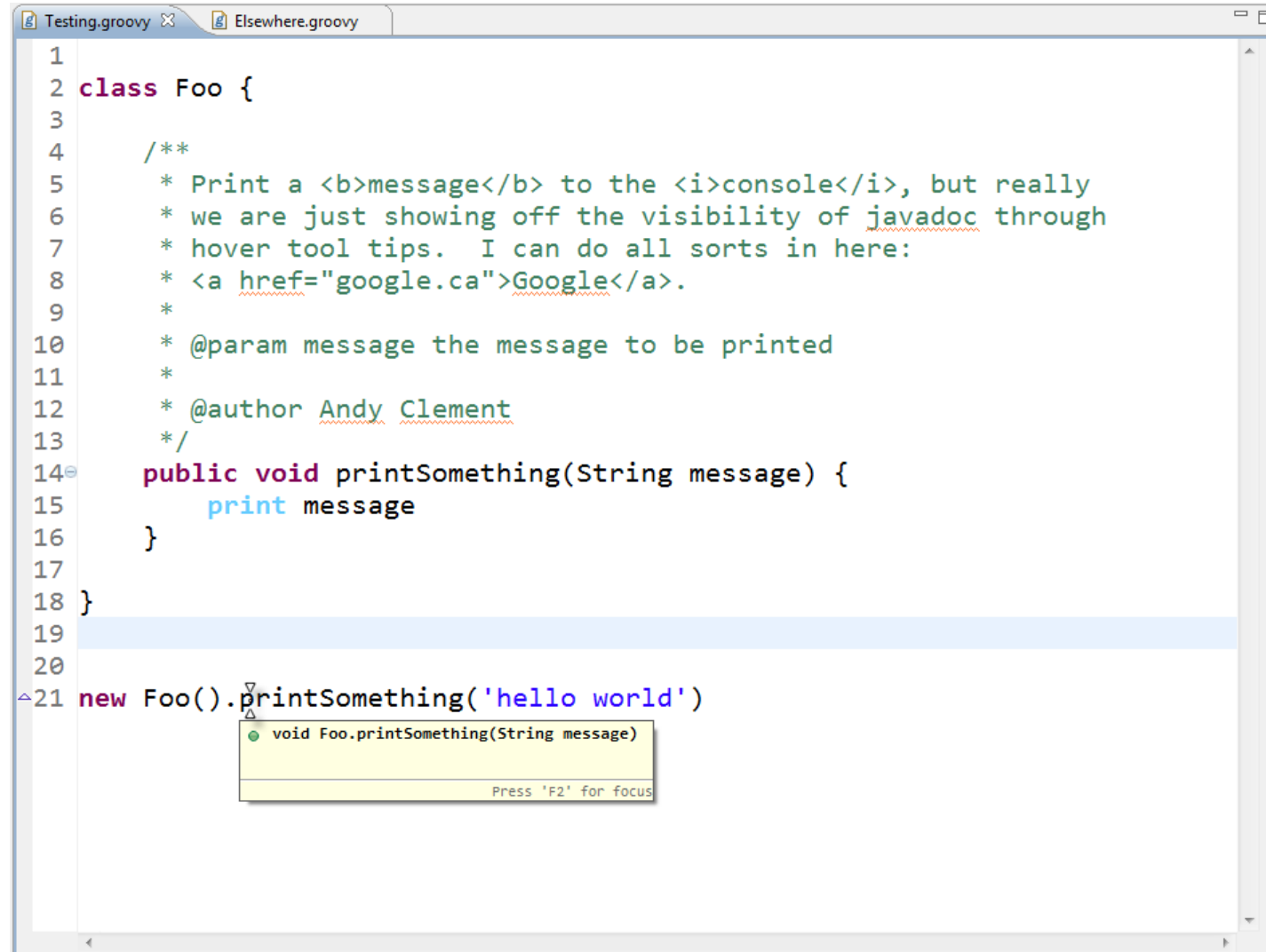
Per questo è necessario **inserire nel codice commenti significativi**, che spieghino le funzionalità dei vari metodi, le scelte fatte e quant'altro l'autore ritiene utile per la comprensione del programma.

```
/* Block comment */
import java.util.Date;
/**
 * Doc comment here for SomeClass
 * @version 1.0
 */
public class SomeClass { // some comment
    private String field = "Hello World";
    private double unusedField = 12345.67890;
    private UnknownType anotherString = "AnotherString";
    public SomeClass() {
        //TODO: something
        int localVar = "IntelliJ"; // Error, incompatible types
        System.out.println(anotherString + field + localVar);
        long time = Date.parse("1.2.3"); // Method is deprecated
    }
}
```

JAVA DOC

- *JavaDoc* nacque come strumento interno utilizzato dai ricercatori della Sun che stavano lavorando alla creazione del linguaggio Java e delle sue librerie
- La grande mole di sorgenti spinse alcuni membri del team a creare un programma per la **generazione automatica di documentazione HTML** (formato conosciuto, veloce da leggere, facilmente indicizzabile)
- Serviva un sistema automatico (per gestire la quantità di riferimenti incrociati che ci sono fra le classi e evitare gli errori di battitura).
- *JavaDoc* nacque quindi **per permettere ai programmatori di inserire dei frammenti HTML nei commenti** (ignorati quindi dal compilatore)

JAVAdoc



```
1
2 class Foo {
3
4     /**
5      * Print a <b>message</b> to the <i>console</i>, but really
6      * we are just showing off the visibility of javadoc through
7      * hover tool tips. I can do all sorts in here:
8      * <a href="google.ca">Google</a>.
9      *
10     * @param message the message to be printed
11     *
12     * @author Andy Clement
13     */
14     public void printSomething(String message) {
15         print message
16     }
17 }
18
19
20
21 new Foo().printSomething('hello world')
```

void Foo.printSomething(String message)

Press 'F2' for focus

JAUTODOC

<http://jautodoc.sourceforge.net/index.html>

- JAutoDoc consente di:
 - Completare la documentazione esistente
 - Aggiungere altra documentazione senza modificare quella esistente
 - Sovrascrivere la documentazione esistente
- È possibile definire la visibilità dei campi da commentare, specificare filtri e template, inserire header nei file.

Per l'utilizzo di JAutodoc

- <http://jautodoc.sourceforge.net/index.html#usage>
- Preferenze documentazione:
 - Window -> Preferences -> java -> JAutoDoc
- Generazione documentazione:
 - Click destro su file o progetto -> JAutoDoc -> Add Javadoc
- Inserimento Header (vuoto o definito da un template specifico)
 - Click destro su file o progetto -> JAutoDoc -> Add header
- Generazione Javadoc:
 - Click destro su progetto -> export -> java -> javadoc
 - Può essere necessario specificare il path a javadoc.exe (distribuito con la JDK)
 - Osserva anche le altre opzioni (fogli di stile, modalità di esportazione, path, filtri...)

JAVA DEPLOYMENT

JAR

A JAR (Java ARchive) file is a file that contains the [class](#), image, and sound files for a [Java](#) application or [applet](#) gathered into a single file and possibly compressed.

When a programmer gets a Java program development kit, a small program or utility called "jar" is included.

The jar utility lets the programmer create, list, or extract the individual files from a JAR file. In an enterprise, a Java application can be started with a set of JAR files for use during execution. An off-the-shelf [open source](#) package can be delivered as a JAR file and run with [XML](#) data.

Eclipse Jar Deployment

- Eclipse permette di generare un archivio jar eseguibile partendo da un progetto eseguibile (dotato di metodo main):
 - Click pulsante destro su progetto -> export -> java -> runnable jar file
 - Specificare il launch configuration da usare
 - Specificare il path di destinazione
 - Specificare come trattare le eventuali librerie richieste dall'applicazione
- Una volta generato è possibile eseguire il jar con il comando
`java -jar nome_del_jar.jar`

ECLIPSE TRICKS

Some (useful) Eclipse Tricks

Comando	uso
CTRL+SHIFT+O	sistema automaticamente gli import (N.B. in caso di omonimi in diversi package, chiede conferma)
Click destro -> Refactor -> Rename	rinomina qualsiasi cosa evidenziata (classe, variabile, metodo..) aggiornando in automatico le referenze in tutto il progetto
CTRL+A seguito da CTRL+I	indenta correttamente il codice
F3	Va a definizione (di qualsiasi metodo/variabile/classe... evidenziata dal cursore)
Click destro -> References -> Project	mostra tutti i punti in cui quella variabile/classe/metodo è referenziato
Source -> Generate Constructor using Fields	
Source -> Generate Getters and Setters...	