Università della Calabria – Polo didattico di Crotone
Corso di Laurea in Ingegneria Gestionale

Programmazione Orientata agli Oggetti

A.A. 2008/2009

Introduzione alla programmazione orientata agli oggetti

Ing. Riccardo Ortale

Obiettivo del corso

Obiettivo del corso è acquisire le conoscenze necessarie per realizzare applicazioni informatiche secondo il paradigma della **programmazione orientata agli oggetti**, utilizzando il linguaggio **Java**.

Contenuti delle lezioni:

- Studio della programmazione orientata agli oggetti e sviluppo di applicazioni basate su tale paradigma, facendo uso dei meccanismi e delle librerie standard di Java.

Contenuti delle esercitazioni:

- Progettazione ed implementazione di applicazioni informatiche basate sui concetti presentati a lezione.

Informazioni sul corso

Docente:

Riccardo Ortale

Esercitatore:

Ing. Noemi Tempora

Libro di testo:

Cay S. Horstmann, “Concetti di informatica e fondamenti di Java”, Apogeo, quarta edizione, 2007.

Informazioni sull'esame

Propedeuticità:

- Fondamenti di Informatica.

Modalità di svolgimento dell'esame:

- Prova scritta obbligatoria.
- Prova orale facoltativa (obbligatoria in caso di ammissione con riserva).

La programmazione orientata agli oggetti

- Nella **programmazione orientata agli oggetti** (**POO**) un programma è composto da un insieme di entità software, denominate **oggetti**, che interagiscono secondo determinati schemi. Esempi di linguaggi che supportano la programmazione orientata agli oggetti sono **Java** e **C++**.

- La programmazione orientata agli oggetti differisce dalla cosiddetta **programmazione imperativa**, basata sul concetto di **istruzione**: un programma consiste in una sequenza di istruzioni che specificano in modo dettagliato le operazioni che devono essere eseguite dall'elaboratore per risolvere il problema dato. Esempi di linguaggi imperativi sono **Fortran**, **Pascal** e **C**.

- E' possibile utilizzare i linguaggi orientati agli oggetti per realizzare programmi prevalentemente imperativi (*come nel corso di Fondamenti di Informatica*), ma in tal modo non si sfruttano i vantaggi derivanti dall'uso dei meccanismi propri della programmazione orientata agli oggetti.

Oggetti

I linguaggi orientati agli oggetti (*object oriented*) sono basati sul concetto di ***oggetto***.

Un *oggetto* è una entità software che incapsula un insieme di ***attributi*** (o *dati*) e comprende un insieme di ***operazioni*** (o *metodi* o *funzioni*) che manipolano i dati.

L'insieme dei valori assunti dagli attributi costituisce lo ***stato*** dell'oggetto.

L'insieme delle operazioni definiscono il ***comportamento*** dell'oggetto.

Oggetti

Gli oggetti sono generalmente ***entità passive***: in un dato istante nell'esecuzione di un programma un solo oggetto è ***attivo*** (ovvero *in esecuzione*).

Un oggetto diventa attivo quando riceve l'invocazione di un metodo da parte di un altro oggetto. Mentre l'oggetto ricevente è attivo, il richiedente è passivo (non in esecuzione) in attesa dei risultati del metodo invocato.

Una volta che i risultati sono stati inviati al richiedente, l'oggetto ricevente torna in uno stato passivo ed il richiedente prosegue la sua esecuzione.

Classi

Ogni oggetto è istanza di una determinata ***classe***.

Una *classe* definisce una tipologia di oggetti, che possono essere descritti dallo stesso insieme di attributi e dallo stesso insieme di metodi.

Due istanze della stessa classe condividono le stesse ***proprietà strutturali*** (attributi) e le medesime ***proprietà comportamentali*** (metodi), ma sono comunque oggetti distinti.

Classi

Il concetto di classe fornisce al programmatore uno strumento per creare *nuovi tipi* che possono essere usati facilmente come i *tipi predefiniti* (come *int*, *float*, *double*, etc.).

Un tipo è la rappresentazione concreta di un *concetto*. Ad es., il tipo predefinito *double*, insieme alle sue operazioni +, -, * e /, fornisce una rappresentazione del concetto di numero reale.

Una classe è, quindi, un nuovo tipo definito dal programmatore. Si progetta un nuovo tipo per fornire la definizione di un concetto, che non è definito direttamente tra i tipi predefiniti.

Ad esempio, può essere utile definire le classi *Resistore*, *Condensatore*, o *Diodo* in un programma per l'analisi o la progettazione di circuiti elettronici.

Proprietà fondamentali della POO

Classificazione. Ogni oggetto è istanza di una classe, che rappresenta un insieme di oggetti aventi le stesse proprietà strutturali e comportamentali.

Incapsulamento. Meccanismo che rende possibile il principio dell'*information hiding*, ovvero la capacità degli oggetti di nascondere al mondo esterno la propria organizzazione in termini di struttura e logica interna.

Ereditarietà. Meccanismo attraverso il quale una classe incorpora struttura e comportamento definiti da una classe più generale. La classe da cui si eredita è detta *superclasse*, mentre la classe discendente è detta *sottoclasse*.

Polimorfismo. Capacità di supportare operazioni con la medesima firma e comportamenti diversi, situate in classi diverse ma derivanti da una stessa superclasse.

Esempio: una classe e due oggetti

```
// Esempio1.java

class Data {

    public int giorno;
    public int mese;
    public int anno;

    public void init(int g, int m, int a) {
      giorno = g;
      mese = m;
      anno = a;
    }

}
```

attributi

metodi

```
public class Esempio1 {

 public static void main (String args[]) {

   Data a = new Data();

   a.init(1,1,1980);

   Data b = new Data();

   b.init(15,8,2006);

   System.out.println(a.mese); // 1

   System.out.println(a.anno); // 1980

   System.out.println(b.giorno); // 15

   b.giorno = a.giorno + 7;

   System.out.println(b.giorno); // 8

 }

}
```

Campi pubblici e campi privati

```
// Esempio2.java

class Data {

    private int giorno;
    private int mese;
    private int anno;

    public void init(int g, int m, int a) {
      giorno = g;
      mese = m;
      anno = a;
    }

    public void stampa() {
      System.out.println(giorno+"-"+
        mese+"-"+anno);
    }
}
```

```
public class Esempio2 {

 public static void main (String args[]) {

   Data a = new Data();

   a.init(1,1,1980);

   System.out.println(a.giorno); // errore

   a.mese = 3; // errore

   a.stampa(); // 1-1-1980

   Data b = new Data();

   b.init(15,8,2006);

   b.giorno = a.giorno + 7; // errore

   b.stampa(); // 15-8-2006

 }
}
```

Campi pubblici e campi privati

La dichiarazione di variabile o metodo (sia statico che di istanza) può essere preceduta da opportuni *modificatori,* tra cui i ben noti **public** oppure **private** (oppure da nessuno dei due). Cosa significano? Quali sono le differenze?

Modificatore	**Accessibilità:** *lettura e scrittura* per variabili (<obj>.<variabile>) *invocazione* per metodi (<obj>.<metodo>(<parametri>))
private	Accessibile solo all'interno della classe di <obj>
nessuno	Accessibile solo nel ***package*** che contiene la classe di <obj>
protected	Accessibile nel package che contiene la classe di <obj>, e in tutte le classi che *ereditano* da essa (anche se dichiarate in package differenti)
public	Sempre, cioè ovunque la classe di <obj> sia accessibile

Nella tabella i modificatori sono elencati in ordine crescente di visibilità. Si noti che:
• per garantire l'incapsulamento delle informazioni, le variabili devono essere dichiarate **private**;
• il nome del modificatore **protected** è fuorviante, perché consente una visibilità persino maggiore dell'assenza di modificatore.

Costruttori

```
// Esempio3.java

class Data {

    private int giorno;
    private int mese;
    private int anno;

    public Data(int g, int m, int a) {
      giorno = g;
      mese = m;
      anno = a;
    }

    public void stampa() {
      System.out.println(giorno+"-"+
        mese+"-"+anno);
    }
}
```

costruttore

```
public class Esempio3 {

 public static void main (String args[]) {

   Data a = new Data(1,1,1980);

   a.stampa(); // 1-1-1980

   Data b = new Data(15,8,2006);

   b.stampa(); // 15-8-2006

 }

}
```

Nota: I costruttori hanno lo stesso nome della classe e non specificano alcun valore di ritorno.

Metodi di accesso

```
// Esempio4.java

class Data {
    private int giorno;
    private int mese;
    private int anno;
    public Data(int g, int m, int a) {
      giorno = g;
      mese = m;
      anno = a;
    }
    public int getGiorno() {
      return giorno;
    }
    public int getMese() {
      return mese;
    }
    public int getAnno() {
      return anno;
    }
}
```

metodi di accesso

```
public class Esempio4 {
 public static void main (String args[]) {
   Data a = new Data(3,5,75);
   System.out.println(a.giorno); // errore
   System.out.println(a.getGiorno()); // 3
   System.out.println(a.getMese()); // 5
   System.out.println(a.getAnno()); // 75
 }
}
```

Una classe Esame

```
// Esempio5.java

class Esame {
  private String nome;
  private int voto;
  private boolean lode;

  public Esame(String n, int v, boolean l) {
      nome = n;
      voto = v;
      lode = l;
  }

  public void stampa() {
    System.out.print(nome+": "+voto);
    if (lode) {
      System.out.print(" e lode");
    }
    System.out.println();
  }
}
```

```
public class Esempio5 {
 public static void main (String args[]) {
   Esame e1 =
     new Esame("POO", 28, false);
   e1.stampa(); // POO: 28


   Esame e2 =
     new Esame("Calcolo 2", 30, true);
   e2.stampa(); // Calcolo 2: 30 e lode
 }
}
```

L’operatore this

```
class Esame {
  private String nome;
  private int voto;
  private boolean lode;

  public Esame(String nome, int voto, boolean lode) {
    nome = nome; // errore
    this.nome = nome;
    this.voto = voto;
    this.lode = lode;
  }

  void stampa() {
    System.out.print(nome+": "+voto);
    if (lode) {
      System.out.println(" e lode");
    }
    System.out.println();
  }
}
```

Costruttori sovraccaricati

```
// Esempio7.java
class Data {
  private int giorno;
  private int mese;
  private int anno;
  public Data(int g, int m, int a) {
    giorno = g;
    mese = m;
    anno = a;
  }
  public Data(int a) {
    giorno = 1;
    mese = 1;
    anno = a;
  }
  public Data() {
    giorno = 1;
    mese = 1;
    anno = 1980;
  }
  public void stampa() {
    System.out.println(giorno+"-"+
      mese+"-"+anno);
  }
} // fine classe Data

public class Esempio7 {
 public static void main (String args[]) {
  Data a = new Data(3,3,2003);
  a.stampa(); // 3-3-2003
  Data b = new Data(1990);
  b.stampa(); // 1-1-1990
  Data c = new Data();
  c.stampa(); // 1-1-1980
 }
}
```

Membri statici

```
// Esempio8.java

class MyClass {
    public int x;
    public static int y = 0;
    public MyClass(int x) {
      this.x = x;
      y++;
    }
}
```

```
public class Esempio8 {
 public static void main (String args[]) {
  System.out.println(MyClass.y); // 0
  MyClass a = new MyClass(8);
  System.out.println(MyClass.y); // 1
  MyClass b = new MyClass(4);
  System.out.println(MyClass.y); // 2
  MyClass c = new MyClass(6);
  System.out.println(MyClass.y); // 3
 }
}
```

Nota: I membri statici sono utilizzati per condividere informazioni tra tutti gli oggetti della classe

Metodi statici

```
// Esempio9.java

class MyClass {
  public int x;
  public static int y = 0;
  public MyClass(int x) {
    this.x = x;
    y++;
  }
  public static long f() {
    x = x+1; // errore: campo non statico
    return y*y;
  }
}
```

```
public class Esempio9 {
 public static void main (String args[]) {
  System.out.println(MyClass.y); // 0
  MyClass a = new MyClass(8);
  System.out.println(MyClass.y); // 1
  MyClass b = new MyClass(4);
  System.out.println(MyClass.y); // 2
  System.out.println(MyClass.f()); // 4
 }
}
```

Nota: I metodi statici non possono accedere a membri non statici

Assegnamenti tra oggetti

```
// Esempio10.java

class MyClass {
  public int x;
  public int y;
  public MyClass(int x, int y) {
    this.x = x;
    this.y = y;
  }
  public void stampa() {
    System.out.println(x+" "+y);
  }
}
```

```
public class Esempio10 {
 public static void main (String args[]) {
  MyClass a = new MyClass(2,3);
  MyClass b = a;
  b.stampa(); // 2 3
  b.x = 10;
  a.stampa(); // 10 3
 }
}
```

Costruttori di copia

```
// Esempio11.java

class MyClass {
  public int x;
  public int y;
  public MyClass(int x, int y) {
     this.x = x;
     this.y = y;
  }
  public MyClass(MyClass obj) {
    this.x = obj.x;
    this.y = obj.y;
  }
  public void stampa() {
    System.out.println(x+" "+y);
  }
}
```

costruttore di copia

```
public class Esempio11 {
 public static void main (String args[]) {
  MyClass a = new MyClass(2,3);
  MyClass b = new MyClass(a);
  b.stampa(); // 2 3
  b.x = 10;
  a.stampa(); // 2 3
 }
}
```

Array di oggetti

```
// Esempio12.java

class MyClass {
  public int x;
  public int y;
  public MyClass(int x, int y) {
     this.x = x;
     this.y = y;
  }
  public void stampa() {
    System.out.println(x+" "+y);
  }
}
```

```
public class Esempio12 {
 public static void main (String args[]) {
  int n = 4;
  MyClass v[] = new MyClass[n];
  for (int i=0; i<n; i++) {
    v[i] = new MyClass(i,i*2);
  }
  for (int i=0; i<n; i++) {
    v[i].stampa();
  }
 }
}
```