Classi Java

Le classi estendono il concetto di "struttura" di altri linguaggi

Definiscono:

- I dati (detti campi o attributi)
- Le azioni (dette metodi o funzioni membro) che sui dati agiscono

Possono essere definite

- Dal programmatore (ex. Automobile)
- Dall'ambiente Java (ex. String, System, etc.)

La "gestione" di una classe avviene mediante

- Definizione della classe
- Instanziazione di Oggetti della classe

Definizione di una Classe

Definizione

```
class <nomeClasse) {
<campi>
<metodi>
}
```

Esempi

- Classe contenente dati ma non azioni

```
class DataOnly {
boolean b;
char c;
int i;
float f;
}
```

- Classe contenente dati e azioni

```
class Automobile {

    //Attributi
```

```
    String colore;
    String marca;
    boolean accesa;

    //metti i in moto
    void mettiInMoto() {
        accesa = true;
    }

    //vernicia
    void vernicia (String nuovoCol) {
        colore = nuovoCol;
    }

    // stampaStato
    void stampaStato () {
        System.out.println("Questa automobile è una "+ marca + " " +
colore);
        if (accesa)
        System.out.println("Il motore è acceso");
        else
        System.out.println("Il motore è spento");
    }
}
```

Dati & Metodi

- Public: visibili all'esterno della classe
- Private: visibili solo dall'interno della classe
- Protected: ...
- Nessuna specifica (amichevole): ...

La definizione di classe non rappresenta alcun oggetto.

Polimorfismo (overloading)

Una classe può avere più metodi con lo stesso nome

I metodi devono essere distinguibili in base a

- Numero dei parametri
- Tipo dei parametri

Il metodo da eseguire viene scelto in base a

- Numero e tipo di parametri

Il metodo da eseguire **NON** viene scelto in base al valore di ritorno

Esempio

```
class Automobile {
String colore;
void vernicia () {
colore = "bianco";
}
void vernicia (int i) {
switch (i) {
case 1: colore = "nero"; break;
...
}
}
void vernicia (String nuovoCol) {
colore = nuovoCol;
}
}
```

Instanziare una Classe

Crea degli oggetti appartenenti a una classe

Gli oggetti sono caratterizzati da

- Classe di appartenenza - tipo (ne descrive attributi e metodi)
- Stato (valore attuale degli attributi)
- Identificatore univoco (reference - handle - puntatore)

Per creare un oggetto occorre

- Dichiarare una istanza
- Tecnica analoga a quella utilizzata per i tipi primitivi
- La dichiarazione non alloca spazio ma solo una riferimento (puntatore) che per default vale null
- Allocazione e inizializzazione
- Riservano lo spazio necessario creando effettivamente l'oggetto appartenente a quella classe

Creare un Oggetto

Il costrutto new

- Crea una nuova istanza della classe specificata, allocandone la memoria
- Restituisce il riferimento all'oggetto creato

```
Automobile a = new Automobile ();
Motorcycle m = new Motorcycle ();
String s = new String ("ABC");
```

- Chiama il costruttore del nuovo oggetto (vedere in seguito)

Utilizzo della Classe

Per "gestire" una classe occorre

- Accedere ai metodi della classe
- Accedere agli attributi della classe

Messaggi

L'invio di un messaggio provoca l'esecuzione del metodo

Inviare un messaggio ad un oggetto

- Usare la notazione "puntata" oggetto.messaggio(parametri)
- Sintassi analoga alla chiamata di funzioni in altri linguaggi

I metodi definiscono l'implementazione delle operazioni

I messaggi che un oggetto può accettare coincidono con i nomi dei metodi

- p.es mettiInMoto, vernicia, etc.

I messaggi includono i parametri

- vernicia("Rosso")

Esempi

```
Automobile a = new Automobile();
a.mettiInMoto();
a.vernicia("Blu");
```

Caso particolare

- Metodi che devono inviare messaggi allostesso oggetto cui appartengono
- In questo caso la notazione puntata è superflua in quanto è sottinteso il riferimento

```
public class Libro {
int nPagine;
public void leggiPagina (int nPagina) {...}
public void leggiTutto () {
for (int i=0; i<nPagine; i++)
leggiPagina (i);
}
}
```

Attributi

Stessa notazione "puntata" dei messaggi oggetto.attributo

- Il riferimento viene usato come una qualunque variabile

```
Automobile a=new Automobile();
a.colore = "Blu";
boolean x = a.accesa;
10* I metodi che fanno riferimento ad
attributi dello stesso oggetto possono
tralasciare il rif-oggetto
public class Automobile {
String colore;
void vernicia(){
colore = "Verde";
// colore si riferisce
// all'oggetto corrente
}
}
```

- Esempio (messaggi e attributi)

```
public class Automobile {
String colore;
public void vernicia () {
colore = "bianco";
}
public void vernicia (String nuovoCol) {
colore = nuovoCol;
}
}
...
Automobile a1, a2;
a1 = new Automobile ();
a1.vernicia ("verde");
a2 = new Automobile ();
```

Costruttore

Specifica le operazioni di inizializzazione (attributi, etc.) che vogliamo vengano eseguite su ogni oggetto della classe appena viene creato

Tale metodo ha

- Lo stesso nome della classe
- Tipo non specificato

Non possono esistere attributi non inizializzati

- Gli attributi vengono inizializzati comunque con valori di default

Se non viene dichiarato un costruttore, ne viene creato uno di default vuoto e senza parametri

Spesso si usa l'overloading definendo diversi costruttori

Si osservi che la distruzione di oggetti (garbage-collection) non è a carico del programmatore

Esempio (costruttori con overloading)

```
Class Window {
String title;
String color;
// Finestra senza titolo nè colore
Window () {
...
}
// Finestra con titolo senza colore
Window (String t) {
...
title = t;}
// Finestra con titolo e colore
Window (String t, String c) {
...
title = t; color = c;}
}
```

Operatore this (Puntatore Auto-referenziante)

La parola riservata this e' utilizzata quale puntatore auto-referenziante

- this riferisce l'oggetto (e.g., classe) corrente

Utilizzato per

- Referenziare la classe appena istanziata
- Evitare il conflitto tra nomi

```
class Automobile{
String colore;
...
...
void vernicia (String colore) {
this.colore = colore;
}
}
. . .
Automobile a2, a1 = new Automobile;
a1.vernicia("bianco"); // a1 == this
a2.vernicia("rosso");
// this == a2
```

- Evitare il conflitto tra nomi

```
class Automobile{
String colore;
...
...
void vernicia (String colore) {
this.colore = colore;
}
}
. . .
Automobile a2, a1 = new Automobile;
a1.vernicia("bianco"); // a1 == this
a2.vernicia("rosso");
// this == a2
```

Notazioni Puntate

Le notazioni puntate possono essere combinate

- System.out.println("Hello world!");
- System è una classe del package java.lang
- out è una variabile di classe contenente il riferimento ad un oggetto della classe PrintStream che punta allo standard output
- println è un metodo della classe PrintStream che stampa una linea di testo

Operazioni su reference

Definiti gli operatori relazionali == e !=

- Attenzione: il test di uguaglianza viene fatto sul puntatore (reference) e NON sull'oggetto
- Stabiliscono se i reference si riferiscono allo stesso oggetto

È definita l'assegnazione

È definito l'operatore punto (notazione puntata)

NON è prevista l'aritmetica dei puntatori

Visibilità ed encapsulation

Motivazione

- Modularità = diminuire le interazioni
- Information Hiding = delegare responsabilità

Supporto sintattico

- private attributo/metodo visibile solo da istanze della stessa classe
- public attributo/metodo visibile ovunque

Getters e setters

Aggiungere metodi per leggere scrivere un attributo privato

Può infrangere (indirettamente) l'encapsulation

```
String getColor() {
return color;
}
void setColor(String newColor) {
color = newColor;
}

class Automobile {
public String colore;
}
Automobile a = new Automobile();
a.colore = "bianco"; // ok
class Automobile {
private String colore;
public void vernicia(String colore)
{this.colore = colore;}
// ok
}

Automobile a = new Automobile();
a.colore = "bianco"; // error
a.vernicia("verde"); // ok
```

Variabili e metodi di classe

Variabili di classe

- Rappresentano proprietà comuni a tutte le istanze
- Esistono anche in assenza di istanze (oggetti)
- Dichiarazione: static
- Accesso: nome-classe . attributo

```
class Automobile {
static int numeroRuote = 4;
}
Automobile.numeroRuote;
```

Metodi di classe

Funzioni non associate ad alcuna istanza

- Dichiarazione: static
- Accesso: nome-classe . metodo()

```
class HelloWorld {
public static void main (String args[]) {
System.out.println("Hello World!");

//p.es  cos(x): metodo static della classe Math, ritorna un double
double y = Math.cos(x);
}
}
```