Corso sul linguaggio Java

Modulo JAVA4

A2–Utilizzo di classi

Prerequisiti

- Implementazione di classi
- Funzionamento di una classe
- Parametri formali e attuali di una funzione

Introduzione

In questa Unità vediamo alcuni concetti di **OOP** in Java, come la possibilità di utilizzare **array di oggetti**, e di avere metodi **statici**, ossia metodi che per essere istanziati, non richiedono un oggetto.

Vediamo alcuni concetti, già noti dalla programmazione procedurale, come la possibilità per una funzione (metodo) di avere oggetti come parametri e come valori di ritorno.

Array di oggetti

Un **vettore di oggetti** segue le stesse regole dei vettori di variabili semplici.

ATTIVITA': creare una classe *Libro* con attributi *codLibro, nCopie, autore, editore, titolo, prezzo*. Fornire un costruttore senza parametri e i metodi modificatori *setXXX*() e accessori *getXXX*() per gli attributi.

Analisi del problema

Per la classe richiesta, poniamo come attributi interi *codLibro, nCopie*, come reale il *prezzo*, come stringa *autore, editore* e *titolo.*

Forniamo un costruttore senza parametri *Libro()* e almeno i seguenti metodi:

- *setXXX*(…) metodo modificatore per impostare l'attributo XXX;
- *getXXX*() metodo accessore per estrarre il valore dell'attributo XXX.

Creiamo come file separato la classe *testLibro.java*.

Array di oggetti

ATTIVITA': rappresentare la classe *Libro* in UML.

Libro	
- **int** codLibro;	codice del libro
- **int** nCopie;	numero copie presenti
- **String** autore;	autore del libro
- **String** titolo;	titolo
- **String** editore;	nome editore
- **double** prezzo;	prezzo di vendita
+ Libro ();	costruttore senza parametri
+ Libro(**int** c, **int** nc, **double** pr, **String** aut, **String** edit, **String** tit);	costruttore con parametri
+ **void** getXXX();	estrae valori dell'attributo XXX
+ **void** setXXX();	imposta l'attributo XXX

ATTIVITA': implementare la classe *Libro*

Array di oggetti

```
import java.io.*;
public class Libro
{   int codLibro,  nCopie;
    String autore, titolo, editore;
    float prezzo;
    public Libro ()
    {       inizializza attributi;    }
    public void setCodice (int codice)
    {       codLibro = codice}
    …
    public int getCodice()
    {       return codLibro; }
    ……
}
```

Per semplicità abbiamo indicato i soli metodi *setCodice*() e *getCodice*()

ATTIVITA': implementare la classe *testLibro*. tenendo presente che deve consentire di rappresentare una serie di libri.

Array di oggetti

```
import java.io.*;
class TestLibro
{   public static void main (String args[])
    {       final int MAX = 3; int cod;
            Libro elenco[] = new Libro[MAX];
            for (int i=0; i<MAX; i++)
            {           Libro l = new Libro();
                        lettura cod;
                        l.setCodice(cod);
                        …
                        elenco[i]=l;
            }
            for (int i=0; i<MAX; i++)
            {           System.out.println(elenco[i].getCodice());
                        …
            }
    } // end main
}
```

Creazione array di oggetti di classe *Libro*

Creazione oggetto di classe *Libro* da inserire

Immette valori attributo

Inserisce oggetto nel vettore *elenco[ ]*

Stampa il vettore

Membri static

Può capitare il caso in cui:

- un attributo debba sempre avere lo stesso valore per tutte le istanze (oggetti) di una classe (ad es. il numero di ruote di una vettura);
- un metodo (o un attributo) sia legato alla classe anziché agli oggetti che essa origina (come le funzioni introdotte nel Modulo 2, o i *metodi della classe Math*).

In questo caso si parla di **membri static**.

Se un membro (**attributo** o **metodo**) è definito **static**, esso ha visibilità globale, ossia è **condiviso** da tutti gli oggetti della classe.

Attributi static

Gli **attributi static**

- hanno un valore visibile da tutti gli oggetti della classe;
- si dicono anche **variabili di classe**;
- si dichiarano come le variabili istanza con la sintassi

  static *tipo ident_ variabile;*
- possono essere inizializzati:
  - nel costruttore
  - nel **main** ()
  - al momento della dichiarazione
- nel caso di classi esterne, si accedono con la sintassi

  nomeclasse.nomeattributo

Attributi static

```
public class NumeroIntero
{  private static int x=3;
   public int getX()
   {  return x;     }
   public void setX (int newX)
   {  x = newX; }
    public static void main(String args[])
    {   NumeroIntero num1 = new NumeroIntero();
        NumeroIntero num2 = new NumeroIntero();
        num1.setX(1);   num2.setX(2);
        System.out.println("num1.x = " + num1.getX());
        System.out.println("num2.x = " + num2.getX());
    }
}  // end class
```

Variabile **static** dichiarata e inizializzata

Si imposta un valore diverso di *x* nei due oggetti

In stampa si ottiene il valore 2, *ultimo valore assegnato*.

Metodi static

I **metodi static**

- si dicono anche **metodi di classe**;
- non richiedono un oggetto per essere istanziati;
- hanno interfaccia con un numero di parametri pari al numero di oggetti su cui il metodo agisce;
- nel caso di classi esterne, si istanziano con la sintassi

  nomeclasse.nomemetodo (*listaparametri*);

Metodi static

```
public class MetodiStatici
{  private int dato;
   public MetodiStatici()
   {   dato=0;     }
   void setMember (int d)
   {  dato=d;     }
   static void getMember (MetodiStatici M)
   {  System.out.println("Dato: " + M.dato); }
}
```

L'oggetto va *passato come parametro*

```
public class TestMetodiStatici
{ public static void main(String args[ ])
  { try
    {  MetodiStatici as1 = new MetodiStatici();
       MetodiStatici as2 = new MetodiStatici();
       as1.setMember(5);
       as2.setMember(8);
       MetodiStatici.getMember(as1);
       MetodiStatici.getMember(as2);
    }      /* fine try */
    catch (Exception E)
    { System.out.println("Errore");   }
  }/* fine main */
} /* fine classe */
```

Istanza mediante il nome della classe

Il riferimento **Oggetto corrente**

Quando si istanzia più volte una classe, si ottengono oggetti indipendenti, ciascuno con i propri membri.

Se si creano molti oggetti da una classe, si corre il rischio di avere più copie degli stessi metodi in memoria?

NO, in realtà il compilatore fa sì che i metodi vengano condivisi. Quando si istanzia un metodo su un oggetto viene creato un riferimento chiamato **riferimento oggetto corrente** e si indica con la parola chiave **this**.

Il riferimento **Oggetto corrente**

Questo riferimento è utile anche quando si vuole usare in un metodo (come parametro o come variabile locale) una variabile con lo stesso nome dell'attributo che si utilizza all'interno del metodo stesso.

Ad esempio, un metodo che debba incrementare un attributo *contatore* (variabile istanza) di un valore *contatore* (parametro), potrebbe essere il seguente:

Parametro *contatore*

```
void Totale (int contatore)
{  contatore = contatore + contatore;    }
```

Qual è l'attributo? Quale il parametro ?

Il riferimento **Oggetto corrente**

La confusione tra il nome del parametro e quello della variabile istanza può essere evitata se si usa il **riferimento oggetto corrente** come segue:

Parametro *contatore*

```
void Totale (int contatore)
    {
        this.contatore=this.contatore+ contatore;
    }
```

Riferimento alla variabile istanza

Riferimento al parametro formale

Oggetti come parametri

Come per i tipi predefiniti (**int**, **float**, ecc) anche un oggetto, può essere passato come parametro di un metodo. Nell'esempio precedente

```
static void getMember (MetodiStatici M)
    {  System.out.println("Dato: " + M.dato); }
```

il metodo *getMember()* riceve come parametro un oggetto.

Consideriamo la seguente classe *Punto* e si osservi il metodo *Distanza*().

ATTIVITA': implementare la classe *Punto* con attributi *x* ed *y*, un costruttore con parametri, uno senza ed un metodo *Distanza*(Punto *p*) per calcolare la distanza tra due punti.

Oggetti come parametri

```
public class Punto
{ private
    float X, Y;
  public Punto()
  { X=0;  Y=0;
  }
  public Punto (int a, int b)
  {  X=a;  Y=b;
  }
  public double Distanza (Punto P)
  { return (Math.sqrt( Math.pow((X - P.X), 2)+
                       Math.pow((Y - P.Y),2)));
  }
}
```

Si noti che il metodo *Distanza(Punto P)* riceve <u>come parametro un oggetto</u> di classe *Punto* e calcola la distanza tra due punti.(uno è il parametro, *l'altro è l'oggetto che istanzia il metodo)*

Oggetto di classe *Punto* passato come **parametro formale**

ATTIVITA': aggiungere alla classe *Punto* il metodo **void** *getP*() che stampi ordinatamente le coordinate di P nella forma (X, Y).

Oggetti come parametri

ATTIVITA': creare la classe *TestPunto.java*

```
import java.io.*;
public class TestPunto
{ public static void main (String args[]){
  double d=0;
   // creazione oggetto Tastiera
  try{ Punto P0 = new Punto(3, 4); /* Crea primo punto */
       Punto P1= new Punto(0, 2);  /* Crea secondo punto */
       d= P0.Distanza(P1);         /* Calcola la distanza */
       System.out.print("\ndistanza: " +  d + "\n");
      }
     …..
} /* end class*/
```

L'istanza del metodo *Distanza(….)* riceve P1 come **parametro attuale**

ATTIVITA': aggiungere alla classe *Punto* il metodo *pMedio*(Punto *p*) per calcolare il punto medio tra due punti.

Oggetti come valori di ritorno

```
public Punto pMedio (Punto P)
{   Punto Ptemp =  new Punto();
    Ptemp.X=(P.X+X)/2;
    Ptemp.Y=(P.Y+Y)/2;
    return Ptemp;
}
```

Come si vede *pMedio(Punto P)* riceve un **oggetto come parametro** e produce **un oggetto come valore di ritorno** che è il punto medio tra P e il punto (oggetto) che lo istanzierà nel programma principale)

ATTIVITA': come si istanzia il metodo in questo caso?

Nel programma di test basterà aggiungere le istruzioni:

```
System.out.print("Il punto medio e' ");
Punto T = P0.pMedio(P1);
T.getP();            // stampa le coordinate di P
```


Oggetti come valori di ritorno

Quindi:

- quando si implementa un metodo, il valore di ritorno può essere **void** o un **tipo qualunque** (anche un oggetto della classe stessa che potrebbe essere creato dal metodo);
- normalmente un metodo opera sull'**oggetto corrente**, individuato dal **puntatore "this**" Per questo motivo i metodi presentano un argomento in meno nella loro interfaccia.

Utilizzo di metodi **static**

ATTIVITA': aggiungere alla classe *Punto* il metodo

static *pMedio*(Punto *p1*, Punto *p2*) per calcolare il punto medio tra due punti.

```
public static Punto pMedio (Punto P1, Punto P2)
{   Punto Ptemp =  new Punto();
    Ptemp.X=(P1.X+P2.X)/2;
    Ptemp.Y=(P1.Y+P2.Y)/2;
    return Ptemp;
}
```

Il metodo

Medio(pMedio(Punto *p1*, Punto *p2)*

può coesistere con quello non statico grazie al **polimorfismo**.

Nel programma di test basterà aggiungere le istruzioni:

```
System.out.print("Il punto medio e' ");
Punto P2 = Punto. pMedio(P0, P1);
P2.getP();
```

Il metodo **static** non richiede un oggetto per essere istanziato, ma il nome della classe. *Si noti la presenza di 2 parametri.*

Argomenti

- Array di oggetti
- Membri **static**
- Attributi **static**
- Metodi **static**
- Il riferimento **Oggetto corrente**
- Oggetti come parametri
- Oggetti come valori di ritorno
- Utilizzo di metodi **static**

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