Metodologie di Programmazione
Terzo Appello

docenti: Ivano Salvo, Flavio Chierichetti
Sapienza Università di Roma, 19 settembre 2014

Parte Prima

Le soluzioni corrette a questa parte garantiscono la sufficienza.

Esercizio 1.1: Subtyping e Overriding Considerate le seguenti definizioni di classi e interfacce:

```
interface I {
   void m(J x);
   }
interface J {
   void n();
   }

class A implements I {
  public void m(J x) {
     System.out.println("A.m()");
     }
  }

class B extends A {
   public void m(C x) {
      System.out.println("B.m()");
      }
   }

class C extends A implements J {
   public void m(C x) {
      System.out.println("C.m()");
      x.n();
      }
   public void n() {
      System.out.println("C.n()");
      }
   }
```

Scrivete l'output prodotto dal `main`, nei punti `p1`...`p5`. Motivate **brevemente** le risposte. Nell'evenutualità che una riga produca un errore in compilazione o in esecuzione scrivete il perchè e valutate le rimanenti righe immaginando che la riga che causa l'errore sia rimossa.

```
public class TerzoAppello{
   public static void main(String[] args){
      /*p1*/ I x1 = new C(); x1.m(x1);
      /*p2*/ I x2 = new C(); x2.m((J) x2);
      /*p3*/ I x3 = new A(); x3.m((C) x3);
      /*p4*/ I x4 = new C(); ((C)x4).m((C) x4);
      /*p5*/ B x5 = new B(); x5.m(new C());
 }
```

Esercizio 1.2: Programmazione Procedurale. Due vettori a e b (di lunghezza rispettivamente n ed m) sono *simili* se contengono gli stessi elementi, eventualmente con diversi numeri di occorrenze. Per chi ama la logica, se:

$$\forall i : 0 \leq i < n.\ \exists j :\ 0 \leq j < m . a[i] = b[j]$$
$$\wedge\ \forall i : 0 \leq i < m.\ \exists j :\ 0 \leq j < n . b[i] = a[j]$$

Prendendo ad esempio array di interi, l'array {1, 1, 3} è simile all'array {1, 3, 1}, all'array {3,1}, ed anche all'array {3, 3, 3, 1}, ma non all'array {1, 2, 3}. Scrivere un metodo statico JAVA

```
boolean simili(Object[] a, Object[] b)
```

che presi in input due vettori restituisce `true` se sono simili.

Corredare il metodo di opportune precondizioni, postcondizioni e invarianti di eventuali cicli.

Esercizio 1.3: Programmazione sui Generics. Sia data l'interfaccia:

```
interface Predicato<A>{
    boolean soddisfa(A a);
    }
```

Supponete di arricchire l'implementazione delle liste generiche viste a lezione con un metodo:

```
List<A> filtra(Predicato p)
```

che restituisce una nuova lista che contiene tutti gli oggetti che soddisfano il predicato `p`.

Parte Seconda

Esercizio 2.1: Programmazione sui Generics. Supponete di arricchire l'implementazione delle liste generiche viste a lezione con un metodo:

```
Coppia<List<A>,List<B>> dividi()
```

che restituisce una coppia di liste, in cui la prima contiene tutti gli elementi della lista ricevente che si trovano in posizione dispari e la seconda contiene tutti gli elementi che si trovano in posizione pari.

Dare una implementazione ricorsiva del metodo scrivendone il codice per le liste vuote e le liste non vuote.

Esercizo 2.2: Progettazione di Classi Generiche. Supponendo di avere a disposizione un esecutore JAVA che abbia gli stack come *unico* tipo di dato (con le usuali operazioni `A top()`, `void pop()`, `void push(A a)` e `boolean isEmpty()`, definire una classe che implementa la struttura dati coda, con le usuali operazioni `void enqueue(A a)` e `A dequeue()`. [SUGG: usare 2 stack per simulare la coda, usandoli opportunamente].