Corso sul linguaggio Java

Modulo JAVA4

B1– Proprietà della OOP

Prerequisiti

- Programmazione elementare ad oggetti
- Proprietà della OOP
- Concetto di conversione di tipo (**casting**)
- Gerarchia di classi

Introduzione

Lo scopo di questa Unità è quello di mostrare in pratica l'applicazione delle proprietà di ereditarietà e di polimorfismo.

Si vedrà come sfruttare la possibilità di riusare software già collaudato e funzionante, allo scopo di diminuire i tempi di sviluppo.

Nel riuso del software si possono espandere le funzionalità di quello esistente, o mascherarne gli effetti.

Ereditarietà

```
public class Quadrilatero
{  protected int a,b,c,d;
public Quadrilatero (int l1, int l2, int l3, int l4)
{   a=l1;  b=l2;
    c=l3;  d=l4;   }
public Quadrilatero()
{a=0; b=0; c=0;  d=0;  }
public int Perimetro()
{ return a+b+c+d; }
public void getQ()
{   System.out.println("\na= " + a + "\nb= " + b +"\nc= " + c + "\nd= " +
    d);   }
}
```

Attributi **protected**

La classe pubblica *Quadrilatero* (salvata nel file *Quadrilatero.java*) è creata come classe base. Possiede un costruttore con parametri e uno senza.

Il metodo *Perimetro*() fornisce la somma delle misure dei lati.

Il metodo *getQ()*() stampa le misure dei lati.

Ereditarietà

```
class Rettangolo extends Quadrilatero
{ public Rettangolo (int l1, int l2)
  {   a=l1; b=l2; c=l1; d=l2;    }
  public int Area ()   {return a*b;}
}
class Quadrato extends Quadrilatero
{  public Quadrato (int l)
{   a=l; b=l; c=l; d=l;   }
public int Area ()
{   return (int) Math.pow(a,2);  }
}
```

La classe *Rettangolo* (*Rettangolo.java*) eredita da *Quadrilatero* i 4 dati membro. Il costruttore riceve i due parametri (che rappresentano le due dimensioni del rettangolo)

Il metodo *Perimetro*() è ereditato senza ulteriori modifiche

Il metodo *Area*() fornisce la misura dell'area del rettangolo

Analogamente si procede per la classe *Quadrato* (salvata in *Quadrato.java*)

ATTIVITA': disegnare la gerarchia delle classi

Ereditarietà

```
import java.io.*;
public class TestQuadrilatero
{ public static void main (String args[ ])
{
 InputStreamReader  in = new InputStreamReader(System.in);
 BufferedReader tastiera = new BufferedReader(in);
 try  {  creazione oggetti Q, R e Qd;
         stampa lati e perimetro di Q;
         stampa lati, perimetro e area di R;
         stampa lati, perimetro e area di Qd;   }
 catch (Exception e)
 {  System.out.print("\n Errore!");   }
} /*main*/
}  /*Classe*/
```

Programma di testing delle classi (file *TestQuadrilatero.java*)

ATTIVITA': creare le classi

- Quadrilatero
- Rettangolo
- Quadrato
- TestQuadrilatero

e testarne il funzionamento

Upcasting e downcasting

Spesso capita che un oggetto di una classe debba essere convertito in un oggetto di una superclasse o viceversa. Considerate le due classi Quadrilatero e Rettangolo potremmo voler creare oggetti del tipo:

Quadrilatero Q1 = **new** Rettangolo(1,2);

upcasting

downcasting

Rettangolo R1 = **new** Quadrilatero (3,4,5,6)

Upcasting e downcasting

L'**upcasting** (*promozione alla superclasse*) può essere realizzato indifferentemente con una delle seguenti

Quadrilatero Q1 = **new** Rettangolo(5,4);
Quadrilatero Q1 = (Quadrilatero) **new** Rettangolo(5,4);

e non crea problemi in quanto nella conversione, al limite, si perde l'uso di alcuni attributi.
Il **downcasting** invece richiede l'operatore *cast* come indicato

Quadrilatero Q2=new Rettangolo(2,3);
Rettangolo R2=(Rettangolo) Q2;

In qualunque altro modo, si hanno errori in compilazione o in esecuzione.

Polimorfismo

Come sappiamo, un stesso metodo può assumere varie forme (stesso nome, ma diversa firma). Questa proprietà della **OOP** prende il nome di **polimorfismo**.

Il **polimorfismo** si presenta quando:

- in una **gerarchia di classi** un metodo viene ridefinito all'interno di una classe, coprendo quello originario (**overriding** dei metodi)
- all'interno di **una medesima classe**, un metodo è presente più volte, con lo stesso nome, ma con interfaccia diversa, come numero e/o tipo dei parametri (**overloading** dei metodi)

Overloading

La classe *Quadrilatero* presenta due costruttori con interfaccia diversa. In questo caso il **polimorfismo** si presenta come **overloading**.

```
public class Quadrilatero
{  int a,b,c,d;
public Quadrilatero (int l1, int l2, int l3, int l4)
{   a=l1;   b=l2;   c=l3;   d=l4;   }
public Quadrilatero()
{   a=0;  b=0;   c=0;   d=0;   }
public int Perimetro()
{   return a+b+c+d;   }
public void GetQ()
{   System.out.println("\na= " + a + "\nb= " + b +"\nc= "
    + c + "\nd= " + d);
}
}
```

Overriding

```
class Rettangolo extends Quadrilatero
{   public Rettangolo (int l1, int l2)
    { a=l1; b=l2; c=l1; d=l2;  }
    public int Area ()
    { return a*b;}
}
class Quadrato extends Quadrilatero
{ public Quadrato (int l)
   { a=l; b=l; c=l; d=l;  }
   public int Area ()
   {  return (int) Math.pow(a,2);}
}
```

Le due classi *Rettangolo* e *Quadrato*, entrambe derivate da *Quadrilatero*, presentano ciascuna una propria versione del metodo *Area()*.

In questo caso il **polimorfismo** si manifesta come **overriding**.

Istanza di metodi in una gerarchia

Il polimorfismo, sia come *overloading* che come *overriding*, pone alcune questioni:

Se abbiamo due costruttori in overloading, come riconoscere quando viene istanziato l'uno o l'altro?

La scelta avverrà al momento della creazione dell'oggetto, in base all'interfaccia

Se abbiamo due metodi in overriding, come riconoscere quando viene istanziato uno o l'altro?

In base all'oggetto che lo istanzia.

Chiamata del metodo originale

```
class Quadrilatero
{  int a,b,c,d;
public Quadrilatero (int l1, int l2, int l3, int l4)
{   a=l1;   b=l2;  c=l3;  d=l4;
}
public int Perimetro()
{   return a+b+c+d;
}
public void getQ()
{   System.out.println("\na= " + a + "\nb= " + b +"\nc= " + c + "\nd= " + d);
}
}
```

Data la classe *Quadrilatero* vogliamo derivare la classe *Rettangolo*

Chiamata del metodo originale

```
class Rettangolo extends Quadrilatero
{ public Rettangolo (int a, int b)
  {   super (a, b, a, b);
  }
  public int Perimetro (int a, int b)
  {   return super.Perimetro();
  }
}
```

La classe *Rettangolo* chiama il **metodo originale** costruttore di *Quadrilatero* mediante **super**().

Il metodo *Perimetro*() invece di essere ridefinito, chiama anche questo il **metodo originale** mediante **super**. *Perimetro*(),

Con la parola chiave **super** si indica un oggetto della superclasse

Chiamata del metodo originale

```
import java.io.*;
public class TestQuadrilatero
{ public static void main (String args[])
  { // creazione oggetto Tastiera
    try{   Quadrilatero Q = new Quadrilatero(1,2,3,4);
           Rettangolo R = new Rettangolo(1,2);
           Q.getQ();
           System.out.println("Perimetro quadrilatero: " + Q.Perimetro());
           R.getQ();
           System.out.println("Perimetro rettangolo..: " + R.Perimetro());
      }
    catch (Exception e)
    {  System.out.print("\n Si è verificato un errore");     }
  } /*Main*/
}  /*Classe*/
```

Il programma *TestQuadrilatero.java* mostra l'utilità della chiamata del metodo originale

Classi astratte

Per trarre dall'ereditarietà il massimo vantaggio, si potrebbe pensare di progettare una superclasse *che menzioni proprietà e comportamenti generali senza fornire nessuna implementazione specifica.*

Queste classi si dicono **classi astratte**.

Le classi derivate da una classe astratta, che forniscono l'implementazione di tutti i metodi mancanti, si dicono **classi concrete.**

Ad esempio, pensiamo alla classe *Alimento*: non esiste un alimento in generale, esistono solo alimenti specifici (acqua, cioccolata, carne); ma tutti gli alimenti possiedono *caratteristiche comuni* (provenienza, potere calorico, tenore in grassi, carboidrati, proteine, ecc).

Quindi la classe *Alimento* può essere considerata astratta perché raccoglie in sé proprietà simili a vari tipi di alimento;

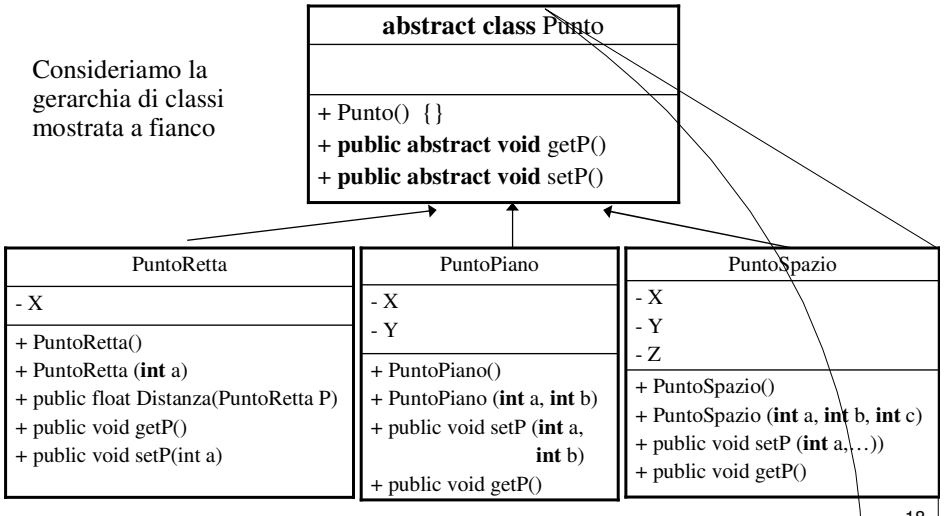
Classi astratte

E' ovvio che una classe astratta:

- non può essere istanziata: non ha senso parlare di un oggetto di classe *Alimento*, poiché va specificato di che alimento si tratti;
- non può essere la classe terminale di una gerarchia: la classe astratta, per definizione, serve a derivare classi concrete e non può quindi terminare una gerarchia di classi;
- deve possedere almeno un metodo astratto, ossia un metodo di cui non si fornisce l'implementazione.

La classe astratta *Punto*

Consideriamo la gerarchia di classi mostrata a fianco



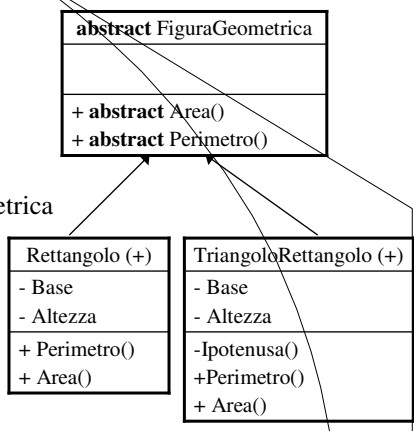
La classe astratta *Punto*

Osservazioni:

- come si nota, la classe *Punto* è astratta poiché contiene i metodi astratti *setP*() e *getP*();
- i metodi astratti vengono espansi in modo specifico nelle singole sottoclassi;
- ciascuna sottoclasse ridefinisce il proprio costruttore in forma senza parametri e con parametri;
- la classe *PuntoRetta* definisce il metodo *Distanza (PuntoRetta P)* per il calcolo della distanza tra due punti sulla retta.

Classe astratta *FiguraGeometrica*

```
public abstract FiguraGeometrica
{ ….
  abstract float Area ();
  abstract float Perimetro();
}
public class Rettangolo extends FiguraGeometrica
{  float Base, Altezza;
   public float Perimetro()
   {return (Base+Altezza)*2;  }
}
```



Documentazione di sistema

L'ambiente Java, estremamente ricco e complesso, necessita di un adeguato sistema di documentazione, che descriva nei dettagli ciascuna applicazione.

Per quanto concerne la **documentazione del software di sistema** si ha un eccezionale, voluminoso e dettagliato insieme di pagine web (ad esempio per la versione 5.0) sul sito

http://java.sun.com/j2se/1.5.0/docs/

Documentazione applicativi

Per quanto riguarda invece la **documentazione del software applicativo** (come quello che sviluppiamo noi) esiste una funzionalità **Javadoc** © che genera la documentazione in modo automatico.

Javadoc è un'applicazione Java (reperibile nelle cartelle di sistema della SDK) che elabora:

- le dichiarazioni (attributi e metodi)
- i commenti

producendo un file HTML, con le descrizioni della classe, dei costruttori e della visibilità (**public**, **private**, **protected**).

È bene evitare di inserire manualmente marcatori HTML in quanto java già compone correttamente le pagine documentali

Javadoc

I **commenti doc** sono i commenti utilizzabili nel codice Java che servono a produrre documentazione automatica.

Commenti doc	
/** descrizione commento */	Descrizione monolinea
/** * descrizione commento */	Descrizione multilinea
/** * * @tag commento del tag */	Descrizione di un tag (che sono elencati di seguito)

I tag di *Javadoc*

I tag di **javadoc** più importanti sono i seguenti:

Tag	**Effetto**
@author *nome* (*)	Nome dell'autore
@param *nome descrizione*	Descrizione dei parametri
@return *descrizione*	Descrizione tipo restituito
@throws *nomeclasse descrizione*	Descrizione classe di eccezione
@see *riferimento*	Aggiunta See Also: ……
@since *versione*	Versione in cui viene introdotta

(*) non visualizzato nella pagina web

Javadoc da finestra DOS

Per eseguire **Javadoc** da finestra DOS:

- aprire la finestra DOS
- posizionarsi nella cartella in cui c'è il file sorgente **.java**
- lanciare il comando

  C:…..>javadoc *nomefile*.java <INVIO>
- vengono creati alcuni file tra cui *nomefile*.html
- aprire *nomefile*.html con il browser preferito

Javadoc da ambiente integrato

Per eseguire **Javadoc** da ambiente integrato (TexPad, Eclipse):

- selezionare nell'ambiente di sviluppo l'opzione per eseguire un comando
- specificare che si vuole eseguire il comando **javadoc** (presente nelle cartelle di sistema di Java SDK) eventualmente reperendolo con *Sfoglia*
- inserire come parametro di **javadoc** *nomefile*.java
- eseguire il comando
- vengono creati alcuni file tra cui *nomefile*.html
- aprire *nomefile*.html con il browser preferito

Javadoc in JCreator

Per installare javadoc in JCreator.

- selezionare **Configure…Options…** dalla barra dei menu. Apparirà una finestra di dialogo **Options**
- selezionare **Tools** dal pannello di sinistra
- cliccare il bottone **New**
- selezionare **Program** dalla lista. Apparirà la finestra di dialogo **Open**
- esplorare i file e le cartelle per trovare il file **Javadoc.exe**.
- fare doppio click sul file o cliccare sul bottone **Open**

Javadoc in JCreator

- cliccare il bottone **Apply** situato in basso alla finestra di dialogo options. Questo aggiunge l'utility al menù **Tools** nella parte sinistra del pannello
- selezionare **javadoc** dal menu Tools nella parte sinistra del pannello e inserire le seguenti informazioni:

  Arguments: -d "$[FileDir]\doc" $[FileName] -author -version -private

  Initial directory: $[FileDir]
- fare click su **OK**.

Nel menu Tools ora appare **javadoc**.

Javadoc in JCreator

Per eseguire javadoc in JCreator:

- aprire/creare un nuovo progetto o un file java
- compilare il codice
- selezionare **Tools..javadoc**
- una finestra di DOS apparirà appena la documentazione HTML sarà creata. Se c'è un errore, lo si vedrà nella finestra DOS che rimarrà aperta. Se non ci sono errori, la finestra si chiuderà automaticamente.

La documentazione HTML sarà creata nella sottocartella **doc** della cartella che contiene la classe.

Nella cartella **doc** si trova il file **index.html** che, aperto, consente di vedere la documentazione generata.

Argomenti

- Ereditarietà
- Upcasting e downcasting
- Polimorfismo
- Overloading
- Overriding
- Istanza di metodi in una gerarchia
- Chiamata del metodo originale
- Classi astratte
- La classe astratta *Punto*
- La classe astratta *FiguraGeometrica*
- Documentazione di sistema
- Documentazione applicativi
- *Javadoc*
- I tag di *Javadoc*
- *Javadoc* da finestra DOS
- *Javadoc* da ambiente integrato
- *Javadoc* in JCreator

Altre fonti di informazione

- P.Gallo, F.Salerno – Informatica Generale 1, ed. Minerva Italica
- M.Romagnoli, P.Ventura – Linguaggio C/C++, ed. Petrini
- M. Bigatti – Il linguaggio Java, ed. Hoepli