Programmazione ad Oggetti – Modulo A (Esame del 9/1/2015)

Esercizio 1

Considerate la seguente gerarchia di classi:

```
class A {
public void m( int x ) { System.out.println("A.m:" + x); }
public void m( double x ) { m((int) x); }
}

public class B extends A {
public void m( int x ) { System.out.println("B.m:int:" + x); }
public void m( double d ) { System.out.println("B.m:dbl:" + d); }
}
```

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.m(1);`
2. `((B)aa).m(2.0);`
3. `((A)bb).m(3.0);`
4. `bb.m(4.0);`
5. `bb.m(5);`
6. `ab.m((double)6);`
7. `((B)ab).m(7.0);`
8. `ab.m(8.0);`
9. `((B)ab).m(9);`
10. `((A)ab).m(10.0);`

Soluzione Esercizio 1

1. Tipo statico `A`, tipo dinamico: `A`, best match: `m(int)`, funzione eseguita: `A.m(int)`, risultato: `A.m:1`
2. Tipo statico `A`, tipo dinamico: `A`, best match: `m(double)`, cast exception a runtime (`A` non è sottotipo di `B`)
3. Tipo statico `B`, tipo dinamico: `B`, best match: `m(double)`, cast a buon fine (`B` è sottotipo di `A`), funzione eseguita: `B.m(bouble)`, risultato: `B.m:dbl:3.0`
4. Tipo statico `B`, tipo dinamico: `B`, best match: `m(double)`, funzione eseguita: `B.m(double)`, risultato: `B.m:dbl:4.0`
5. Tipo statico `B`, tipo dinamico: `B`, best match: `m(int)`, funzione eseguita: `B.m(int)`, risultato: `B.m:int:5`
6. Tipo statico `A`, tipo dinamico: `B`, best match: `m(double)`, funzione eseguita: `B.m(double)`, risultato: `B.m:dbl:6.0`
7. Tipo statico `A`, tipo dinamico: `B`, best match: `m(double)`, cast a buon fine (`B` è sottotipo di `B`), funzione eseguita: `B.m(double)`, risultato: `B.m:dbl:7.0`
8. Tipo statico `A`, tipo dinamico: `B`, best match: `m(double)`, funzione eseguita: `B.m(double)`, risultato: `B.m:dbl:8.0`
9. Tipo statico `A`, tipo dinamico: `B`, best match: `m(int)`, cast a buon fine (`B` è sottotipo di `B`), funzione eseguita: `B.m(int)`, risultato: `B.m:int:9`
10. Tipo statico `A`, tipo dinamico: `B`, best match: `m(double)`, cast a buon fine (`B` è sottotipo di `A`), funzione eseguita: `B.m(double)`, risultato: `B.m:dbl:10.0`

Esercizio 2

Considerate la seguente rappresentazione degli insiemi

```
class Set<T>
{
private ArrayList<T> contents = new ArrayList<T>();

public Iterator<T> iterator(){ return contents.iterator(); }

public void add(T val)
{ if (!contents.contains(val)) contents.add(val); }

public void remove(T val){ contents.remove(val); }
}
```

Completate la definizione della classe `MinSet<T>`

```
class MinSet<T extends Comparable<T>> extends Set<T>
{
private T min; // minimo dell'insieme
// costruisce un MinSet inizializzando opportunamente min
public MinSet()
{ ...}

// aggiunge il nuovo valore a this, aggiornando il campo
// min nel caso il nuovo valore sia il nuovo minimo
public void add(T val)
{ ... }

// rimuove una occorrenza di val da this, aggiornando il
// min in tutti i casi in cui sia necessario
public void remove(T val)
{ ... }
}
```

Ricordiamo qui di seguito la specifica dellinterfaccia `Comparable<T>`.

```
interface Comparable<T>
{
public int compareTo(T t)
// confronta this con t restituendo un intero negativo, zero, o un intero
// positivo se, rispettivamente, this e minore, uguale o maggiode di t
}
```

Soluzione Esercizio 2

```
class MinSet<T extends Comparable<T>> extends Set<T>
{
  private T min; // minimo dell'insieme

  // costruisce un MinSet inizializzando opportunamente min
  public MinSet()
  { }

  // aggiunge il nuovo valore a this, aggiornando il campo
  // min nel caso il nuovo valore sia il nuovo minimo
  public void add(T val)
  {
    if (val==Null) return;
    super.add(val);
    if (min==Null || min.compareTo(val)>0)
      min=val;
  }

  // rimuove una occorrenza di val da this, aggiornando il
  // min in tutti i casi in cui sia necessario
  public void remove(T val)
  {
    super.remove(val);
    if( min.compareTo(val)==0 )
    {
      Iterator<T> iter=iterator();
      if (iter.hasNext())
min=iter.next();
      else min=Null;
      while(iter.hasNext())
      {
T tmp=iter.next();
if (min.compareTo(val)>0)
  min=val;
      }
    }
  }
}
```

Esercizio 3

Sia data la seguente interfaccia

```
interface Predicate<T>
{ boolean evaluate(T e); }
```

Implementare il metodo `filter` che prenda in input un `Predicate<T>` e un `Iterator` su di un sottotipo di `T` e restituisca un iteratore che iteri sui soli elementi dell'iteratore in ingresso che soddisfino al predicato (`evaluate` su quegli elementi dia `true`).
Ricordiamo qui di seguito la specifica dellinterfaccia `Iterator<T>`.

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

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

  /**
  * Rimuove dalla collezione l'ultimo elemento ritornato dall'iteartore (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

La soluzione più rapida, senza implementare un nuovo iteratore è:

```
  public static <T, S extends T> Iterator<S> filter(Predicate<T> pred, Iterator<S> iter) {
    ArrayList<S> filtered=new ArrayList<S>();
    while(iter.hasNext()) {
      S current=iter.next();
      if (pred.evaluate(current)) filtered.add(current);
      }
    return filtered.iterator();
    }
```

Nel caso si voglia implementare un nuovo iteratore:

```
class filteredIterator<T,S extends T> implements Iterator<S> {
  private Iterator<S> base_iterator;
  private Predicate<T> predicate;
  S current;

  public filteredIterator(Predicate<T> pred, Iterator<S> iter) {
    predicate=pred;
    base_iterator=iter;
    getCurrent();
    }

    private void getCurrent() {
    current=null;
    while (current==null && base_iterator.hasNext()) {
      S val=base_iterator.next();
      if (predicate.evaluate(val)) current=val;
      }
    }

  public boolean hasNext() { return current!=null; }

  public S next() {
    S result=current;
    getCurrent();
    return result;
    }

  public void remove() { throw new UnsupportedOperationException(); }
  }
```

e il metodo `filter` diventa semplicemente:

```
  public static <T, S extends T> Iterator<S> filter(Predicate<T> pred, Iterator<S> iter) {
    return new filteredIterator<T,S>(pred,iter);
    }
```

Esercizio 4

Considerate le seguenti classi che descrivono le monete della valuta americana. In ciascuna classe, il metodo `value()` restituisce il valore della moneta.

```
class Penny { public double value(){ return 0.01; } }

class Nickel { public double value(){ return 0.05; } }

class Dime { public double value(){ return 0.1; } }

class Quarter { public double value(){ return 0.25; } }
```

Definite un nuovo tipo `Coin` e modificate le classi precedenti in modo da completare il codice della classe `PiggyBank` (salvadanaio) descritto qui di seguito.

```
public class PiggyBank
{
// Inizialmente il salvadanaio e vuoto
public PiggyBank()
{ ... }

// Aggiunge al salvadanaio il coin c
public void save(Coin c)
{ ... }

// Estrae dal salvadanaio un coin dello stesso valore di c
// restituisce la moneta estratta o null se non ci sono
// monete dello stesso valore
public Coin extract(Coin c)

// Restituisce tutto il contenuto del salvadanaio
public Iterator<Coin> breakPiggyBank()
{ ... }

// Restituisce il valore totale del contenuto espresso in centesimi
public double total()
{ ... }

...
}
```

Soluzione Esercizio 4

```
  interface Coin { double value(); }

  class Penny implements Coin { public double value(){ return 0.01; } }

  class Nickel implements Coin { public double value(){ return 0.05; } }

  class Dime implements Coin { public double value(){ return 0.1; } }

  class Quarter implements Coin { public double value(){ return 0.25; } }


  public class PiggyBank {
    private ArrayList<Coin> coins;

    // Inizialmente il salvadanaio e vuoto
    public PiggyBank() {
      coins = new ArrayList<Coin>();
      }

    // Aggiunge al salvadanaio il coin c
    public void save(Coin c) {
      coins.add(c);
      }

    // Estrae dal salvadanaio un coin dello stesso valore di c
    // restituisce la moneta estratta o null se non ci sono
    // monete dello stesso valore
    public Coin extract(Coin c) {
      for (Coin e:coins) {
if (e.value()==c.value()){
  coins.remove(e);
  return e;
  }
}
      return null;
      }

    // Restituisce tutto il contenuto del salvadanaio
    public Iterator<Coin> breakPiggyBank() {
      Iterator<Coin> iter=coins.iterator();
      coins=new ArrayList<Coin>();
      return iter;
      }

    // Restituisce il valore totale del contenuto espresso in centesimi
    public double total() {
      double tot=0;
      for (Coin c:coins) tot += c.value();
      return tot;
      }

  }
```

Esercizio 5

State scrivendo un programma gestionale e dovete gestire l'anagrafe delle persone legali con cui la ditta ha rapporti. Una *persona legale* è idetificata da una *partita iva* (una stringa) e nei confronti della ditta possono essere *clienti* o *fornitori*.
Per i clienti possiamo ottenere lo storico delle vendite (nella forma di un `Iterator<Vendite>`), mentre dai fornitori possiamo ordinare nuovi prodotti (`void ordina(Prodotto p, int quantita)`).
La stessa persona legale può essere sia cliente che fornitore e può essere registrato nel sistema con un ruolo, mentre il secondo ruolo può essere aggiunto sucessivamente. Cioè una persona legale può essere registrata nel sistema come cliente e sucessivamente diventare anche fornitore, o essere registrato come fornitore e sucessivamente diventare cliente. Una volta che una persona legale ha assunto un ruolo lo mantiene per sempre.
Scrivete il codice Java per implementare le entià ed il loro comportamento (lasciando in bianco l'implementazione dei metodi). Motivate le scelte nella strutturazione delle classi ed interfacce utilizzate.