Hello World in Java

```
1 /**
2 * Hello World: versione senza oggetti
3 *
4 * @author Pinco Pallo
5 * @version 9/3/2010
6 */
7 public class Hello0
8 {
9       // variabile di classe
10      static private String s = "Hello,  world" ;
11
12      /**
13      * Il main (metodo di classe)
14      */
15      public static void main(String[] args)
16      {
17              // stampa la stringa
18              System.out.println(s + " ! " );
19      }
20 }
```


Hello World in C

```
1 /**
2 * Hello-World
3 *
4 * @author Pinco Pallo
5 * @version 9/3/2010
6 */
7
8 #include <stdlib.h>
9
10 /* variabile con la stringa da stampare */
11 char *s = "Hello,  world!";
12
13 int main(int argn, char *argv[])
14 {
15       /* stampa la stringa */
16      printf(s);
17 }
```


Compilare ed Eseguire

Per compilarlo

In *C* `> gcc hello0.c –o hello`

In Java `> javac Hello0.java`

Per eseguirlo

In *C* `> ./hello`
`Hello,  world!`

In Java `>java Hello0`
`Hello,  world!`

Main (1)

In entrambi i casi viene eseguito il codice di una funzione di nome main

In *C* `13 int main(int argn, char *argv[])`

In Java `15  public static void main(String[] args)`

Il `main` del programma in *C* non è parte di nessun blocco, mentre quello in Java è all'interno di una definizione di classe.

In Java le funzioni sono dette metodi e devono essere definite in una classe.

I concetti di classe ed oggetti sono nuovi in Java rispetto a *C*.

Main (2)

Come in *C*, in Java può esserci solo un metodo di nome `main`.

In Java, riceve come input un array di stringhe (tipo **`String[]`**) e non restituisce nulla (tipo **`void`**) .

(con l'overloading, possono esserci altri metodi di nome main che devono ricevere o restituire parametri di tipo o in numero diversi)

Il `main` deve essere dichiarato `public static` .

`public` indica che il metodo è visibile da tutti;

`static` indica che il metodo è associato alla classe e non ai suoi oggetti

Anche la classe che contiene il main deve essere **`public`**.

In un file puo esserci solo una classe public e il suo nome deve coincidere con quello del file.

Per convenzione, i nomi delle classi iniziano con una lettere maiuscola, mentre tutti gli altri nomi iniziano con una lettera minuscola.

String

La classe `Hello0` contiene anche la definizione della variabile con la stringa da stampare (questa variabile è globale nel programma *C*)

`10  static private String s = "Hello,  world" ;`

La definizione della stringa è preceduta da `static private`.

- `private` indica che la variabile è visibile solo all'interno della classe dove è definita;
- `static` che si tratta di una variabile di classe, comune a tutti gli oggetti della classe, che esiste indipendente dall'esistenza degli oggetti della classe stessa

Notare che in Java abbiamo un tipo `String` per rappresentare le stringhe, che sono oggetti istanze della classe `String`.

L'assegnazione alla variabile della stringa "`Hello, world!`" crea un nuovo oggetto della classe `String` contenente questa stringa ed associa alla variabile un riferimento (un puntatore) a questo oggetto.

Librerie

La funzione `printf` che stampa una stringa fa parte della libreria standard del C. Per poterla utilizzare abbiamo incluso `stdlib.h`.

In Java, per stampare la stringa abbiamo scritto

```
18          System.out.println(s + " ! " );
```

- `System` è una classe della librerie della *piattaforma Java*;
- `out` è una variabile pubblica che contiene un oggetto che rappresenta lo standard output ;
- `println` è un metodo dell'oggetto out che riceve come parametro una stringa e la stampa sullo standard output (aggiungendo un carattere di fine linea).

Notare l'operatore di somma su stringhe che corrisponde alla concatenazione di stringhe.

Piattaforma Java

Con piattaforma Java (Java platform) si intende
- l'insieme delle librerie disponibili in ogni installazione Java; (ovvero)
- l'insieme delle API (Application Programming Interface) fornite da Java.

Questo insieme di librerie
- è molto esteso;
- copre gran parte delle strutture dati di base;
- è indipendente dall'architettura o dal sistema operativo.

Permette di realizzare programmi complessi e con funzionalità normalmente dipendenti dalla API fornite da un particolare SO senza far riferimento a nessun sistema operativo o architettura.

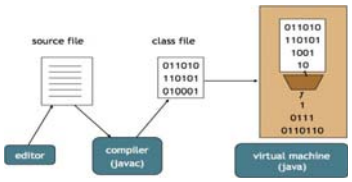
Un'applicazione scritta per la piattaforma Java può essere eseguita senza cambiamenti su un qualsiasi sistema operativo che supporta la piattaforma Java, ovvero sulla quale è disponibile la cosiddetta Java Virtual Machine.

Compilazione ed Esecuzione: JVM

Il compilatore java (comando `javac`) compila le classi contenute nel file sorgente e per ogni classe "NomeClasse" produce un file "NomeClasse.class" contenente il codice Java Bytecode della classe.

Java Bytecode è il linguaggio eseguito (piu correttamente interpretato) dalla Java Virtual Machine o JVM.

Il comando `java` invoca la Java Virtual Machine e fornisce come input una classe. La JVM comincia l'esecuzione chiamando il metodo `main` della classe, segnalando un errore nel caso in cui tale metodo non sia presente, o non sia visibile (public).



Commenti

```
1 /**
2 * Hello World: versione senza oggetti
3 *
4 * @author Pinco Pallo
5 * @version 9/3/2010
6 */
 ...
12     /**
13     * Il main (metodo di classe)
14     */
 ...
17            // stampa la stringa
```

Commenti principali aperti da `/**` e chiusi da `*/` o come in C++ (riga 17).

Notare anche `@author` e `@version` utilizzati per generare la documentazione della classe chiamando il programma javadoc

Hello World con oggetti

```
1 /**
2 * Class Hello World: versione con oggetti
3 *
4
5 */
6 class Hello
7 {
8       // il metodo fa il lavoro
9       public void go()
10      {
11              System.out.println( "Hello,  world");
12      }
13      /** Il main (metodo di classe)
14      */
15      public static void main(String[] args)
16      {
17              Hello hi = new Hello();
18              hi.go();
19      }
20 }
```


Greeter: classe per generare saluti

```
1 /**
2 * Classe Greeter per generare saluti
3 */
4
5 public class Greeter
6 {
7   /** costruisce un oggetto per salutare una persona.
8       @param aName è il nome della persona da salutare
9       */
10      private String name;
11
12      public Greeter(String aName)
13      {
14              Name = aName;
15      }
16      public String sayHello()
17      {
18              Return "Helli, " + name + "!";
19      }
20 }
```

Il costruttore di Greeter

Metodo **Greeter** (stesso nome della classe) definito nella classe:

- per questo metodo non è specificato il tipo del valore restituito;
- un metodo con lo stesso nome della classe è detto costruttore;
- un costruttore restituisce sempre un oggetto della classe corrispondente;
- questo costruttore prevede una stringa come parametro di input.

Per una stessa classe si possono avere costruttori con diverse sequenze di parametri di input;

Un costruttore viene chiamato SOLO attraverso il comando **new**, ad esempio come in `new Greeter("World'')`

- Se non è specificato nessun costruttore, allora la classe NomeClasse ha il costruttore di default NomeClasse() senza parametri di input (per esempio, la classe Hello precedentemente definita);
- Se si definisce un costruttore esplicito, il costruttore di default non è più accessibile.

Per provare la classe Greeter

```
1 /**
2 * Classe GreeterTester per provare Greeter
3 */
4
5 public class GreeterTester
6 {
7
8       public static void main(String[] args)
9       {
10            Greeter worldGreeter = new Greeter("world");
11            String greeting = worldGreeter.sayHello();
12            System.out.println(greeting);
13       }
14 }
```

La classe crea un nuovo oggetto con **new**, esegue il metodo **sayHello** e poi stampa la string prodotta.

Gli Oggetti

Un oggetto è un pezzo di software cui possiamo associare uno stato e un comportamento.

Gli oggetti sono usati per modellare entità reali o astratte (ad esempio, una monovolume di colore rosso, o una lista di interi maggiori di zero).

Gli oggetti sono creati dinamicamente (in Java dal comando `new`) e tutti gli oggetti hanno un proprio stato. (In Java non c'e bisogno di distruggere gli oggetti, che spariscono automaticamente per garbage collection quando non piu utilizzati).

Lo stato di un oggetto è rappresentato dalle sue variabili, le cosiddette variabili di istanza o attributi dell'oggetto.

Il comportamento degli oggetti è modellato dalle funzioni ad essi associati, i metodi dell'oggetto.

I metodi possono avere dei parametri per ricevere informazioni necessarie alla loro esecuzione.

Metodi e Messaggi (1)

Nella programmazione OO, anzichè parlare di invocazione dei metodi, si usa spesso la metafora del passaggio di messaggi:

`System.out. println ("Hello , world !'')`

invia il messaggio corrispondente a println all'oggetto System.out;

il messaggio porta una stringa (Hello, world!) come informazione aggiuntiva.

In pratica, i metodi hanno sempre un parametro implicito: l'istanza dell'oggetto che riceve il messaggio.

Metodi e Messaggi (2)

- I metodi public corrispondono ai messaggi che un oggetto puo ricevere.
- Un oggetto puo avere metodi private non visibili fuori dell'oggetto, ma utilizzati dagli altri suoi metodi.
- Un oggetto risponde alla ricezione di un messaggio con un'azione (l'esecuzione del codice del corrispondete metodo) e restituendo un valore (il valore di ritorno del metodo).
- I metodi possono accedere allo stato dell'oggetto che riceve il messaggio (possono leggerlo e/o modicarlo)

Lo stato di un oggetto (1)

- Lo stato di un oggetto è rappresentato dal valore dei suoi attributi.
- Un attributo o variabile d'istanza è una variabile dichiarata all'interno di una classe e non preceduta dall'attributo `static` .
- Gli attributi possono essere `public` e quindi accessibili (in lettura e scrittura) anche all'esterno dell'oggetto.
  - Ad esempio, se l'oggetto `obj` ha un attributo `public attribute` , si puo accedere all'attributo con `obj.attribute`
- Gli attributi `private` sono visibili solo all'interno e possono essere letti/modicati solo dai suoi metodi.
- Dichiarando `private` gli attributi di un oggetto, si nasconde il suo stato all'esterno (information hiding).

Lo stato di un oggetto (2)

Si possono però prevedere metodi appositi per accedere a tutto o parte dello stato di un oggetto o per ottenere il valore di un attributo composto calcolato a partire dallo stato dell'oggetto.

Nascondendo gli attributi effettivamente dichiarati all'interno di una classe e rendendo l'oggetto accessibile solo attraverso alcuni metodi, possiamo astrarre rispetto all'effettiva implementazione dell'oggetto e mostrare all'utente un oggetto con uno stato e un comportamento indipendenti da inutili dettagli implementativi.

Le Classi (1)

- Le classi rappresentano gli oggetti di un certo tipo.
- Le classi sono una sorta di schema o prototipo dal quale creare gli oggetti.
- I metodi e gli attributi `static` sono comuni a tutti gli oggetti di una classe e sono quindi detti variabili o metodi della classe. Ad esempio, out è una variabile `static` della classe `System` ed essendo `public` è accessibile con `System.out`;
- la classe Math di Java ha vari metodi static per il calcolo delle più comuni funzioni matematiche, tra cui il metodo

```
int abs( int a)
```

per il calcolo del valore assoluto di un intero, che puo essere chiamato con

```
Math.abs(i)
```

(se `i` è un oggetto di tipo `int`)

Le Classi (2)

Quando si definisce una classe si definisce automaticamente anche il tipo degli oggetti di quella classe, o meglio dei riferimenti a oggetti della classe.

Ad esempio, in

```
Greeter worldGreeter = new Greeter("World'');
```

la variabile worldGreeter ha tipo Greeter, ovvero contiene un riferimento a un oggetto della classe Greeter.

Anche **new** restituisce un riferimento a un oggetto della classe del costruttore che segue **new**.

Ci sono differenti usi del nome di una classe:

- per definire la classe e il file ad essa associato;
- per definire i costruttori della classe;
- per il tipo dei riferimenti agli oggetti della classe.