Programmazione ad Oggetti – Modulo A (Esame del 5/6/2015)

Esercizio 1

Considerate la seguente gerarchia di classi:

```
class A {
  public static void f() { System.out.println("A.f()"); }
  public static void g() { f(); }
  public void h() { System.out.println("A.h()"); }
  public void i() { f(); }
}

class B extends A {
  public static void f() { System.out.println("B.f()"); }
  public static void g() { f(); }
  public void h() { System.out.println("B.h()"); }
  public void i() { f(); }
}
```

siano inoltre date le seguenti definizioni:

```
A aa = new A();
A ab = new B();
B bb = new B();
```

Quale è il risultato della compilazione e della (eventuale, nel caso la compilazione non dia errori) esecuzione dei seguenti frammenti?

1. `aa.g();`
2. `aa.h();`
3. `aa.i();`
4. `((B)ab).f();`
5. `ab.g();`
6. `ab.h();`
7. `ab.i();`
8. `bb.g();`
9. `bb.h();`
10. `bb.i();`

Soluzione Esercizio 1

Innanzitutto è da notare che a differenza di altri linguaggi Java permette di invocare un metodo `static` sia con la notazione `classe.metodo()` che con quella `oggetto.metodo()`. In entrambi i casi, però la risoluzione è statica e non esiste overriding. I metodi `B.f()` e `B.g()` fanno, pertanto, shadowing di quelli in `A` e non rappresentano degli override.

1. `aa.g()`: Tipo statico `A`, chiama staticamente `A.g()` che a sua volta chiama `A.f()` . Risultato: `A.f()`.
2. `aa.h()`: Tipo statico `A`, best match `void A.h()`, tipo dinamico `A`, risoluzione dinamica `void A.h()`. Risulato: `A.h()`.
3. `aa.i()`: Tipo statico `A`, best match `void A.i()`, tipo dinamico `A`, risoluzione dinamica `void A.i()` che chiama `A.f()`. Risultato: `A.f()`.
4. `((B)ab).f()`: Tipo statico `A`, dinamico `B` quindi cast valido. dopo il cast tipo statico `B`, chiama staticamente `B.f()` . Risultato: `B.f()`.
5. `ab.g()`: Tipo statico `A`, indipendentemente dal tipo dinamicao chiama staticamente `A.g()` che a sua volta chiama `A.f()` . Risultato: `A.f()`.

6. `ab.h()`: Tipo statico `A`, best match `void A.h()`, tipo dinamico `B`, risoluzione dinamica `void B.h()`. Risultato: `B.h()`.
7. `ab.i()`: Tipo statico `A`, best match `void A.i()`, tipo dinamico `B`, risoluzione dinamica `void B.i()` che chiama `B.f()`. Risultato: `B.f()`.
8. `bb.g()`: Tipo statico `B`, chiama staticamente `B.g()` che a sua volta chiama `B.f()` . Risultato: `B.f()`.
9. `bb.h()`: Tipo statico `B`, best match `void B.h()`, tipo dinamico `B`, risoluzione dinamica `void B.h()`. Risultato: `B.h()`.
10. `bb.i()`: Tipo statico `B`, best match `void B.i()`, tipo dinamico `B`, risoluzione dinamica `void B.i()` che chiama `B.f()`. Risultato: `B.f()`.

Esercizio 2

Implementate una classe `Razionale` che rappresenta un numero razionale come rapporto a/b di due interi a e b. Usate `int` per il numeratore e denominatore. La classe deve includere

- un costruttore che prende numeratore e denominatore come input;
- due metodi statici `plus` e `times` che implementano addizione e moltiplicazione tra due `Razionale` passati come argomento;
- due metodi membro `add` e `multiply` che aggiungono e moltiplicano l'oggetto per il `Razionale` argomento;
- la funzione `equals` per verificare che due `Razionale` rappresentino lo stesso numero.

Suggerimenti:
$a/b + c/d = (ad + bc)/(bd)$
$a/b * c/d = (ac)/(bd)$
$a/b == c/d \Leftrightarrow ad == bc$.

Soluzione Esercizio 2

```
class Razionale
{
  private int numeratore, denominatore;

  public Razionale(int numeratore, int denominatore)
  {
    this.numeratore = numeratore;
    this.denominatore = denominatore;
  }

  public Razionale add(Razionale r)
  {
    numeratore = numeratore*r.denominatore  + denominatore*r.numeratore;
    denominatore = denominatore * r.denominatore;
    return this;
  }

  public static Razionale plus(Razionale l, Razionale r)
  {
    Razionale result=new Razionale(l.numeratore, l.denominatore);
    return result.add(r);
  }

  public Razionale multiply(Razionale r)
  {
    numeratore = numeratore*r.numeratore;
    denominatore = denominatore * r.denominatore;
    return this;
  }
```

```
  public static Razionale times(Razionale l, Razionale r)
  {
    Razionale result=new Razionale(l.numeratore, l.denominatore);
    return result.multiply(r);
  }

  public boolean equals(Object o)
  {
    if (o==null || !(o instanceof Razionale )) return false;
    else return equals((Razionale)o);
  }

  public boolean equals(Razionale r)
  {
  if (r==null) return false;
  else return numeratore*r.denominatore == denominatore*r.numeratore;
  }
}
```

Esercizio 3

Implementare il metodo `reverseIterator` che prenda in input un `Iterator<T>` e restituisca un iteratore che iteri gli elementi dell'iteratore in ingresso in ordine inverso.
Ricordiamo qui di seguito la specifica dell'interfaccia `Iterator<T>`.

```
interface Iterator<E>
{
  /**
  * Restituisce true se l'iteratore ha ancora elementi disponibili
  */
  boolean hasNext();

  /**
  * Restituisce il successivo elemento nell'iterazione.
  * Lancia NoSuchElementException se non ci sono elementi disponibili
  */
  E next();

  /**
  * Rimuove dalla collezione l'ultimo elemento ritornato dall'iteratore (operazione opzionale).
  * Se l'operazione non e' supportata lancia UnsupportedOperationException.
  * Se l'operazione e' supportata ma next() non e' ancora stato chiamato, lancia IllegalStateException.
  */
  void remove();
}
```

Nota: **non** è obbligatorio implementare un nuovo iteratore.

Soluzione Esercizio 3

In questo caso l'unico modo per ottenere l'effetto voluto è quello di invertire il contenuto in una collezione, infatti un iteratore non può essere copiato, resettato o muoversi all'indietro, risulta quindi impossibile creare un iteratore che inverta al volo senza una collezione di supporto.

```
  public static Iterator<T> filter(Iterator<T> iter) {
    ArrayList<T> reversed=new ArrayList<T>();
    while(iter.hasNext()) {
      reversed.add(0,iter.next());
      }
    return reversed.iterator();
    }
```

Esercizio 4

Il metodo di Newton-Raphson e' un algortimo per risolvere numericamente l'equazione $f(x) = 0$, e consiste nell'iterare la ricorrenza

$$x^{(t+1)} = x^{(t)} - \frac{f(x^{(t)})}{f'(x^{(t)})}$$

fino a che la differenza (in valore assoluto) tra $x^{(t)}$ e $x^{(t+1)}$ è minore di una tolleranza data.
Sia data la seguente interfaccia che astrae il comportamento di una funzione derivabile

```
interface Function
{
  // calcola il valore della funzione in x, i.e.,  f(x)
  double value(double x);

  //calcola il valore della derivata della funzione in x, i.e., f'(x)
  double derivative(double x);
}
```

si implementi il seguente metodo statico della classe `MyApp`

```
//restituisce la soluzione di f(x)=0 partendo da x0 e fermandosi quando iterazioni sucessive
// sono a distanza inferiore a toll
double NewtonRaphson(Function f, double x0, double toll)
```

Implementate inoltre le seguenti classi che implementano `Function` e che costituiscono elementi per la costituzione di funzioni complesse:

- `Constant`: questa classe mantinene un `double` e rappresenta una funzione costante: `value` su questa calsse deve restituire la costante immagazzinata;
- `Variable`: questa classe corrisponde alla x: `value` restituisce il suo input;
- `Plus`: questa classe mantiene due `Function` e ne rappresenta la somma;
- `Times`: questa classe mantiene due `Function` e ne rappresenta il prodotto. Si ricorda che $\frac{d}{dx}(f(x)g(x)) = f'(x)g(x) + f(x)g'(x)$.

Si scriva, infine, la funzione `main` della classe `MyApp` che generi un oggetto che rappresenti la funzione $2x^2 - 3$, ne calcoli uno zero (risolva $2x^2 - 3 = 0$) e lo stampi a video.

Soluzione Esercizio 4

```
interface Function
{
  // calcola il valore della funzione in x, i.e.,  f(x)
  double value(double x);

  //calcola il valore della derivata della funzione in x, i.e., f'(x)
  double derivative(double x);
}

class Constant implements Function
{
  private double val;

  public Constant(double val) { this.val=val; }

  public double value(double x) { return val; }

  public double derivative(double x) { return 0.0; }
 }

class Variable implements Function
{
  public double value(double x) { return x; }
```

```
  public double derivative(double x) { return 1.0; }
 }

class Plus implements Function
{
  private Function left, right;

  public Plus(Function left, Function right)
  {
    this.left=left;
    this.right=right;
  }

  public double value(double x) { return left.value(x) + right.value(x); }

  public double derivative(double x) { return left.derivative(x) + right.derivative(x); }
}

class Times implements Function
{
  private Function left, right;

  public Times(Function left, Function right)
  {
    this.left=left;
    this.right=right;
  }

  public double value(double x) { return left.value(x) * right.value(x); }

  public double derivative(double x)
  { return left.derivative(x)*right.value(x) + left.value(x)*right.derivative(x); }
}

public class MyApp
{
  public static double NewtonRaphson(Function f, double x0, double toll)
  {
    double x1 = x0 - f.value(x0)/f.derivative(x0);

    while (Math.abs(x1-x0)>toll)
    {
      x0 = x1;
      x1 = x0 - f.value(x0)/f.derivative(x0);
    }

    return x1;
  }

  public static void main(String[] args)
  {
    Function fx2 = new Times(new Variable(), new Variable()); // x^2
    Function f2x2 = new Times( new Constant(2), fx2); //2*x^2
    Function f= new Plus(f2x2, new Constant(-3)); // 2*x^2-3

    double x = NewtonRaphson(f, 1.0, 1e-3);

    System.out.println("la funzione si annulla per x=" + x);
  }
}
```

Esercizio 5

Sia data la classica gerarchia di forme:

```
interface Shape
{
  void draw(Raster o);
}

class Square implements Shape {...}
class Circle implements Shape {...}
class Triangle implements Shape {...}
class Rectangle implements Shape {...}
class Ellipse implements Shape {...}
...
```

dove il metodo `draw` disegna l'interno della forma (forma "piena").
L'interfaccia `Raster` fornisce un modo semplificato per disegnare linee orizzontali

```
interface Raster
{
// disegna una linea orizzontale sulla riga "row" a partire dalla colonna "startColumn"
// fino alla colonna "endColumn"
void drawSegment(int row, int startColumn, int endColumn);
}
```

Date queste classi base che si considerano non modificabili, siete stati commissionati per scrivere del codice che disegni le forme geometriche con un tratteggio a 45^o (dall'alto a destra, verso il basso a sinistra) con pieni e vuoti della stessa dimensione (vedere Figura 1).

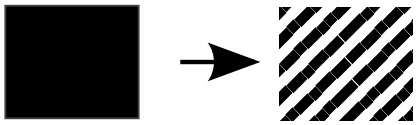

Figura 1: Rettangolo pieno e rettangolo tratteggiato.

Si commenti la soluzione adottata, con particolare attenzione alla scalabilità, alla duplicazione e alla possibilità di riutilizzo dell'approccio a nuove estensioni.