

Principi di OOP con JAVA

lezioni

08 - gen - 2004

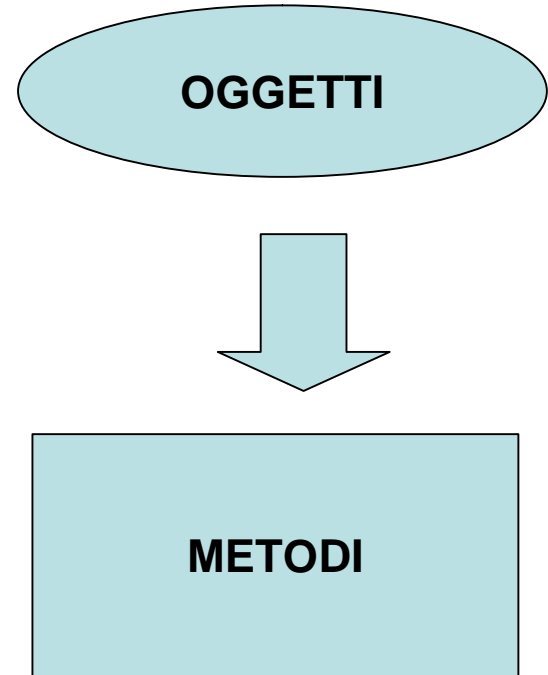
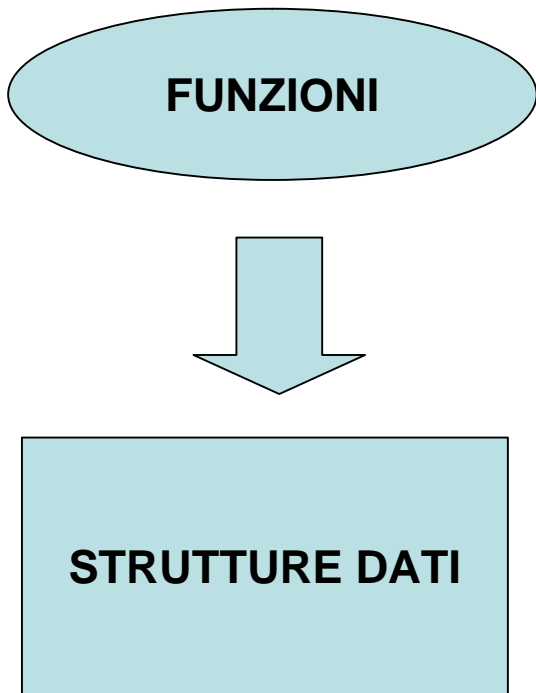
15 - gen - 2004

Tommaso Di Noia

t.dinoia@poliba.it

OOP
(elementi)

Programmazione Procedurale e Orientata agli Oggetti.



Principi di OOP [Alan Kay](1)

- **Qualsiasi cosa è un OGGETTO.**
 - Si può pensare ad un oggetto come ad una “strana” variabile. Memorizza dati, ma ...
 - È possibile “fare delle richieste” a tale oggetto chiedendogli di eseguire operazioni su se stesso.

Principi di OOP [Alan Kay]

(2)

- **Un programma è un insieme di oggetti che, tramite messaggi, si dicono l'un l'altro cosa fare**
 - Per fare una richiesta ad un oggetto, si “mandano messaggi” a tale oggetto.
 - Più concretamente si può pensare ad un messaggio come ad una richiesta per richiamare un metodo che appartiene ad un particolare oggetto.

Principi di OOP [Alan Kay]

(3)

- **Ogni oggetto ha una propria memoria composta da altri oggetti**
 - È possibile creare un nuovo tipo di oggetto utilizzando altri oggetti già presenti.
 - In tal modo è possibile scrivere programmi complessi nascondendo tale complessità nella semplicità dei singoli oggetti.

Principi di OOP [Alan Kay]

(4)

- **Ogni OGGETTO ha un tipo.**
 - Ogni oggetto è una **istanza** di una **classe**.
Dove **classe** è sinonimo di **tipo**
type ≡ class
 - La caratteristica più importante di una **classe** è: Quali messaggi è possibile mandargli?
Come è possibile interagire con essa?

Principi di OOP [Alan Kay]

(5)

- **Tutti gli OGGETTI dello stesso tipo possono ricevere gli stessi messaggi.**
 - Ex. Poiché un oggetto di tipo “circle” è anche un oggetto di tipo “shape”, un oggetto che istanzia “circle” sicuramente può accettare anche messaggi per oggetti che istanziano il tipo (classe) “shape”.

Oggetto [Booch]

- Identità: espressa da un nome
- Stato: include le proprietà dette *attributi* che descrivono gli oggetti
- Comportamento: rappresentato da funzioni dette *metodi* che utilizzano o cambiano il valore degli attributi

Concetti Object Oriented

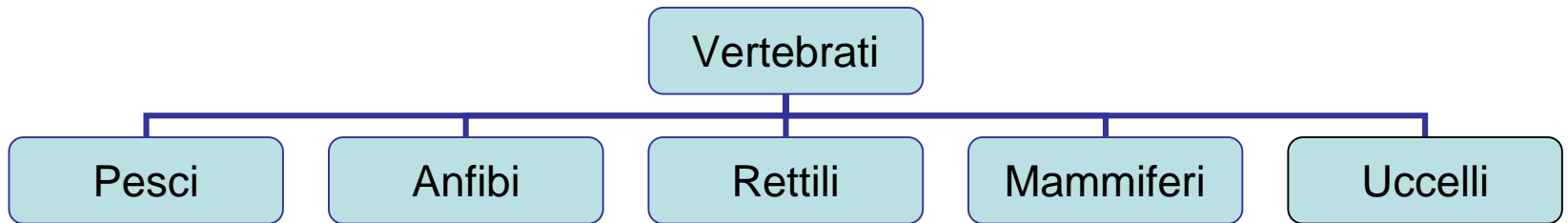
- Oggetto: una entità del mondo reale
- Classe: un insieme di oggetti aventi le stesse caratteristiche
- Attributi: proprietà di classi ed oggetti che ne definiscono le caratteristiche
- Metodi: definiscono il comportamento di tutti gli oggetti appartenenti alla stessa classe
- Operazioni: definiscono il comportamento degli oggetti istanze di una classe

Caratteristiche dell'OOP

- Ereditarietà:
 - ciascuna classe può essere definita in termini di una classe esistente. La nuova classe (sottoclasse) contiene automaticamente la definizione di elementi propri della classe originaria (superclasse)
- Polimorfismo:
 - pluralità di forme, gli oggetti possono ridefinire le operazioni della classe di cui fanno parte
- Information Hiding (incapsulamento):
 - ciascuna classe nasconde al proprio interno i dettagli implementativi

Esempio: ereditarietà

Classificazione delle specie animali



Classe

- Un insieme di oggetti aventi le stesse caratteristiche. E' caratterizzata da:
 - Identità: definisce il nome della classe
 - Attributi: la classe non ha stato, ma definisce proprietà locali che sono l'astrazione delle proprietà comuni agli oggetti istanze della classe
 - Metodi: definiscono il comportamento della classe. Rappresentano i servizi che possono essere richiesti da un oggetto. I metodi sono implementazioni delle operazioni

Visibilità delle proprietà

Una classe è concettualmente divisa in due parti:

- Una parte visibile che fornisce l'unico modo tramite il quale è possibile operare sugli oggetti della classe e descrive che cosa, in termini di operazioni ammissibili è possibile fare sugli oggetti
- Una parte nascosta il cui contenuto non è visibile all'esterno della classe e che riguarda come le funzionalità visibili sono realizzate

JAVA

Distribuzioni (java.sun.com)

- J2SE (Java 2 Standard Edition):
 - Mette a disposizione un compilatore, vari tool per la gestione del codice, un appletviewer ed un ricco set di API per lo sviluppo di applicazioni e applet.
- J2EE (Java 2 Enterprise Edition):
 - Orientata allo sviluppo di applicazioni distribuite. Mette a disposizione un insieme di semplici tool per la gestione di un application server ed un semplice DBMS (cloudscape) ed un ricco set di API per lo sviluppo di servlet e di tutte le tecnologie che ruotano attorno ad esse.
- J2ME (Java 2 Micro Edition):
 - Un ambiente altamente ottimizzato per i dispositivi con scarsa disponibilità di risorse hardware (smart card, telefonini, palmari). Mette a disposizione un ricco set di API per lo sviluppo di midlet ed alcuni tool di gestione (tra i quali vari simulatori di dispositivi di telefonini).

Documentazione su java

Ambienti di sviluppo integrato (IDE):

- NetBeans IDE
- Borland JBuilder
- Sun One Studio (non più supportato da SUN)

Documentazione su tutte le librerie presenti nelle distribuzioni:

- Java Doc – documentazione in formato ipertestuale

Java - Caratteristiche

- **Portabile:** non prevede aspetti dipendenti dalla implementazione
es. le dimensioni dei tipi predefiniti sono fisse
- **Indipendente dall'architettura:** il compilatore genera un bytecode interpretabile in processori diversi (Java è sia *compilato* che *interpretato*)
- **Robusto:** il compilatore riscontra molti errori che in altri linguaggi sono riscontrabili solo in esecuzione
- **Sicuro:** è destinato all'uso in ambienti distribuiti e di rete
- **Altamente versatile:** grazie ad un ricco set di API è possibile scrivere qualsiasi tipo di applicazione.

Il bytecode

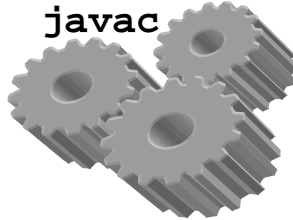
file **HW.java**

```
public class HW {  
    public HW() {  
    }  
    public static  
    void main(String[]  
    args) {  
  
        System.out.println  
        ("Hello,  
        WORLD!");  
    }  
  
}
```

codice sorgente

compilatore

javac



bytecode

file **HW.class**

```
0 0:getstatic #2 <Field  
PrintStream System.out>  
1 3:ldc1 #3 <String "lo ne  
ho viste di cose che voi umani\nnon  
potreste nemmeno immaginarvi.">  
2 5:invokevirtual #4 <Method  
void PrintStream.println(String)>  
3 8:return
```

Hello, WORLD!

symbian



interprete

java



JAVA

Classi e Oggetti in java

- Classe è l'unità fondamentale dei programmi java
 - definisce la struttura degli oggetti
 - contiene campi e metodi
- Il file sorgente .java deve avere lo stesso nome della classe.
- Sintassi per la definizione di una classe:
 - *ModificatoreAccesso* **class** Nomeclasse
 {//definizione dei campi
 // definizione dei metodi }
- Creazione di un oggetto ↔ creazione di un'istanza della classe
- Sintassi per la creazione di un oggetto
 - *NomeClasse* oggetto = **new** *CostruttoreClasse*();

Metodo costruttore

Esempio: HW.java

```
public class HW {  
    public HW() {  
    }  
    public static void main(String[] args){  
        System.out.println("Hello,WORLD!");  
    }  
}
```

Metodi in java

- Sintassi per definire un metodo
 - `ModificatoreAccesso tipo_restituito nome_metodo (Lista argomenti)`
`{ // dichiarazioni e statement }`
- Sintassi per chiamare un metodo:
 - `oggetto.metodo(argomenti)`
- Metodi *costruttori*
 - usati per inizializzare gli oggetti di una classe
 - **new costruttore** permette di creare un nuovo oggetto nella classe
 - hanno lo stesso nome della classe
 - se il codice della classe non specifica alcun costruttore java fornisce *costruttori predefiniti* - senza argomenti - che inizializzano i valori a quelli predefinti

Esempio (1) : Useless.java

```
public class Useless {
    private String modificata;
    private String originale;
    public Useless(String org) {
        originale = org;
        modificata = originale+"QUESTA E' LA STRINGA
                                modificata";

        System.out.println("E' stato creato un nuovo
        oggetto appartenente alla classe Useless \n");
    }
    public void showModificata(){
        System.out.println(modificata);
    }
    public void showOriginale(){
        System.out.println(originale);
    }
}
```

Esempio (2) :

UseUseless.java

```
public class UseUseless {
    public UseUseless() {
    }
    public static void main(String[] args) {
        Useless oggetto1 = new Useless("primo
            oggetto inutile creato \n");
        oggetto1.showOriginale();
        Useless oggetto2 = new Useless("secondo
            oggetto inutile creato \n");
        oggetto2.showModificata();
    }
}
```

Esempio (3) : UseUseless

```
java UseUseless
```

```
Useless oggetto1 = new Useless("primo oggetto inutile creato\n");  
oggetto1.showOriginale();
```

```
oggetto1:  
Useless
```

```
originale = "primo oggetto  
inutile creato \n"
```

```
E' stato creato un nuovo oggetto appartenente  
alla classe Useless
```

```
primo oggetto inutile creato
```

```
Useless oggetto2 = new Useless("secondo oggetto inutile creato \n");  
oggetto2.showModificata();
```

```
oggetto2:  
Useless
```

```
originale = "secondo oggetto  
inutile creato \n"
```

```
E' stato creato un nuovo oggetto appartenente  
alla classe Useless
```

```
secondo oggetto inutile creato  
QUESTA E' LA STRINGA modificata
```

Metodi in java

- Sintassi per definire un metodo
 - ModificatoreAccesso tipo_restituito nome_metodo (Lista argomenti)
{ // dichiarazioni e statement}
- Sintassi per chiamare un metodo:
 - oggetto.metodo(argomenti)
 - OSS: Se l'argomento è un oggetto, questo viene passato per riferimento.
- Metodi *costruttori*
 - usati per inizializzare gli oggetti di una classe
 - **new costruttore** permette di creare un nuovo oggetto nella classe
 - hanno lo stesso nome della classe
 - un costruttore non restituisce valore.
 - se il codice della classe non specifica alcun costruttore java fornisce *costruttori predefiniti* - senza argomenti - che inizializzano i valori a quelli predefinti

Gli Oggetti in Java (ed altri particolari) (1)

- A differenza di altri linguaggi di programmazione orientati agli oggetti (es: C++, SmallTalk, Object Pascal) in Java l'allocazione degli oggetti è sempre dinamica, tutto viene caricato nella memoria heap.
- Non esiste un metodo distruttore. Il **garbage collector** della JVM si occupa di distruggere gli oggetti che non sono più referenziati. (La chiamata al **garbage collector** può essere forzata dall'utente)

Gli Oggetti in Java (ed altri particolari) (2)

- Poiché non sono disponibili i puntatori (croce e delizia di ogni programmatore) per creare le strutture dati dinamiche (alberi, liste, code,...) sono messe a disposizione delle classi container che si occupano di gestire tali strutture (**List**, **Map**, **Set**).
 - Per gestire tali strutture esistono le classi iterator che permettono di separare la struttura dai metodi per accedervi.