Corso sul linguaggio Java

Modulo JAVA4

A1 – Classi e oggetti

Prerequisiti

- Oggetti reali e oggetti software
- Proprietà e comportamento di un oggetto
- Programmazione elementare in Java

Introduzione

In questa Unità vediamo i primi concetti della **OOP** in Java.

Si esaminano le tecniche elementari per la realizzazione di una classe e alcuni concetti di base.

Astrazione

Quando si deve **progettare** qualcosa occorre:

- schematizzare la realtà considerando soltanto gli aspetti che ci interessano
- creare un modello semplificato che però svolga le funzioni che ci interessano
- considerare gli aspetti dell'entità che ci interessano ad un livello di dettaglio sufficiente per potere operare con l'entità stessa

Questa fase progettuale si dice **astrazione**.

Quando si progetta una classe, si fa un lavoro di astrazione: la classe consente di creare **oggetti software** che simulano **oggetti reali**

Implementazione

Una volta che si è creato un modello astratto, occorre **realizzare** il software che consente di:

- **rappresentarlo** (**attributi**)
- **operare** con esso (**metodi**)

Questa fase si dice **implementazione**.

Una volta implementata una classe, possiamo creare da essa quanti oggetti vogliamo ed operare con essi solo conoscendone l'interfaccia.

Programmazione ad oggetti

La **programmazione strutturata** (tipica dei linguaggi Pascal, C, Fortran) si è evoluta dagli anni '60 fino agli anni '80, consentendo di raggiungere buoni standard di produzione software.

Dagli anni '80 in poi, la tecnica della programmazione ha subito un'ulteriore evoluzione mediante la **tecnica di programmazione ad oggetti** che consente un notevole passo in avanti per la realizzazione di software:

- **riusabile** (si evita di doverlo riscrivere)
- **modificabile** (si adatta con facilità a nuove esigenze)
- **sicuro** (che consenta *protezione dei dati*)

Il linguaggio Java
Applicazioni

In Java, ogni programma è una **classe**.

Programmare ad oggetti in Java significa scrivere un'**applicazione** *che definisca un insieme di classi*:

- **una classe che modella il programma**
- **una classe per ciascuna tipo di oggetto** necessario per l'esecuzione del programma

Il linguaggio Java
Programmazione in Java

La programmazione **OOP** in Java coinvolge i seguenti aspetti:

- *conoscenza del linguaggio Java* (sintassi e semantica di Java)
- *uso di oggetti e classi predefiniti* (ad esempio, definiti nelle **API** di Java o in altri package a disposizione)
- *definizione di nuove classi*

Rivediamo brevemente i concetti già noti di Java, alla luce del paradigma ad oggetti.

Struttura di una classe

Nome della classe

```
public class Scrittore
/* Un esempio di classe */
{ public static void main (String[] args)
  {  System.out.println("Questo è il corso");
     System.out.println("di Java");
  }
}
```

In questo tipo di classe abbiamo il solo metodo: **main()**

- *Scrittore* è una classe
- Quando si esegue questa applicazione, viene *costruito automaticamente* un oggetto di classe *Scrittore*

Struttura di una classe

Definizione della classe

```
public class Scrittore
/* Un esempio di classe */
{ public static void main (String[] args)
  {  System.out.println("Questo è il corso");
     System.out.println("di Java");
  }
}
```

Il **corpo** della classe è delimitato dalle parentesi graffe principali

Struttura di una classe

Definizione metodo

```
public class Scrittore
/* Un esempio
   di classe */
{ public static void main (String[] args)
  {  System.out.println("Questo è il corso");
     System.out.println("di Java");
  }
}.
```

Nella definizione di un metodo poniamo le istruzioni che esso dovrà eseguire.

Il metodo **main**() *è l'unica operazione che può essere utilizzata direttamente dall'utente di una applicazione* in quanto viene eseguito automaticamente all'avvio dell'applicazione.

Struttura di una classe

Oggetto schermo

```
public class Scrittore
/* Un esempio
   di classe */
{ public static void main (String[] args)
  {  System.out.println("Questo è il corso");
     System.out.println("di Java");
  }
}
```

Messaggio da inviare a **System.out**

- Ognuna delle due istruzioni invia un messaggio a un oggetto. L'oggetto **System.out** visualizza le frasi indicate

Funzionamento di una classe

Il funzionamento della classe *Scrittore* è il seguente (dopo aver ovviamente compilato il programma):

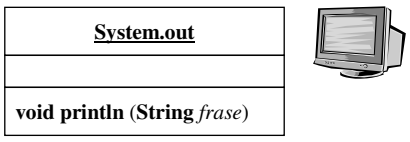
- si lancia l'applicazione
- viene creato automaticamente un oggetto di classe *Scrittore*
- l'oggetto *Scrittore* riceve il messaggio **main**()
- si eseguono in sequenza le istruzioni contenute nel metodo **main()** .
- **main**() lancia il messaggio **println()** all'oggetto **System.out** che visualizzerà sullo schermo la frase

  Questo è il corso
- successivamente, **main**() invia il secondo messaggio a **System.out** che visualizza la frase

  di Java

Funzionamento di una classe

L'applicazione può visualizzare le frasi sullo schermo perché utilizza l'oggetto **System.out**



System.out

- è un oggetto definito dalle **API** di Java
- modella lo *schermo del calcolatore*
- sa eseguire l'operazione di stampa **println()** che visualizza una frase (che è il parametro dell'operazione)

Dichiarazione di una classe

La sintassi per la **dichiarazione** di una classe Java è la seguente:

public class *ident*
{
dichiarazione attributi;
definizione dei metodi;
}

Nome della classe

Davanti a ciascun attributo si può specificare **public** o **private**.

Implementazione dei metodi

Davanti a ciascun metodo si può specificare **public** o **private**.

Dichiarazione di una classe

La **dichiarazione** di una classe ha le seguenti caratteristiche.

- Trattandosi di una dichiarazione si osserva che la *classe non occupa memoria*
- Quando la classe viene istanziata, si crea **l'oggetto** (considerato come una variabile) che *occupa effettivamente memoria*.
- Si può scrivere
  - *nello stesso file* che contiene il metodo **main**(), nel quale si istanzia la classe stessa ed i suoi metodi.
  - *in un file esterno* rispetto al **main()** (**classi esterne**)

Dichiarazione di una classe

Le clausole **public** o **private** prendono il nome di **specificatori di accesso** ed indicano se l'attributo o il metodo a cui si riferiscono può essere accessibile direttamente da*ll'utilizzatore della classe*.

In generale, per scopi di protezione, gli attributi vengono dichiarati **private** in modo che siano accessibili *solo tramite appositi metodi di lettura e scrittura* predisposti dal progettista e non direttamente dall'utente.

Implementazione di classi

Come accennato in precedenza, la classe si può trovare:

- all'interno dello stesso file che contiene il metodo **main**(). Nel **main**() vengono istanziati i metodi della classe
- in un file esterno rispetto al **main**().

Vediamo un esempio in cui si usa la prima tecnica.

La classe *Coppia*

ATTIVITA': si vuole creare una classe *Coppia* con 2 attributi privati *x* ed *y*, un metodo costruttore e due metodi per comunicare all'esterno i valori degli attributi. Scrivere l'analisi del problema.

Analisi del problema

Per la classe richiesta sono necessari due attributi, *x* ed *y* che, per semplicità dichiariamo interi. La classe *Coppia* è l'unica classe necessaria e i suoi oggetti dovranno avere le seguenti funzionalità:

- un costruttore senza parametri *Coppia()*;
- un costruttore con parametri, per creare oggetti con uno stato definito dall'utente *Coppia (***int** a, **int** b)
- un metodo (*accessore*) *getx(),* per comunicare il valore *x*
- un metodo (*accessore*) *gety*(), per comunicare il valore *y*.

La classe *Coppia*

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

Coppia	
- int x **- int** y	Primo attributo Secondo attributo
+ Coppia() + Coppia (**int** x, int y) + **void** getx() + **void** gety()	Costruttore senza parametri Costruttore con parametri Comunica il valore di x Comunica il valore di y

ATTIVITA': implementare la classe *Coppia*

La classe *Coppia*

File *Coppia.java*:

```
import java.io.*;
public class Coppia
{   private int x,y;
    public void getx()
    {  System.out.println ("x="+x);   }
    public void gety()
    {  System.out.println ("y="+y);   }
    public Coppia()
    {   x=0;   y=0;    }
    public Coppia (int a, int b)
    {   x=a;   y=b;    }
Segue…
```

Attributi privati

Metodi pubblici

I **costruttori** (con parametri o senza) della classe *Coppia* servono ad inizializzare l'oggetto.

La classe *Coppia*

```
…Segue
public static void main(String args[ ])
   {  int a,b;
      ....... // creazione oggetto Tastiera
      try
      {       a=Integer.parseInt(Tastiera.readLine());
              b=Integer.parseInt(Tastiera.readLine());
              Coppia c1 = new Coppia(a, b), c2=new Coppia();
              c1.getx();      c1.gety();
              c2.getx();      c2.gety();
       }      /* end try */
      catch (Exception e)  {  System.out.print("\n Si è verificato un errore");
   }  // end catch
    }  // end main
}   // end class
```

Creazione oggetti *c1* e *c2* di classe *Coppia* mediante i due costruttori

ATTIVITA': creare la classe *Coppia* e la classe *testCoppia (contenente il* **main()**) in due file separati e verificarne il funzionamento.

Classi esterne

file Coppia.java

```
class Coppia
{ private int x, y;
  public void getx()
  { System.out.println("x="+x);}
  public void gety()
  {System.out.println("y="+y);}
    public Coppia(int a, int b)
    { x=a; y=b; }
}
```

Soluzione con **classi esterne**

file testCoppia.java

```
import java.io.*;
public class TestCoppia
{   public static void main(String args[ ])
    { int a, b;
      creazione oggetto tastiera;
      try
      { a=Integer.parseInt(Tastiera.readLine());
        b=Integer.parseInt(Tastiera.readLine());
        Coppia c = new Coppia(a, b);
        c.getx();   c.gety();
      } /* fine try */
      catch (Exception E)
      { System.out.println("Errore"); }
    } /* fine main */
} /* fine classe */
```


Classi esterne

Perché in genere conviene la tecnica con **classi esterne**?

- mantiene tutte le classi definite dall'utente in file separati (**modularità**), *compilabili singolarmente*. Una volta compilato il programma sorgente, è sufficiente disporre della versioni **.class**
- il file **.class** può essere copiato in altre cartelle ed *utilizzato da altre applicazioni* (**riusabilità**)
- per *modificare una classe* si interviene solo su questa e non su tutta l'applicazione (facilità di manutenzione)
- mantiene separato il programma di *testing* che utilizza le classi.

Dichiarazione degli attributi

Come si è visto, gli **attributi** si dichiarano con

tipo ident;

dove

- *tipo* è il tipo di dato dell'attributo
- *ident* è l'identificatore assegnato all'attributo dal programmatore

Definizione dei metodi

Sempre dall'esempio, si vede che i metodi si **definiscono** con

tipo ident (*lista_par*)
{ *istruzioni*;
}

dove

- *tipo* indica il tipo del valore ritornato
- *ident* indica il nome assegnato al metodo
- *lista_par* indica la lista dei parametri, ciascuno espresso come

tipo ident

tipo indica il tipo di dato a cui appartiene *ident*

Metodi *get* e *set*

Per consuetudine, i metodi accessori e modificatori hanno un prefisso standard e precisamente:

- **get…()** serve ad emettere il valore di un attributo
- **set…()** serve ad impostare il valore di un attributo

Nell'esempio della classe *Coppia*, il metodo

public void getx() esegue la stampa di x

public void gety() esegue la stampa di y

ATTIVITA': modificare la classe *Coppia* in modo da prevedere i metodi *setx* (**int** x) e *sety* (**int** y) che permettono di impostare, rispettivamente, il valore dell'attributo *x* ed *y*. Modificare la classe *testCoppia* in modo da verificare la funzionalità dei metodi aggiunti.

Il metodo costruttore

Tra i metodi figurano i **costruttori**:

public Coppia () // costruttore senza parametri

public Coppia (**int** a, **int** b) // costruttore con parametri

I costruttori

- servono ad *inizializzare* i valori degli attributi al momento della creazione dell'oggetto;
- devono avere lo *stesso nome della classe*;
- non è previsto *alcun valore di ritorno*;
- possono essere *con parametri o senza*;
- possono essere *presenti in più forme nella stessa classe* (v. sopra)

Utilizzo degli oggetti

La **creazione di un oggetto** di classe *Coppia* si effettua con l'istruzione

Coppia c **= new** Coppia (a,b);

che si dice **istanza** della classe e grazie al costruttore, mette a disposizione l'oggetto *c*.

L'oggetto *c:*

- *occupa memoria*, a differenza della classe, che è un modello astratto;
- consente l'**accesso ad un membro** con la notazione "." (*dot notation*) per cui
  - *c.getx()* attiva il metodo *getx*()
  - *c.gety()* attiva il metodo *gety*().

In generale l'attivazione di un metodo si esegue con la sintassi

oggetto.ident (*lista_param*)

Argomenti

- Astrazione
- Implementazione
- Programmazione ad oggetti
- Il linguaggio Java
- Struttura di una classe
- Funzionamento di una classe
- Dichiarazione di una classe
- Implementazione di classi
- La classe Coppia
- Classi esterne
- Dichiarazione degli attributi
- Definizione dei metodi
- Metodi *get* e *set*
- Il metodo costruttore
- Utilizzo degli oggetti

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