Fondamenti di Java

Il linguaggio cult per la programmazione a oggetti in quattro lezioni

La nascita, la metodologia, lo sviluppo e le caratteristiche uniche che hanno fatto di Java il linguaggio più usato in rete

Tipi di metodi

- Costruttori: permettono di istanziare gli oggetti a partire dalle classi, assegnando valori agli attributi.
- Modificatori: permettono di modificare lo stato dell'oggetto cioè i valori dei suoi attributi.
- Query: restituiscono il valore degli attributi dell'oggetto

I metodi costruttori

- La creazione dell'oggetto si ottiene tramite l'invocazione del costruttore preceduto dall'operatore new.
- I costruttori sono metodi speciali che hanno la funzione di allocare nello heap un'area di memoria delle dimensioni adatte a contenere l'oggetto ed eventualmente di assegnare dei valori agli attributi.

Caratteristiche dei costruttori

- I costruttori hanno alcune caratteristiche specifiche
- Hanno lo stesso nome della classe
- Non hanno tipo restituito
- Sono sempre i primi metodi ad essere dichiarati
- Se sono più di uno, si distinguono per numero e tipo di parametri, cioè per la segnatura o firma o prototipo del metodo

Creare un oggetto

- Poiché i costruttori sono indispensabili per la creazione degli oggetti, se in una classe non se e dichiara almeno uno, Java crea automaticamente un costruttore che istanzia un oggetto vuoto.
- L'istruzione per creare un oggetto r appartenente alla classe Rettangolo sarà quindi:

  Rettangolo r=new Rettangolo();

Istanziare gli oggetti

- Un oggetto istanziato grazie al costruttore della classe costituisce in Java una variabile del tipo della classe cui l'oggetto appartiene.
- Per questo motivo, non si dichiarano in genere oggetti del tipo di una classe all'interno della classe stessa, anche se è possibile farlo
- Per dichiarare oggetti di una classe è necessario ricorrere a una classe esterna
- La classe speciale String permette di dichiarare oggetti stringa senza scrivere il costruttore; tuttavia non si tratta di un'eccezione alla regola,ma solamente di una contrazione dell'istruzione completa
- String s=new String();

Invocare metodi ed attributi

- Poichè ogni oggetto derivato da una classe possiede tutta le caratteristiche e le capacità definite per la classe, per ciascun oggetto sono definiti tutti i metodi e tutti gli attributi previsti per la classe.
- Per invocare un metodo o un attributo si ricorre alla stessa *dot notation* già vista per le strutture nel linguaggio C.
- Questo perché una classe si può immaginare come una struttura contenete funzioni oltre che variabili.

La dot notation

- Per accedere a un attributo base di un oggetto r di una classe Rettangolo si scriverà perciò:

  r.base;

- Per accedere al metodo calcolaArea():

  r.calcolaArea();

- Se si vuole accedere a un metodo o a un attributo della classe stessa in cui sono definiti si non è necessario ricorrere alla dot notation, ma in caso di ambiguità si può ricorrere all'operatore this:

  this.base;

Principi fondamentali di OOP

- La OOP si basa su tre principi fondamentali:
  - L'incapsulamento
  - L'ereditarietà
  - Il polimorfismo
- L'incapsulamento consiste nel racchiudere tutto il codice all'interno delle classi, inserendo in questi moduli i dati e il codice che opera su di essi

Incapsulamento

- Lo scopo dell'incapsulamento è duplice
- Si vogliono creare moduli software indipendenti
- Si vuole proteggere il codice dalle intrusioni esterne
- Se un attributo è dichiarato *private*, si può accedere ad esso solo attraverso opportuni metodi query

public, private, protected

- Lo scopo di includere il codice nelle classi è la possibilità di stabilire a quali elementi della classe è possibile accedere dall'esterno
- A questo scopo si utilizzano tre specificatori: *public, private, protected*
- Posti davanti a un attributo o a un metodo, stabiliscono se è possibile accedere ad esso al di fuori dei confini della classe.
- Questo principio è noto come *information hiding*

Information hiding

Attributi e metodi possono essere definiti:

- *public*: un attributo o un metodo che può essere invocato al di fuori della classe in cui è dichiarato
- *protected*: è un grado di protezione intermedio, un attributo o un metodo può essere invocato solo nella classe stessa in cui è dichiarato e nelle eventuali sottoclassi che sono state ottenute estendendola
- *private*: un attributo o un metodo può essere invocato solo nella classe stessa in cui è dichiarato

Se nessuno specificatore è indicato, l'accesso è ristretto alle classi appartenenti allo stesso package (*public*)

Classe Persona

```
public class Persona {
  String nome;
  int n;

  void presenta(){
  System.out.println("Buongiorno, mi chiamo
    "+nome+"e ho "+n+" anni");
  }
}
```

Classe ProvaPersona

```
public class ProvaPersona {
  public static void main(String[] args) {
    Persona p1=new Persona();
    p1.nome="Mario Rossi";
    p1.n=20;
    p1.presenta();
  }
}
```

I metodi nella classe Rettangolo

```
public class Rettangolo {

    private double base;
    private double altezza;

    Rettangolo(){}

    Rettangolo(double base, double altezza){
        this.base=base;
        this.altezza=altezza;}

void inserisciBase(){
base=Double.parseDouble(JOptionPane.showInputDialog("Inserisci base del
      rettangolo"));}

void inserisciAltezza(){
altezza=Double.parseDouble(JOptionPane.showInputDialog("Inserisci altezza del
      rettangolo"));}
```

I metodi nella classe Rettangolo

```
void visualizzaBase(){
 JOptionPane.showMessageDialog(null, "La base del rettangolo
    misura "+base);}

void visualizzaAltezza(){
 JOptionPane.showMessageDialog(null, "L'altezza del
    rettangolo misura "+altezza);}

double calcolaPerimetro(){
 return 2*(base+altezza);}

double calcolaArea(){
 return base*altezza;}

}
```

La classe ProvaRettangolo

```
import javax.swing.JOptionPane;

public class ProvaRettangolo {

   public static void main(String[] args) {

     Rettangolo r1=new Rettangolo();
     double perimetro;
     double area;

     r1.inserisciBase();
     r1.inserisciAltezza();

    perimetro=r1.calcolaPerimetro();
    JOptionPane.showMessageDialog(null, "Il perimetro del rettangolo misura "+perimetro);
    area=r1.calcolaArea();
   JOptionPane.showMessageDialog(null, "L'area del rettangolo misura "+area);

   }
}
```

La classe ProvaRettangolo

```
import javax.swing.JOptionPane;

public class ProvaRettangolo {

    public static void main(String[] args) {

      Rettangolo r2=new Rettangolo(4.2, 6.8);
      double perimetro;
      double area;

     r2.visualizzaBase();
     r2.visualizzaAltezza();

    perimetro=r2.calcolaPerimetro();
    JOptionPane.showMessageDialog(null, "Il perimetro del rettangolo misura "+perimetro);
    area=r2.calcolaArea();
    JOptionPane.showMessageDialog(null, "L'area del rettangolo misura "+area);

  }
}
```

Variabili primitive e di riferimento

- Java è un linguaggio di programmazione fortemente tipizzato.
- Come in C, anche in Java ogni variabile per essere usata deve essere dichiarata
- In Java sono previsti due categorie di variabili, le variabili primitive e le variabili di riferimento (*reference*) agli oggetti.
- Le variabili primitive sono del tutto analoghe a quelle del C
- Le variabili di riferimento possono essere pensate come frecce (riferimenti) agli oggetti

I reference e le variabili di istanza

- Una variabile di tipo primitivo viene memorizzata direttamente nello stack.
- Una variabile rifrimento assegnata a un oggetto mantiene nello stack soltanto il *reference* a una locazione dello heap.
- Il *reference* di una variabile è una cella di memoria che contiene un indirizzo che rappresenta una locazione precisa nello heap.
- In tale locazione viene allocato l'oggetto vero e proprio
- La creazione dell'oggetto si ottiene tramite l'invocazione del costruttore della classe preceduto dall'operatore new

I tipi primitivi e le classi wrapped

- In Java i tipi primitivi per le variabili sono derivati dal linguaggio C:
  - int
  - float
  - double
  - char
  - boolean
- Per ognuno dei tipi primitivi esiste la corrispondente classe *wrapped*

Le classi wrapped

- In Java esistono classi predefinite, che possono essere utilizzate come in C possono essere utilizzate le istruzioni e le funzioni di libreria.
- Per ognuno dei tipi primitivi di Java, esiste una corrispondente classe predefinita, detta classe wrapped che consente di creare oggetti analoghi alle variabili primitive e che mette a disposizione una serie di metodi utili che operano su questi dati.
- Le variabili di questo tipo sono oggetti.

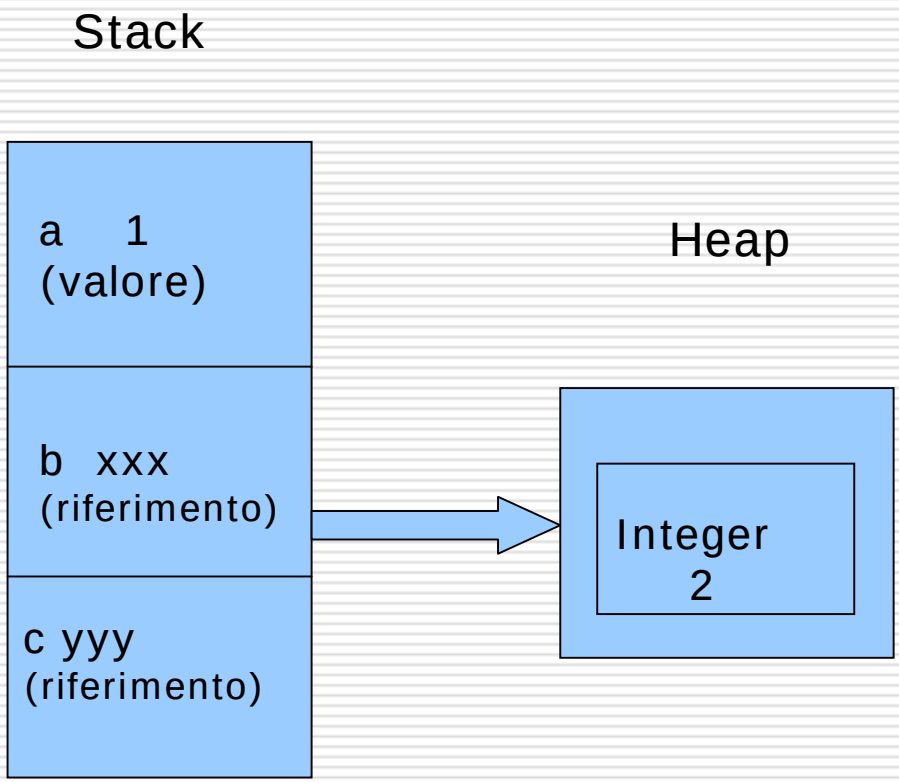
Un esempio di uso delle variabili

Date le variabili:

int a=1;
è una variabile di tipo primitivo

Integer b=new Integer(2);
è un riferimento a un oggetto di tipo Integer cui viene assegnato il valore 2

Integer c;
è un riferimento a un oggetto di tipo Integer



Un esempio di uso delle variabili

c=b;

i riferimenti b e c puntano a un'unica locazione della memoria heap (di indirizzo xxx)
Assegna anche a c il valore 2

c=3;

assegna anche a b il valore 3, poiché i due reference puntano allo stesso indirizzo
modificando il contenuto, entrambe le variabili risultano modificate.

c=a;

c punta alla locazione che contiene il valore di a

c=3;

L'effetto dell'istruzione c=3 equivale alla chiamata del costruttore:
c=new Integer(3);
a c viene assegnata una nuova locazione
a conserva il suo valore precedente (1)

Confronto tra variabili di istanza

Per confrontare due variabili di istanza:

- l'operatore '==' confronta gli indirizzi di memoria referenziati
- Una variabile b è dunque diversa da una variabile c se punta a una locazione di memoria differente.
- il metodo *equals()* confronta i valori contenuti nello heap da entrambe le variabili

La parola chiave static

- Oltre a public, private e protected esiste in Java la parola chiave static.
- Un metodo o un attributo è definito static quando è possibile invocarlo anche se non sono stati dichiarati oggetti di quella classe.
- Un metodo o un attributo statico può essere invocato sulla classe stessa.
- Esistono infine classi particolari denominate abstract o classi astratte, di cui non possono essere dichiararti oggetti e tutti i metodi sono riferiti alla classe.

Classe Cliente

```
public class Cliente {
  String nome;
  float soldi;
  void compra(float prezzo){
    if(soldi>=prezzo)
       soldi=soldi-prezzo;
    else
       System.out.println(“Credito insufficiente”);}
  void mostraCredito(){
  System.out.println(“Il cliente "+nome+" ha "+soldi+"
     euro");
  }
}
```

Classe Prodotto

```
public class Prodotto {
  String nome;
  float prezzo;
  void mostraPrezzo(){
   System.out.println("Il prodotto "+"
    costa " +prezzo);}
   }
}
```

Classe Vendita

```
public class Vendita {
  public static void main(String[] args) {
    Cliente c=new Cliente();
    Prodotto p=new Prodotto();
    c.nome="Mario Rossi";
    c.soldi=500;
    p.mostraPrezzo();
    c.compra(p.prezzo);
    c. mostraCredito();}
}
```