

# Tutorato Progettazione, Algoritmi e Computabilità

## Convenzioni Java

**Dott. Michele Beretta**

Prof.ssa Patrizia Scandurra

Università degli Studi di Bergamo

2023 – 2024

- 1 Convenzioni Java
- 2 Commenti e Documentazione
- 3 Deployment con JAR

# Convenzioni Java

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

È importante **rispettare le convenzioni** riconosciute in quanto facilita la comprensibilità, il riuso e la manutenzione del codice.

- Organizzazione dei file
- Package
- Blocchi
- Nomi (funzioni, classi, variabili, etc.)
- Commenti
- Documentazione (Java DOC)

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:

- 1 Commenti iniziali (autore, versione, data, copyright, etc.)
- 2 Package e import
- 3 Dichiarazione della classe (o dell'interfaccia)

La definizione del package occupa la prima riga, mentre le righe successive prevedono l'import dei package necessari per la classe

```
package java.awt;
```

```
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, etc.)

- 1 *Documentazione* della classe o dell'interfaccia (`/** */`).
- 2 *Dichiarazione* della classe o dell'interfaccia.
- 3 *Commento generale*, se necessario (`/* */`).
- 4 *Class (static) 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*, raggruppati per funzionalità e non per scope o visibilità. L'obiettivo è quello di far comprendere il codice nel modo più facile e rapido.

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

Si

```
for (int i = 0; i < 8; i++) {  
    double angle = 2 * Math.PI / 8 * i;  
    // more code  
}
```

No

```
int i;  
for (i = 0; i < 8; i++) {  
    // code  
    double angle;  
    // more code  
    angle  
        = 2 * Math.PI / 8 * i;  
    // more code  
}
```

## Examples

```
if (testScore >= 70) {
    if (studentAge < 10) {
        System.out.println("Great job!");
    } else {
        // testScore >= 70 and age >= 10
        System.out.println("You did pass");
    }
} else {
    // test score < 70
    System.out.println("Fail");
}
```

## Examples

```
if (testScore < 70) {  
    System.out.println("Fail");  
    return;  
}  
  
if (age < 10) {  
    System.out.println("Great job!");  
    return;  
}  
  
// testScore >= 70 and age >= 10  
System.out.println("You did pass");
```

Esistono vari standard per il *casing*:

- PascalCase
- camelCase
- snake\_case
- CONSTANT\_CASE
- kebab-case

Oggetto	Case	Esempio
Package	Primo campo minuscolo	<code>java.awt.Class</code>
Classi	PascalCase	<code>class Raster</code>
Interfacce	PascalCase	<code>interface Serializable</code>
Metodi	camelCase	<code>void run()</code>
Variabili	camelCase	<code>int size</code>
Costanti	CONSTANT_CASE	<code>final int HEIGHT = 9</code>

Da ricordare:

- Nomi rappresentativi ed espressivi (no *thing*, *stuff*, e via). Si può fare eccezione per variabili temporanee (e.g., *i*, *j*).
- I metodi di solito hanno un *verbo*.
- Gli altri oggetti di solito sono dei *sostantivi*.

# Commenti e Documentazione

## Importante

È 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.

Nasce 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).

Tag	Descrizione
@author	Nome dello sviluppatore.
@deprecated	(vedere sopra) indica che l'elemento potrà essere eliminato da una versione successiva del software.
@exception	Indica eccezioni lanciate da un metodo; cf. @throws.
@link	Crea un collegamento ipertestuale alla documentazione locale o a risorse esterna (tipicamente internet).
@param	Definisce i parametri di un metodo. Richiesto per ogni parametro.
@return	Indica i valori di ritorno di un metodo. Questo tag non va usato per metodi o costruttori che restituiscono <i>void</i> .
@see	Indica un'associazione a un altro metodo o classe.
@since	Indica quando un metodo è stato aggiunto a una classe.
@throws	Indica eccezioni lanciate da un metodo. Sinonimo di @exception introdotto in Javadoc 1.2.
@version	Indica il numero di versione di una classe o un metodo.

`http://jautodoc.sourceforge.net/index.html`

Consente di

- Aggiungere nuova documentazione
- Completare la documentazione esistente
- Sovrascrivere la documentazione esistente

Si possono definire visibilità dei campi, specificare filtri e template, inserire header nei file, etc.

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

- Preferenze documentazione:  
Window → Preferences → Java → JAutoDoc
- Generazione documentazione:  
Right Click → JAutoDoc → Add JavaDoc
- Inserimento header:  
Right Click → JAutoDoc → Add header
- Generazione JavaDoc:
  - Right Click (Project) → Export → Java → JavaDoc
  - Può essere necessario specificare il path a javadoc.exe
  - Osserva anche le altre opzioni (fogli di stile, path, filtri, ...)

# Deployment con JAR

Un file JAR (Java ARchive) è un file che contiene le *classi*, le *immagini*, e tutti i vari *file* di una applicazione Java o di una applet, raccolti in un unico file e generalmente compressi.

Utilizzando un JDK (Java Development Kit), si ha sempre generalmente inclusa una utility chiamata `jar` che permette la creazione e l'estrazione di file da un singolo file JAR.

In una soluzione complessa, l'applicazione Java può essere realizzata anche da un insieme di file JAR. Un package Java open source può anche essere distribuito come file JAR.

Eclipse permette di generare un archivio JAR eseguibile partendo da un progetto eseguibile (dotato di metodo main):

- 1 Click pulsante destro su progetto → export → java → runnable jar file
- 2 Specificare il launch configuration da usare
- 3 Specificare il path di destinazione
- 4 Specificare come trattare le eventuali librerie richieste dall'applicazione

Una volta generato è possibile eseguire il jar con il comando

```
java -jar name.jar
```