Programmazione a Oggetti
Metodologie di Programmazione
8 Giugno 2012

Nome: _____ Matricola: _____

Metodologia di Programmazione [ ]
Programmazione a Oggetti [ ]

Istruzioni

- Scrivete il vostro nome sul primo foglio.
- Indicare se la vostra prova è riferita all'esame MP o PO
- Scrivete le soluzioni nello spazio riservato a ciascun esercizio.
- **Gli esercizi 9 e 10 sono rivolti a coloro che non hanno svolto le esercitazioni, ovvero le hanno svolte ottenendo una valutazione insufficiente / insoddisfacente. Per chiunque scelga di svolgerli, il punteggio su questi esercizi cancella il punteggio ottenuto nelle esercitazioni.**

LASCIATE IN BIANCO:

1	2	3	4	5	6	7	8	9	10	TOT

1 Esercizio

Descrivere a cosa serve e come funziona il design patter Observer (o Listener). Completate l'esercizio con il codice di una classe di esempio che implementa il ruolo dell'Observer. [3pt]

2 Esercizio

1. In quale situazione e' necessario definire un costruttore *private*? In quale situazione può essere utile definire un costruttore *protected*? Per entrambe le situazioni date un esempio. [2pt]

2. Descrivete le differenze tra metodi `static` e non `static` in Java. Date una versione corretta della seguente definizione di classe:

```
class StaticTest
{
      private int a;
      static private int b;

      public int m1(int c) { return c * (a + b); }
      public static int m2(int d) { return d * (a - b); }
}
```

[2pt]

3 Esercizio

Sia data la seguente gerarchia:

```
    interface M { M m(); }
    interface K { void k(); }

    class A implements M, K {
       public M m() { System.out.print(" A "); return this; }
       public void k() {}
    }
    class B extends A {
       public M m() { System.out.print(" B "); return super.m(); }
}
```

Indicate il tipo statico ed il tipo dinamico per ciascuna (sotto) espressione nei seguenti frammenti di codice. Determinate inoltre il risultato della compilazione e, nel caso la compilazione non dia errori, dell'esecuzione. [3pt]

A. `M a = new B(); B b = ((B)a).m();`

B. `M a = new B(); K b = (K)(a.m());`

C. `M a = new A(); B b = (B)(a.m());`

4 Esercizio

Date le seguenti classi:

```
public class A {
}

public class B {
   public boolean equals(Object o){
      if (o==null || !( o instanceof  B))
        return false;
      return true;
   }
   public int hashCode () {return 0;}
}
```

Scrivere a fianco di ogni istruzione di stampa cosa viene stampato. Nel caso un' istruzione lanci un'eccezione, si assuma che l'esecuzione prosegua alla riga successiva. [3pt]

```
   Set<Object> set = new HashSet<Object>();
   A a = new A();
   set.add(a);
   System.out.println(set.size());   //.........
   set.add(new A());
   System.out.println(set.size());   //.........
   A a2 = a;
   set.add(a2);
   System.out.println(set.size());   //.........
   B b = new B();
   set.add(b);
   System.out.println(set.size());   //.........
   set.add(new B());
   System.out.println(set.size());   //.........
   B b2 = b;
   set.add(b2);
   System.out.println(set.size());   //.........
```

5 Esercizio

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

```
class Penny   { public int value(){ return 1;  } }

class Nickel  { public int value(){ return 5;  } }

class Dime    { public int value(){ return 10; } }

class Quarter { public int value(){ return 25; } }
```

Definite un nuovo tipo `Coin` che vi permetta di completate il codice della classe `PiggyBank` (salvadanaio) descritto qui di seguito, utilizzando la classe che ritenete più opportuna del framework collection di Java. [4pt]

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

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

     }

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

     }

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

     private ............... contents =  ................;
}
```

6 Esercizio

Considerate la classe seguente:

```
class Position
{
     private double x;
     private double y;
     public Position(double x, double y) {this.x=x; this.y=y;}
     public double getX() {return x;}
     public double getY() {return y;}
}
```

Che cosa sbagliato nella codice seguente? [1pt]

```
class Position3D extend Position
{
    private double z;
    public Position3D(double x, double y, double z)
    {
       this.x = x; this.y = y; this.z = z;
    }
    public double getZ() { return z;}
}
```

Correggete il codice della classe `Position` lasciando intatto il codice di `Position3D`. [1pt]

Correggete il codice della classe `Position3D` lasciando intatto il codice di `Position` [2pt]

7 Esercizio

Considerate la seguente rappresentazione degli insiemi.

```
class Set<T>
{
     private ArrayList<T> contents;

     public Set() { this.contents = new  ArrayList<T>(); }

     public Iterator<T> getContents(){ 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 `MaxSet<T>` seguendo le indicazioni date nei commenti. [4pt]

```
class MaxSet<T extends Comparable<T>> extends Set<T>
{
     private T max;  // massimo dell'insieme
     // costruisce un MaxSet inizializzando opportunamente max
     public MaxSet()
     {
       ..................
     }
     // aggiunge il nuovo valore a this, aggiornando il campo
     // max nel caso il nuovo valore sia il nuovo massimo
     public void add(T val)
     {
       ...................
       ...................
     }
     // rimuove una occorrenza di val da this, aggiornando il
     // max in tutti i casi in cui sia necessario
     public void remove(T val)
     {
       ...................
       ...................
       ...................
       ...................
       ...................
     }
}
```

Ricordiamo qui di seguito la specifica dell'interfaccia `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
}
```

8 Esercizio

Siano date le seguenti classi per la gestione di un file system.

```
class File
{
   private String contents;
   private String name, owner;
   private int size;

   public File(String name, String owner)
   {
     this.name = name; this.owner = owner;
   }

   public int size()     { contents.length * 2; }
   public String name()  { return name; }
   public String owner() { return owner; }
}

class Directory
{
   private List<File> contents;
   private String name, owner;

   public Directory(String name, String owner)
   {
     this.name = name; this.owner = owner;
     contents = new LinkedList();
   }
   public String name()  { return name; }
   public String owner() { return owner; }
   public size()
   {
     // restituisce le somme delle dimensioni di ciascun
     // file contenuto nella directory
   }
   public void add(File f) { contents.add(f); }
}
```

Dovete modificare la struttura delle classi in modo che una directory possa contenere non solo file ma anche directory, che a loro volta conterranno file e directory. In particolare:

A. Definite una nuova classe astratta `Resource` per rappresentare file e directory. [1pt]

B. Ridefinite le classi `File` e `Directory` come sottoclassi di `Resource`. La nuova classe `File` mantiene le funzionalità della classe originale. La nuova classe `Directory` dovrà invece gestire una collezione di oggetti `Resource` invece che `File`. Inoltre, il metodo `size()` dovrà restituire la dimensione totale dei file (solo dei file) contenuti nella directory corrente e, ricorsivamente, nelle eventuali directory contenute nella directory corrente. [4pt]

9 Esercizio

Siano date le seguenti definizione di classe ed interfaccia.

```
interface Measurable
{ /** oggetti misurabili */
  public double measure() ;
}

class Queue<T>
{
  /** costruisce una coda vuota */
  public Queue()

  /** aggiunge elem sulla coda. @pre: elem != null   */
  public void enqueue(T elem)

  /** rimuove la testa della cosa e la restituisce */
  T dequeue()

  /** true se la coda e' vuota */
  public boolean empty()

  /** il numero degli elementi in coda */
  public int size()

}
```

Vogliamo definire una nuova classe `BoundedMeasureQueue`, che si comporta come una coda, ma: (i) può contenere solo elementi di tipo `Measure`, e (ii) il totale delle misure degli elementi al suo interno non può superare un valore `double` dato. Nel caso in cui l'aggiunta di un elemento in doda faccia superare la soglia, una chiamata ad `enqueue()` non modifica la coda.

Definite la nuova classe `BoundedMeasureQueue` come sottoclasse della classe `Queue` data.

10 Esercizio

Sia data la sequente definizione.

```
public class BlinkPanel extends JPanel
{
    public BlinkPanel()
    {
       btn = new JButton("Click me");
       label = new JLabel("");
       add(btn); add(label);
       btn.addActionListener(new Blinker());
    }

    public JLabel getLabel() { return label; }
    private JButton btn;
    private JLabel label;

    private class Blinker implements ActionListener
    {
        public void actionPerformed(ActionEvent e)
        {
            if (label.getBackground() == Color.WHITE)
                label.setBackground(Color.BLACK);
            else
                label.setBackground(Color.WHITE);
        }
        private Color col = Color.WHITE;
    }
}
```

Modificate l'implementazione della classe `BlinkPanel` come richiesto nei due punti qui di seguito. NB: le modifiche richieste sull'implementazione sono distinte, non cumulative e non devono modificare le funzionalità pubbliche della classe `BlinkPanel`.

A. Eliminate la classe interna `Blinker` e ridefinitela come classe esterna.

B. Eliminante la classe interna `Blinker` rendendo la classe `BlinkPanel` un `ActionListener`.