

## Lezione 1

---

# Introduzione a Java (prima parte)

## Il linguaggio Java

---

- Nato nel maggio 95 (James Gosling & al.)
- Orientato ad oggetti, basato sulle classi, concorrente
- Pensato principalmente come linguaggio per produrre applicazioni speciali (*applets*) che girano all'interno di una pagina web, ma usato anche come linguaggio di programmazione general-purpose
- Disegnato in modo da massimizzare la portabilità del codice: la compilazione produce un codice binario (bytecode) indipendente dalla macchina, che gira sulla *Java Virtual Machine*
- Ad alto livello e progettato per poter eseguire codice su macchine remote in modo *sicuro*. Assenza di costrutti non sicuri (es. array senza controllo sugli indici)

## Applicazioni e Applets

---

### Applicazioni = programmi “stand-alone”

```
public class Ciao {  
    public static void main(String[] args) {  
        System.out.println("Ciao");  
    }  
}
```

### Applets = eseguibili da un browser Java-compatibile

```
import java.applet.Applet;  
import java.awt.Graphics;  
  
public class Ciao extends Applet {  
    public void paint(Graphics g) {  
        g.drawString("Ciao", 50, 25);  
    }  
}
```

```
<html>  
<applet code="Ciao.class" width=275 height=200> </applet>  
</html>
```

## Analisi dell'applicazione

---

```
public class Ciao {  
    public static void main(String[] args) {  
        System.out.println("Ciao");  
    }  
}
```

#### **class Ciao**

dichiara il nome della classe. Quando questo codice viene compilato, viene generato un file chiamato Ciao.class

#### **public**

il metodo main deve essere accessibile ovunque

#### **static**

il metodo main è legato alla classe Ciao, non ad una sua istanza: questo ne permette l'esecuzione prima che il programma faccia qualsiasi altra cosa

## Analisi dell'applicazione

---

```
public class Ciao {  
    public static void main(String[] args) {  
        System.out.println("Ciao");  
    }  
}
```

### **void**

il metodo main non restituisce nessun valore

### **String[] args**

il metodo main ha come parametro un array di stringhe (di dimensione qualsiasi). Questo permette di chiamare il programma con un numero qualsiasi (anche nullo) di parametri.

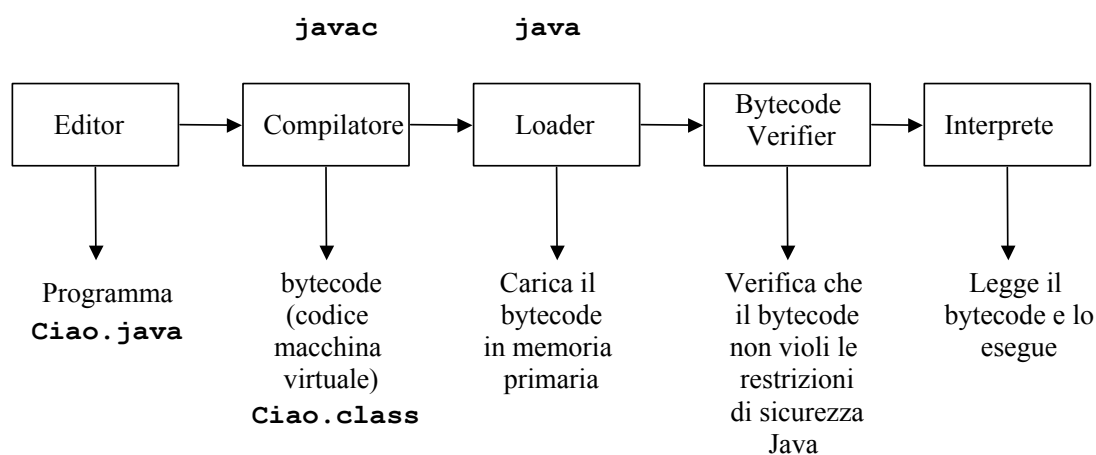
### **System.out.println**

chiamata al *metodo* println dell'*oggetto* out che appartiene alla *classe* System.

La chiamata a questo metodo produce la stampa sullo standard output (video) della stringa passata come parametro attuale.

## Ambiente Java

---



Compilazione: **javac Ciao.java**

Esecuzione: **java Ciao**

## Ambiente Java: requisiti

---

- Uso di un ambiente IDE : in laboratorio è stato installato Netbeans

Vantaggi: debugger, compilazione automatica

Svantaggi: non imparate a gestire le cose da soli !!!

- Uso di JDK (Java Development Kit <http://java.sun.com>) + un qualsiasi editor per scrivere i programmi + API (Application Programming Interface) di Java, disponibili su <http://www.dsi.unive.it/~labasd>

## Tipi e valori

---

Ogni variabile ed espressione ha un tipo conosciuto in compilazione, che ne limita i valori e l'uso.

Ci sono due categorie di tipi, in corrispondenza delle quali ci sono due categorie di valori:

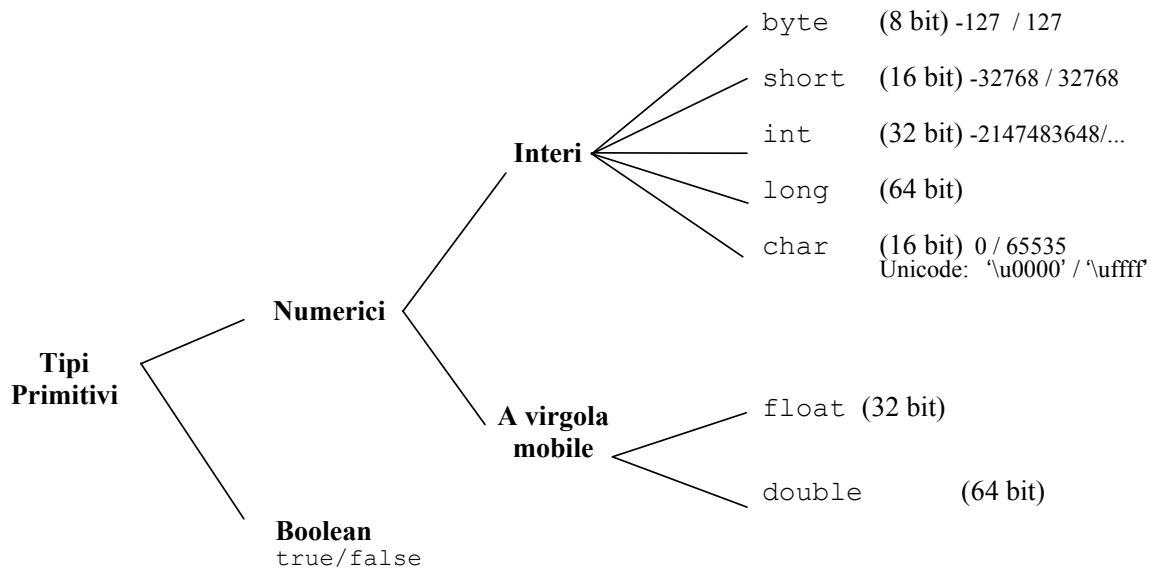
- tipi primitivi & valori primitivi
- tipi riferimento & valori riferimento (riferimenti ad oggetti: array, stringhe, classi, interfacce)

oltre a questi c'è un tipo speciale, il tipo nullo, che non ha nome ed ha come unico valore l'espressione *null*.

Tipi Primitivi:

- Devono essere definiti allo stesso modo in ogni macchina e in ogni implementazione
- una variabile di tipo primitivo ha sempre un valore primitivo dello stesso tipo.
- il valore di una variabile di tipo primitivo può essere modificato solo mediante assegnamento a quella variabile (o da operatori di incremento/decremento)

## Tipi primitivi



## Operatori su tipi primitivi

- operatori di confronto (valori di tipo boolean)  
<, <=, >=, >, ==, !=
- operatori numerici (valori di tipo numerico)
  - unari: +, -
  - additivi e moltiplicativi: +, -, \*, /, % (modulo)
  - di incremento e decremento ++, -- (prefissi e suffissi)
  - operatori di shift: <<, >>, >>>
- operatori logici
  - && (and), || (or), ! (not)
  - complemento bit a bit: ~
  - operatori interi bit a bit: & (and), | (or), ^ (or esclusivo)
- operatore condizionale (b ? e1 : e2)
- Casting

Attenzione: `i = i+1`; `i++`; `i += 1`; fanno la stessa cosa

## Operatori su tipi primitivi

---

- Operatori di shift (solo su tipi interi)
  - $a \ll b$  dà come risultato  $a \cdot 2^b$
  - $a \gg b$  dà come risultato  $a / 2^b$  (preserva il segno)
  - $\ggg$  “lavora” sulla sequenza di bit, mettendo zeri sui bit più significativi (non preserva il segno)
- Casting
  - Le variabili sono automaticamente “promosse” al tipo di dimensioni maggiori (ad esempio int viene promosso automaticamente a long)
  - negli altri casi serve un casting esplicito

```
long valoreLungo = 99L;
int ristretto = (int)valoreLungo; // casting esplicito
long valoreGrande=6           // casting automatico
float valorefloat=1.0f;
double valoredouble = (double)valorefloat;
```

## Underflow e Overflow

---

**Attenzione! gli operatori sugli interi non segnalano in nessun modo problemi underflow o di overflow**

```
public class Test {
    public static void main(String[] args) {
        int i = 1000000;
        System.out.println(i>10 ? ">10" : "<=10");
        System.out.println(i * i);
        long l = i;
        System.out.println(l * l);
    }
}
```

produce il seguente output:

```
>10
-727379968
1000000000000
```

## Numeri in virgola mobile

---

Usa lo standard IEEE 754-1985

- numeri positivi e negativi  $(-1)^s \cdot m \cdot 2^e$
- zero negativo e zero positivo,
- infinito positivo e negativo, es. `Float.POSITIVE_INFINITY`
- un valore speciale NaN (not a number), che rappresenta il risultato di operazioni non permesse. Ci sono quindi le costanti `Float.NaN` e `Double.NaN`

Ogni operazione con un valore NaN produce NaN, e ogni confronto con NaN produce false

Un overflow, sui numeri a virgola mobile, produce un infinito; un underflow produce 0.

La tabella riassume i possibili risultati delle operazioni di divisione e modulo tra numeri in virgola mobile

X	Y	X/Y	X%Y
Finite	+/-0	+/-∞	NaN
Finite	+/-∞	+/-0	X
+/-0	+/-0	NaN	NaN
+/-∞	Finite	+/-∞	NaN

## Esempio

---

```
public class Test {
    public static void main(String[] args) {
        double d = 1e308;
        System.out.println("overflow: " + d*10);

        System.out.println("meno infinito: " + Float.NEGATIVE_INFINITY);

        d = 0.0/0.0;
        System.out.println("0.0/0.0: " + d);

        System.out.print("risultati inesatti con numeri float:");
        for (int i = 0; i < 100; i++) {
            float z = 1.0f / i;
            if (z * i != 1.0f)
                System.out.print(" " + i);
        }

        d = 12345.6;
        System.out.print("\n casting: " + (int)d );
    }
}
```

**produce come risultato:**

```
overflow: INF
meno infinito: -INF
0.0/0.0: NAN
risultati inesatti con numeri float: 0 41 47 55 61 82 83 94 97
casting: 12345
```

## Esempio

---

### Esempio di dichiarazioni, inizializzazioni ed assegnamenti di variabili di tipi primitivi

```
public class Assign{
    public static void main(String[] args){
        static final double pigreco = 3.1416; // dichiara una costante
        int x,y; // dichiara due variabili intere
        float z=3.414f; // dichiara e inizializza una variabile float
        double w=3.1415; // dichiara e inizializza una variabile double
        boolean vero=false; // dichiara e inizializza una var. boolean
        char c; // dichiara una variabile char
        c='A'; // assegna un valore alla variabile di tipo char
        x=6;
        y=1000; // assegna valori alle variabili di tipo int
        ...
    }
}
```

### Esempio di assegnamenti non validi

```
y=pigreco // pigreco non è di tipo int (serve un casting!)
w=175,000 // la virgola non può apparire!
vero=1 // errore comune a programmatori C o C++
z=pigreco // un double non può essere assegnato a un float (è
// necessario il casting!)
```

## Comandi Java: if

---

**Sintassi:**    **If** (<boolean\_expr>  
                  <true\_statement>  
          **[else if** (<boolean\_expr>  
                  <else\_if\_statement>]  
          **[else**  
                  <else\_statement>]

```
public class Test{
    public decidi(int voto){
        if ( voto >= 18)
            System.out.println("Promosso");
        else{
            System.out.println("Bocciato");
            System.out.println("...peccato!");
        }
    }
}
```



## Comandi Java: switch

---

```
public class Toomany {
    public static void howMany(int k) {
        switch (k) {
            case 1: System.out.print("one ");
            case 2: System.out.print("two ");
            case 3: System.out.println("many");
        }
    }
    public static void main(String[] args) {
        howMany(3);
        howMany(2);
        howMany(1);
    }
}
```

**Produce:** many  
two many  
one two many

L'espressione `k` deve essere di tipo `char`, `byte`, `short`, `int`  
altrimenti il compilatore da errore

## Comandi Java: switch

---

```
public class Twomany {
    public static void howMany(int k) {
        switch (k) {
            case 1: System.out.println("one"); break;
            case 2: System.out.println("two"); break;
            case 3: System.out.println("many"); break;
            default: System.out.println("too many");
        }
    }
    public static void main(String[] args) {
        howMany(1);
        howMany(2);
        howMany(3);
        howMany(300);
    }
}
```

**Produce:** one  
two  
many  
too many

## Comandi Java: while & do ... while

---

**Sintassi:** **while** (<boolean\_expression>  
<loop\_body\_statement>

```
public class ContaCaratteri {
    public static void main(String[] args) throws java.io.IOException {
        int num = 0;
        while (System.in.read() != -1){
            num++;
        }
        System.out.println("Input di" + num + " caratteri.");
    }
}
```

---

**Sintassi:** **do**  
<loop\_body\_statement>  
**while** (<boolean\_expression>)

```
public class ScriviFinoAlCento{
    public static void main(String[] args){
        int i = 0;
        do {
            System.out.println(++i);
        } while (i < 100);
    }
}
```

## Comandi Java: for

---

**Sintassi:** **for** ([<inizialization>]; [<condition>]; [<increment>];)

```
public class ScriviFinoAlCentoA{
    public static void main(String[] args){
        for (int i=1; i<=100; i++){
            System.out.println(i);
        }
    }
}
```

---

```
public class ScriviFinoAlCentoB{
    public static void main(String[] args){
        for (int i=1; i<=100; ){
            System.out.println(i++);
        }
    }
}
```

---

```
public class ScriviFinoAlCentoC{
    public static void main(String[] args){
        for (int i=1; ; ){
            System.out.println(i++);
            if (i>100) break;
        }
    }
}
```

## Comandi Java: break & continue

---

**Sintassi:**    **break** [<label>]  
                  **continue**

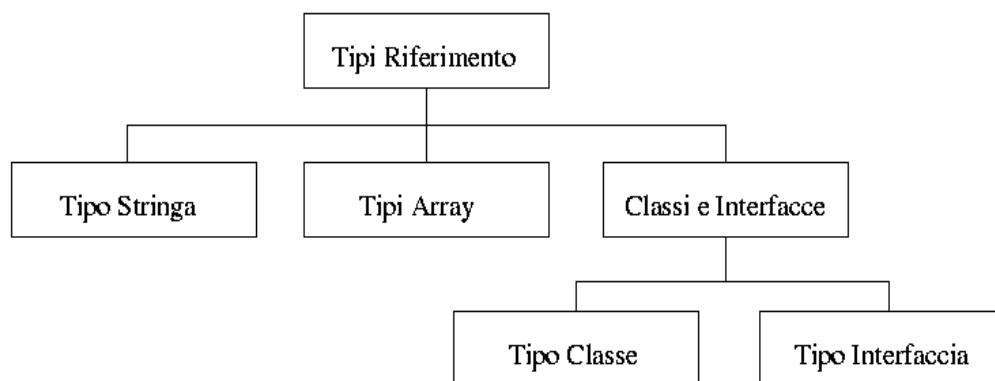
```
start:
  for (i=0; i<100; i++) {
    for (j=0; j<100; j++) {
      if (j == 77)
        continue;
      if (i+j == 50)
        break start;
      \\ ...altre operazioni ..
    }
  }
```

**break** esce dal comando switch/  
while/do/for più interno, a meno  
che non ci sia un'etichetta.  
L'etichetta non indica il punto di  
trasferimento del controllo

**continue** passa subito all'iterazione  
successiva nei comandi while/do/for

## Tipi riferimento

---



## Java: un linguaggio *orientato agli oggetti*

---

Gli “attori” principali dei programmi Java sono gli oggetti

- Gli oggetti memorizzano dati e forniscono operazioni (metodi) per accedere e manipolare tali dati
- Ogni oggetto è istanza di una classe che definisce il tipo di dati che l'oggetto memorizza e le operazioni possibili su tali dati
- Le classi sono quindi il modo Java per creare nuovi tipi

Una classe Java è definita da

- variabili o campi (fields) definite su tipi primitivi o su oggetti
- operazioni (methods)

## Classe = variabili (campi) + metodi

---

### Esempio:

```
public class Imp {
    String i_nome;          // variabili o campi della classe
    String i_cognome;
    String i_reparto;

    public Imp(String nome, String cognome, String reparto) {
        i_nome = nome;
        i_cognome = cognome;
        i_reparto = reparto;
    }

    public void stampa(){
        System.out.println(i_nome + " " +
            i_cognome + " del reparto " + i_reparto);
    }
}
```

## Costruttori

---

Il costruttore è un metodo “speciale” che viene usato per inizializzare gli oggetti appena creati. Il nome del metodo costruttore deve coincidere esattamente con il nome della classe, e non può restituire nessun valore.

```
public class Imp {
    String i_nome;           // variabili o campi della classe
    String i_cognome;
    String i_reparto;

    public Imp(String nome, String cognome, String reparto) {
        i_nome = nome; i_cognome = cognome; i_reparto = reparto;
    }

    public void stampa(){
        System.out.println(i_nome + " " +
            i_cognome + " del reparto " + i_reparto);
    }
}
```

## Creazione di un oggetto

---

Un'istanza di una classe (cioè un oggetto) viene esplicitamente creata usando l'operatore **new** applicato al costruttore della classe

Nell'esempio precedente per dichiarare, creare e inizializzare un oggetto della classe Imp:

```
Imp impiegato;
impiegato = new Imp("Giovanni", "Bianchi", "officina");
```

La chiamata al metodo `stampa()` invocato con

```
impiegato.stampa()
```

stamperà la stringa `Giovanni Bianchi del reparto officina`

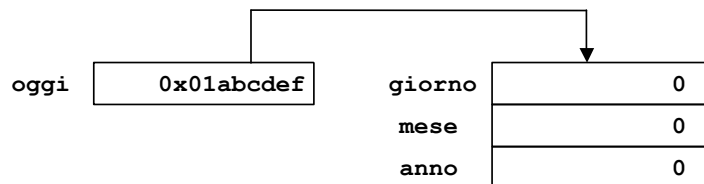
## Creazione di un oggetto

Quando viene dichiarata una variabile di tipo primitivo l'allocazione di memoria è implicita.

Quando viene dichiarata una variabile di tipo riferimento (array, stringhe, classi, interfacce) non viene allocata memoria per l'oggetto.

```
public class Data{
    int giorno;
    int mese;
    int anno;
}
oggi ???????
Data oggi;           // alloca memoria solo per il riferimento

oggi = new Data();   // alloca memoria per l'oggetto e inizializza
                    // le variabili
```



## Costruttori & creazione di un oggetto

L'uso dell'operatore **new** provoca le seguenti azioni:

- Un nuovo oggetto viene creato in memoria e tutte le sue variabili (instance variables) vengono inizializzate con valori standard (cioè **null** per gli oggetti, **false** per i boolean, **zero per tutti gli altri tipi primitivi**)
- Viene chiamato il metodo costruttore per il nuovo oggetto con i parametri specificati. Il costruttore esegue tutte le operazioni necessarie per inizializzare correttamente il nuovo oggetto
- Dopo il ritorno dal costruttore, l'operatore **new** ritorna il riferimento (cioè l'indirizzo di memoria) dell'oggetto appena creato. Se **new** compare a destra di un assegnamento, il riferimento viene memorizzato nella variabile che compare a sinistra dell'assegnamento

## Oggetti e assegnamenti

```
public class Value { int val; }

public class Test {
    public static void main(String[] args) {
        int i1 = 3;
        int i2 = i1;
        i2 = 4;
        System.out.print("i1=" + i1);
        System.out.println(" mentre i2=" + i2);

        Value v1 = new Value();
        v1.val = 5;
        Value v2 = v1;
        v2.val = 6;
        System.out.print("v1.val=" + v1.val);
        System.out.println(" e v2.val=" + v2.val);
    }
}
```

i1 i2  
■ ■  
due variabili distinte

v1 v2  
■  
stessa istanza

**produce come risultato:**

```
i1=3 mentre i2=4
v1.val=6 e v2.val=6
```

## Metodi: passaggio dei parametri

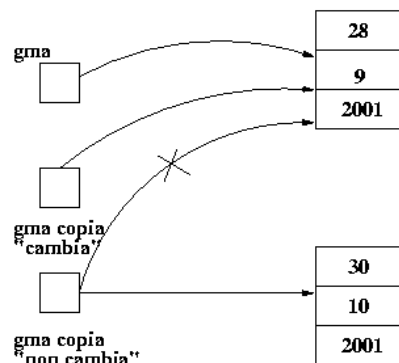
In un metodo il passaggio dei parametri è sempre per valore (cioè nel metodo si usano copie dei valori passati al chiamante)

Se si passa un oggetto, si passa il valore del suo riferimento e non l'oggetto stesso. Il contenuto dell'oggetto può essere modificato accedendo ai suoi campi e metodi. Non può però essere cambiato il riferimento all'oggetto stesso

```
public void noncambia(Data d) {
    d = new Data();
    d.giorno = 30; d.mese = 10; d.anno = 2001;
}

public void cambia(Data gma) {
    gma.giorno = 27; gma.mese = 9; gma.anno = 2003;
}

Data gma = new Data();
gma.giorno = 28; gma.mese = 9; gma.anno = 2001;
noncambia(gma);
cambia(gma);
```



Un metodo può ritornare un singolo valore. Per ritornare più valori si deve definire un oggetto apposito.

## Overloading dei nomi dei metodi

---

Nella stessa classe è possibile definire diversi metodi con lo stesso nome, purché differiscano nella lista degli argomenti.

Ad esempio,

```
public void println(int i);
public void println(float f);
public void println(String s);
```

Il tipo restituito può essere diverso, ma in ogni caso la lista degli argomenti deve essere diversa in modo tale da permettere una determinazione non ambigua del metodo che effettivamente viene chiamato.

## Overloading per i Costruttori

---

Possono esserci diversi costruttori (che differiscono per la lista di parametri), che possono invocarsi a vicenda

```
public class Impiegato{
    private String nome;
    private int salario;

    // pre: nome non nullo
    // post: costruisce un record Impiegato secondo i parametri
    public Impiegato(String nome, int salario){
        this.nome = nome;
        this.salario = salario;
    }
    // pre: n non nullo
    // post: costruisce un record Impiegato secondo i parametri
    public Impiegato(String n){
        this(n,0);
    }
    // post: costruisce un record Impiegato
    public Impiegato(){
        this("Sconosciuto", 0);
    }
}
```

La parola chiave **this** permette di effettuare una referenza implicita all'istanza corrente della classe (cioè all'oggetto) su cui un metodo



## Java e i puntatori

---

In Java non esiste il tipo di dato puntatore perchè si vogliono evitare i comportamenti pericolosi dei programmi:

- La gestione esplicita dei puntatori (come in C e C++) provoca problemi quando si libera in modo scorretto una zona di memoria
- L'aritmetica dei puntatori può provocare riferimenti a zone di memoria non valide

Nonostante l'assenza di puntatori "espliciti", Java si basa sui puntatori molto più di quanto avvenga in C o C++ :

La dichiarazione di un oggetto è sempre la dichiarazione di un riferimento ad un oggetto, che deve essere poi seguita da un'esplicita inizializzazione, attraverso l'uso di un costruttore o l'assegnamento di una variabile già inizializzata

```
Data data1, data2;  
data1 = new Data();  
data2 = data1;
```

La notazione con il punto per accedere a campi e metodi è in realtà un riferimento indiretto

```
data1.giorno // data1 non è il nome dell'oggetto ma il  
// riferimento al suo indirizzo
```

## Garbage Collector

---

Dopo aver usato new per allocare memoria per un oggetto, come si libera la memoria quando l'oggetto non serve più?

Java ha un meccanismo automatico di garbage collection: si tratta di un'applicazione che gira in background e tiene traccia dei riferimenti agli oggetti. Quando un oggetto non viene più riferito, il garbage collector provvede a liberare la memoria

## Il metodo main

---

Alcune classi sono progettate per essere usate da altre classi, mentre altre sono progettate per essere applicazioni stand-alone. Le classi che definiscono un'applicazione devono contenere il metodo **main**

```
public static void main(String[] args) {  
    // main method body  
}
```

Il metodo ha come parametro un array di stringhe che sono i parametri del programma, dati dalla linea di comando

Il metodo può essere usato per testare i metodi della classe

## Come documentare il codice Java

---

Quando si scrive una classe è importante allegare ad essa un'adeguata documentazione riguardo alle sue variabili e metodi, con relativi parametri e valori di ritorno. Questo permette di avere codice comprensibile e facilmente manutenibile.

Il programma **javadoc**, fornito con Java, permette di documentare automaticamente una classe java, purchè tale classe venga commentata in modo opportuno (comprensibile da javadoc).

Il comando **javadoc <nome file>.java** produce in uscita dei file HTML che possono essere visti e stampati mediante un browser.

I commenti javadoc sono dei blocchi che iniziano con `/**`, terminano con `*/` e ciascuna linea tra questi deve iniziare con un singolo asterisco `*`. Si assume che ciascun commento contenga una frase descrittiva seguita da una linea vuota, seguita poi da linee speciali che iniziano con un javadoc tag.

## Come documentare il codice Java

---

I javadoc tag più comuni sono:

**@author** <nome>

identifica l'autore della classe e deve precedere la definizione della classe stessa

**@param** <nome> <descrizione>

identifica un parametro e deve comparire prima della dichiarazione di un metodo

**@return** <descrizione>

descrive il tipo di ritorno di un metodo e deve comparire prima della sua dichiarazione

**@exception** <nome> <descrizione>

identifica una condizione di errore segnalata da un metodo

## Come documentare il codice Java: esempio

---

```
/**
 * Definisce un record con i dati dello studente
 *
 * @author Mario Bianchi
 * @version 1.0
 */

public class Student {
    private int S_mat;
    private String S_cognome;
    private String S_nome;

    /**
     * Costruisce il record per lo studente
     *
     * @param stud_mat      numero di matricola dello studente
     * @param stud_cognome  cognome dello studente
     * @param stud_nome     nome dello studente
     * @version 1.0
     */
    public Student(int stud_mat, String stud_cognome, String
stud_nome) {
        S_mat = stud_mat;
        S_cognome = stud_cognome;
        S_nome = stud_nome;
    }
}
```

## Come documentare il codice Java: esempio

```
/**
 * Ritorna il numero di matricola dello studente
 *
 * @return numero di matricola dello studente
 * @version 1.0
 */
public int Get_mat() {return S_mat;}

/**
 * Ritorna il cognome dello studente
 *
 * @return cognome dello studente
 * @version 1.0
 */
public String Get_cognome() {return S_cognome;}

/**
 * Ritorna il nome dello studente
 *
 * @return nome dello studente
 * @version 1.0
 */
public String Get_nome() {return S_nome;}
}
```

## Come documentare il codice Java: esempio

### Class Student

```
java.lang.Object
|
+--Student
```

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

Definisce un record con i dati dello studente

#### Constructor Summary

[Student](#)(int stud\_mat, java.lang.String stud\_cognome, java.lang.String stud\_nome)  
Costruisce il record per lo studente

#### Method Summary

java.lang.String	<a href="#">Get_cognome</a> () Ritorna il cognome dello studente
int	<a href="#">Get_mat</a> () Ritorna il numero di matricola dello studente
java.lang.String	<a href="#">Get_nome</a> () Ritorna il nome dello studente

#### Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

# Come documentare il codice Java: esempio

---

## Constructor Detail

### Student

```
public Student(int stud_mat,  
              java.lang.String stud_cognome,  
              java.lang.String stud_nome)
```

Costruisce il record per lo studente

**Parameters:**

stud\_mat – numero di matricola dello studente  
stud\_cognome – cognome dello studente  
stud\_nome – nome dello studente

## Method Detail

### Get\_mat

```
public int Get_mat()
```

Ritorna il numero di matricola dello studente

**Returns:**

numero di matricola dello studente

---

### Get\_cognome

```
public java.lang.String Get_cognome()
```

Ritorna il cognome dello studente

**Returns:**

cognome dello studente

---

### Get\_nome

```
public java.lang.String Get_nome()
```

Ritorna il nome dello studente

**Returns:**

nome dello studente