

Programmazione in Java (8)

Mauro Lorenzutti

Scaletta

- Ripasso ereditarietà
- Esercitazioni
 - Solidi
 - Persone
 - I/O
- Serializzazione

Ripasso ereditarietà

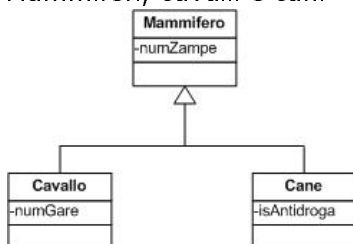
- Concetti base
- Modificatori di visibilità
- Overriding
- La classe Object
- Classi astratte
- Polimorfismo

Concetti base

- L'ereditarietà è l'atto di derivare una nuova classe da una già esistente (esempio progetto casa)
- Lo scopo è il riutilizzo del software
- Le variabili e i metodi ereditati possono essere usati nella classe derivata come se fossero locali
- Gerarchie di classi OK
- Ereditarietà multipla NO!!!!

Un esempio

- Mammiferi, cavalli e cani



Nomi

- La classe Mammifero è definita:
 - Classe base
 - Superclasse
 - Classe padre
- Le classi Cavallo e Cane sono dette:
 - Classe figlio
 - Sottoclasse

Modificatori di visibilità

public	a()
protected	b()
	c()
private	d()

public e private

- I metodi e le variabili *public* sono ereditati da tutte le sottoclassi e accessibili ovunque
- I metodi e le variabili *private* NON vengono mai ereditate e sono visibili solamente all'interno della classe in cui vengono dichiarati

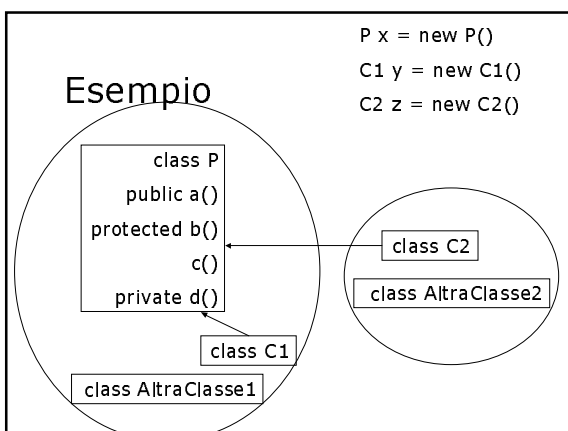
protected

- I metodi e le variabili *protected* sono ereditati da tutte le sottoclassi, indipendentemente dal package di appartenenza.
- Ogni classe nello stesso package può accedere ai membri *protected* di una classe.

Default

- Nessun modificatore specificato
- Sono ereditati solamente dalle sottoclassi che appartengono allo stesso package della superclasse.
- Sono visibili da tutte le classi all'interno dello stesso package.

Esempio



Esempio 2

- Il metodo *c()* ha visibilità di default, quindi:
 - C1 eredita *c()*
 - C2 NON eredita *c()*
 - AltraClasse1 può invocare il metodo *x.c()*
 - AltraClasse2 NO!

Overriding

- Si parla di *overriding* quando una classe figlio definisce un metodo con lo stesso nome e lo stesso prototipo della classe padre.
- Mascherare le variabili non è consigliabile

La classe Object

- È la classe base in Java
- Se una classe non estende direttamente un'altra classe (*extends*) in automatico estende la classe *Object*
- Due metodi particolari:
 - equals()
 - toString()

Classi astratte

- Una classe astratta rappresenta un concetto generico partendo dal quale le sottoclassi possono definire una propria implementazione
- Una classe astratta NON può essere istanziata
- Una classe derivata da una astratta deve ridefinire tutti i metodi astratti della classe padre, altrimenti sarà essa stessa una classe astratta.

Esempio

```
abstract public class Poligono {  
    int numLati;  
    String tipo = "Poligono";  
    public Poligono(int nl){ numLati = nl; }  
    public int numeroDiLati(){return numLati;}  
    public String tipo(){ return tipo;}  
  
    abstract public double area();  
    abstract double perimetro();  
}
```

Polimorfismo

- Polimorfismo: avere più forme
- Un riferimento polimorfico è un riferimento che può puntare a oggetti di tipi differenti in momenti differenti
- Interfacce
- Ereditarietà

Un esempio

```
Poligono elenco[] = new Poligono[4];  
elenco[0] = new Rettangolo(3.2, 4.5);  
elenco[1] = new Quadrato(5);  
elenco[2] = new Triangolo(3,4,5);  
elenco[3] = new TriangoloEquilatero(3);  
for (int i = 0; i<elenco.length; i++)  
    System.out.println("Il perimetro del " +  
        elenco[i].tipo() + " è " +  
        elenco[i].perimetro());
```

Dynamic binding

- Nell'esempio precedente i metodi tipo() e perimetro() vengono collegati solamente durante l'esecuzione
- Si parla di
 - Late binding (collegamento ritardato)
 - Dynamic binding (collegamento dinamico)

Esercizi

- Solidi
 - Estendere il package "geometria" della lezione scorsa
- Persone
 - Esempio in sospeso della lezione scorsa
- I/O
 - Vedi slide seguente

I/O

- Progettare e implementare una gerarchia di classi per astrarre il device di I/O per la classe Persona
- Input da:
 - Tastiera
 - File
- Output su
 - Video
 - File

Serializzazione

- Perché
- Concetti base
- Un po' di codice
- Esercizio

Persistenza degli oggetti

- Quando un programma termina tutti gli oggetti muoiono con lui
- Serve un meccanismo per salvare gli oggetti per farli "vivere" indipendentemente dal programma che li ha generati
- Questo fenomeno è detto **persistenza** degli oggetti

Serializzazione

- Per ottenere la persistenza degli oggetti si può codificarli in un file e poi leggerli → molto scomodo
- Si può serializzarli e scriverli in un file o inviarli attraverso la rete
- Serializzare un oggetto vuol dire rappresentarlo come una sequenza di byte
- Questa rappresentazione in byte può essere usata per ricostruire l'oggetto

Serializzare su file

```
FileOutputStream fos = new  
    FileOutputStream(nomefile);  
  
ObjectOutputStream oos = new  
    ObjectOutputStream(fos);  
  
oos.writeObject(oggetto);
```

De-serializzare da file

- Il processo inverso: da un flusso di byte ricostruire l'oggetto originale

```
FileInputStream fis = new  
    FileInputStream(nomefile);  
  
ObjectInputStream ois = new  
    ObjectInputStream(fis);  
  
TipoOggetto oggetto =  
    (TipoOggetto)ois.readObject();
```

Attenzione!

- L'oggetto da serializzare e tutti i suoi attributi devono essere predisposti alla serializzazione

```
public class Persona implements  
    Serializable{  
  
    private static final long serialVersionUID =  
        1L;
```

Escludere dal flusso

- È possibile in certi casi avere la necessità di escludere una variabile dal flusso di byte

```
private transient String password;
```

Esercizio

- Estendere la gerarchia di I/O per includere una classe che permetta il salvataggio su file di dati serializzati e la successiva lettura e ricostruzione di questi