Fondamenti di Architetture e Programmazione
III Compitino (Recupero)

2 febbraio 2006

- Usate il retro dei fogli per scrivere lo svolgimento degli esercizi di programmazione.
- Nei seguenti quesiti, quando vi è richiesto di scrivere un programma, potete limitarvi al *corpo* del metodo main, assumendo se necessario che in e out siano due variabili di classe ConsoleInputManager e ConsoleOutputManager (rispettivamente), già dichiarate e inizializzate.

1. Scrivete un programma che legga una sequenza di esattamente 10 double, e stampi solo quelli il cui valore arrotondato è pari; ricordiamo che il metodo statico round(**double** x) della classe Math restituisce un long che è ottenuto arrotondando il valore di x.

   Ecco un esempio di esecuzione (le parti in grassetto sono state inserite dall'utente):

   `Valore 1:` **3.12**
   `Valore 2:` **7.56**
   `Valore 3:` **8.1**
   `...`

   ```
   7.56
   8.1
   ...
   ```

(Svolgimento sul retro)

```java
double[] x = new double[ 10 ];
for ( int i = 0; i < 10; i++ )
 x[ i ] = in.readDouble( "Valore " + ( i + 1 ) + ": " );
for ( int i = 0; i < 10; i++ )
 if ( Math.round( x[ i ] ) % 2 == 0 )
  out.println( x[ i ] );
```

2. Supponete che esista una classe Trapezio i cui oggetti corrispondono a dei trapezi rettangoli; questa classe ha un costruttore pubblico

   ```
   public Trapezio( double bMagg, double bMin, double h )
   ```

   che costruisce un trapezio con date base maggiore, base minore e altezza, e un metodo pubblico

   ```
   public double getArea()
   ```

   che restituisce l'area (base maggiore più base minore per altezza diviso due).

   Scrivete un programma che operi come segue:

   (a) chieda all'utente di inserire un numero intero, diciamo N;

   (b) chieda all'utente di inserire un valore double (la soglia), diciamo r;

   (c) dichiari un array di N trapezi e lo riempia con dati inseriti dall'utente (che, per ciascun trapezio, deve inserire la base maggiore, la base minore e l'altezza);

   (d) stampi i soli trapezi *la cui area è maggiore di* r (potete assumere che la classe Trapezio disponga di un opportuno metodo toString()), ciascuno accompagnato dalla sua area.

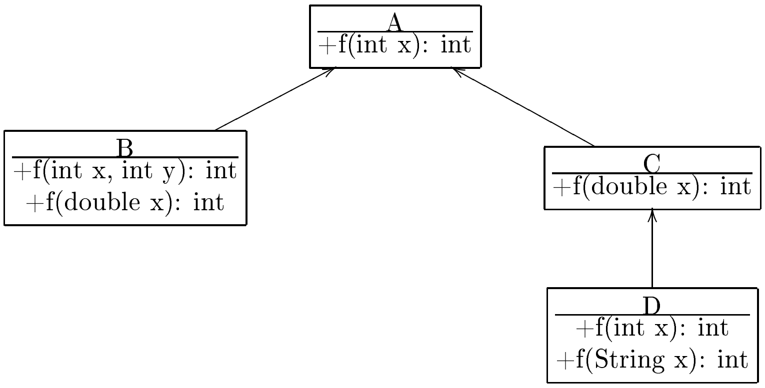
   Ecco un esempio di esecuzione (le parti in grassetto sono state inserite dall'utente):

   ```
   Inserisci un intero: 3
   Inserisci una soglia: 500

   Base maggiore: 33.2
   Base minore: 25.4
   Altezza: 17.2
   Base maggiore: 53
   Base minore: 15.3
   Altezza: 13.2
   Base maggiore: 33.1
   Base minore: 1.4
   Altezza: 100

   Trapezio [B=33.2,b=25.4,h=17.2] 503.96
   Trapezio [B=33.1,b=1.4,h=100] 1725
   ```

(Svolgimento sul retro)

```
int n = in.readInt( "Inserisci␣un␣intero:␣" );
double soglia = in.readDouble( "Inserisci␣una␣soglia:␣" );
Trapezio[] a = new Trapezio[ n ];
int i;
for ( i = 0; i < n; i++ ) {
 double bMagg = in.readDouble( "Base␣maggiore:␣" );
 double bMin = in.readDouble( "Base␣minore:␣" );
 double h = in.readDouble( "Altezza:␣" );
 a[ i ] = new Trapezio( bMagg, bMin, h );
}

for ( int i = 0; i < 10; i++ )
 if ( a[ i ].getArea() > r )
  out.println( a[ i ] + "␣" + a[ i ].getArea() );
```

3. Considerate la seguente gerarchia dei tipi (rappresentata mediante un diagramma UML):



e ipotizzate che tutte le classi abbiano un costruttore pubblico senza argomenti. Assumete le seguenti definizioni e inizializzazioni:

```
A a1, a2, a3, a4;
B b;
C c1, c2;
D d;

a1 = new A(); a2 = new B(); a3 = new C(); a4 = new D();
b = new B();
c1 = new C(); c2 = new D();
D d = new D();
```

Per ciascuno dei seguenti assegnamenti, dite se l'assegnamento è consentito, oppure se richiede (per poter essere compilato) un operatore di cast nel secondo membro (e in tal caso indicate l'assegnamento corretto), oppure se l'assegnamento non è consentito nemmeno con un cast (nel senso che anche con un cast produrrebbe un errore di compilazione o un'eccezione in esecuzione):

- `b=a1`: **No**
- `a3=c1`: **Ok**
- `b=c1`: **No**
- `c1=b`: **No**
- `c1=a2`: **No**
- `c1=d`: **Ok**
- `c2=a3`: `c2=(C)a3`

4. Facendo riferimento al precedente diagramma UML e alle variabili definite e inizializzate come sopra, considerate le seguenti invocazioni di metodo. Per ciascuna di esse dovete dire se è consentita e in caso affermativo dovete indicare il nome della classe che contiene il metodo che verrà effettivamente eseguito:

   - `a1.f(3,4)`: **No**
   - `a2.f(3,4)`: **No**
   - `b.f(3)`: **A:f(int)**
   - `a3.f(5)`: **A:f(int)**
   - `a4.f("pippo")`: **No**
   - `d.f("pippo")`: **D:f(String)**
   - `d.f(3,4)`: **No**
   - `d.f(3.4)`: **C:f(double)**

5. La classe Integer contiene un metodo statico

```
public static String toOctalString( int x )
```

che dato un intero `x` lo converte in una stringa che ne fornisce la rappresentazione ottale. Avete una variabile intera `i` e volete determinare se la lunghezza della sua rappresentazione ottale è maggiore di 6, mettendo il risultato (true o false) nella variabile booleana `z`; scrivete l'istruzione di assegnamento a `z`:

```
z=Integer.toOctalString( i ).length() > 6;
```

6. Considerate la seguente funzione definita ricorsivamente sugli interi:

$$f(x) = \begin{cases} x + f(x+1) & \text{se } x \text{ è multiplo di 3} \\ x + 1 & \text{altrimenti} \end{cases}$$

Scrivete un metodo statico con intestazione

```
public static int f( int x )
```

per il calcolo di f.

```
public static int f( int x ) {
 if ( x % 3 == 0 ) return x+f(x+1);
 return x+1;
}
```

7. Definite una classe di nome Trapezio i cui oggetti corrispondono a dei trapezi rettangoli; questa classe deve avere:

   - un costruttore pubblico

     ```
     public Trapezio( double bMagg, double bMin, double h )
     ```

     che specifica la base maggiore, la base minore e l'altezza;
   - un costruttore pubblico

     ```
     public Trapezio( double baseMagg, double altezza )
     ```

     che assume che la base minore sia la metà della base maggiore;
   - un metodo

     ```
     public double getArea()
     ```

     che fornisce l'area, calcolata come $(B + b)h/2$ (dove B è la base maggiore, b è la base minore e h è l'altezza);
   - un metodo

     ```
     public boolean equals( Trapezio t )
     ```

     che restituisce true se e solo se il trapezio su cui è invocato è uguale a `t`.

(Svolgimento sulla facciata seguente)

```java
class Trapezio {
 private double bMagg, bMin, h;

 public Trapezio(double bMagg, double bMin, double h) {
  this.bMagg = bMagg;
  this.bMin = bMin;
  this.h = h;
 }

 public Trapezio(double bMagg, double h) {
  this( bMagg, bMagg / 2, h );
 }

 public double getArea() {
  return (bMagg+bMin)*h/2;
 }

 public boolean equals(Trapezio t) {
  return bMagg==t.bMagg && bMin==t.bMin
   && h==t.h;
 }
}
```

8. Definite una classe di nome TrapezioColorato che estende Trapezio e i cui oggetti corrispondono a dei trapezi colorati. Questa classe deve avere:

   - un costruttore pubblico

     ```
     public TrapezioColorato( double bMagg, double bMin, double h, String colore
     ```

     che specifica i dati del trapezio e il suo colore;

   - un metodo

     ```
     public boolean stessaColore( TrapezioColorato t )
     ```

     che restituisce true se e solo se il trapezio su cui è invocato e t hanno lo stesso colore.

```
class TrapezioColorato extends Trapezio {
 private String c;

 public TrapezioColorato(double bMagg, double bMin, double h, String colore) {
  super(bMagg,bMin,h);
  this.c = colore;
 }

 public boolean stessoColore( TrapezioColorato t ) {
  return c.equals( t.c );
 }
}
```