Programmazione ad oggetti – Modulo A
Prova scritta 20 Maggio 2013

Nome: _____ Matricola: _____

Istruzioni

- Scrivete il vostro nome sul primo foglio.
- Scrivete le soluzioni nello spazio riservato a ciascun esercizio.
- La consegna di questi esercizi annulla il risultato conseguito nei compitini.
- No libri, appunti o altro.

LASCIATE IN BIANCO:

1	2	3	4	5	TOT

Esercizio 1

Completate la seguente definizione di classe `MemAddress` che realizza il tipo di dato *indirizzo di memoria*.

```
class MemAddress {
   // OVERVIEW: un indirizzo di memoria ha una base ed un offset,
   // entrambi interi non negativi

   public MemAddress(int b, int o) throws AddressFormatException
   // Costruisce un MemAddress con base b e offset o, se b ed o sono
   // entrambi maggiori o uguali a zero; altrimenti lancia
   // l'eccezione

   public MemAddress max(MemAddress a, MemAddress b)
   // Se a e b hanno la stessa base restituisce l'indirizzo con offset
   // maggiore; altrimenti restituisce null.

   public MemAddress shift(int n) throws AddressFormatException
   // Restituisce un nuovo indirizzo con la stessa base di this
   // e come offset il valore ottenuto sommando n all'offset di this.
   // Non modifica this.
   // N.B. n potrebbe essere negativo

   public int toInt()
   // Restituisce il valore numerico corrispondente all'indirizzo,
   // ottenuto dalla somma base + offset.
```

Implementate inoltre un metodo statico `main()` che usi tutti i metodi della classe `MemAddress` senza mai lanciare eccezioni. In particolare, il corpo del metodo deve:

- creare due indirizzi `a` e `b`,
- confrontarli ed assegnare ad una nuova variabile `c` il massimo dei due,
- costruire un indirizzo `d` ottenuto dallo *shift* di `c` di un valore intero a vostra scelta;
- stampare il valore numerico di `d`.

Nel caso si verifichino eccezioni nel corso di queste operazioni, il main le cattura e termina stampando in output il messaggio: `operazione non ammessa`.

Esercizio 2

Sia data la seguente classe per rappresentare un'area di memoria cache di n posizioni.

```
class Cache {
    private char[] contents;

    public Cache(int n)
    {
      if (n <= 0) throw new ArrayOutOfBoundsException();
      contents = new char[n];
    }

    public void write(int i, char c) throws ArrayIndexOutOfBoundsException
    {
      contents[i] = c;
    }

    public char read(int i) throws ArrayIndexOutOfBoundsException
    {
      return contents[i];
    }
}
```

Definite una sottoclasse `UndoCache` di `Cache` con un metodo `undo()` che permetta di annullare l'effetto delle operazioni di `set` sulle posizioni della cache. Il metodo `undo()` si comporta come l'operazione di *annulla* disponibile in un qualunque editor, ovvero: annulla l'effetto dell'ultima operazione di `set()` che non sia stata già annullata; se non ci sono `set()` da annullare, `undo()` non ha alcun effetto.

Esempio:

```
// OPERAZIONE           // STATO DELLA CACHE (_ = carattere nullo)
b = new UndoCache(4);   // _::_::_::_
b.write(1,'a');         // _::a::_::_
b.read(1,'b');          // _::b::_::_
b.undo();               // _::a::_::_
b.write(3,'c');         // _::a::_::c
b.undo();               // _::a::_::_
b.undo();               // _::_::_::_
b.undo();               // _::_::_::_
```

Esercizio 3

Vogliamo costruire un sistema di classi per gestire le operazioni di un file system. La seguente classe astratta descrive la struttura comune degli elementi del file system.

```
abstract class FSItem {
   private String nome;

   // diritti
   protected boolean rr;
   protected boolean ww;

   protected FSItem(String s) {nome = s; rr = true; ww = true; }

   public String name() { return nome; }
   public void chmod (boolean r, boolean w) { rr = r; ww = w; }
}
```

Un `FSItem` ha un nome, due booleani che definiscono i diritti di lettura e scrittura, ed il metodo `chmod()` per modificarli. Le diverse componenti del filesystem sono descritte dalle classi specificate qui di seguito.

```
class Directory extends FSItem {
  // OVERVIEW: una directory e` un FSItem che puo` contenere
  // un insieme di sotto-directories, files o links

  public string ls() throws IllegalOperation
  // Se this permette la lettura, restituisce una stringa che
  // contiene i nomi di tutti gli FSItems contenuti nella
  // directory;  altrimenti lancia l'eccezione

  public void add(FSItem i)
  // Aggiunge i a this se this permette la scrittura;
  // Altrimenti lancia l'eccezione
}


class File extends FSItem {
  // OVERVIEW: un File e` un FSItem con un contenuto di tipo string

  public void set(string s) throws IllegalOperation
  // Se l'operazione di scrittura e` permessa setta ad s il
  // contenuto del file; altrimenti lancia l'eccezione

  public string get()
  // Se l'operazione di lettura e` permessa restituisce il
  // contenuto del file; altrimenti lancia l'eccezione
}

class Link extends FSItem {
   // OVERVIEW: un Link e` un FSItem con associato un altro FSItem,
   // detto il "target". Implementa tutte le operazioni del target,
   // e ciascuna operazione viene implementata mediante
   // l'invocazione della stessa operazione sul target.
}
```

Fornite l'implementazione completa delle classi `Directory`, `File` e `Link`.

Esercizio 4

Considerate la seguente definizione di class, assumendo che il tipo `Element` contenga un metodo `double val()`.

```
public class DataSet
{
   public void add(Element x)
   {
     if (x == null) return;
     elements.add(x);
     total = total + x.val();
     if (elements.size() == 0 || max.val() < x.val())
        max = x;
   }

   public Element max() { return max; }

   public double average()
   {
     int count = elements.size();
     return (count == 0) ? 0 : sum/count;
   }

   private double sum;
   private Element max;
   private List<Element> elements = new LinkedList<Element>();
}
```

Estendete la classe completando la definizione dei due metodi seguenti, e definendo gli eventuali campi necessari per la loro implementazione.

```
/**
 * @pre true
 * @post @nochange
 * @result = l'ampiezza dell'intervallo dei valori degli elementi
 *  nel dataset, ovvero la differenza tra i valori massimo e minimo
 */
public double range()

/**
 * @pre d >= 0
 * @post @nochange
 * @result = un iteratore che permette di ottenere in sequenza gli
 *  elementi la cui misura e' minore del valore d
 */
public Iterator<Element> lessThan(double d)
```

Esercizio 5

Considerate la seguente gerarchia di tipi:

```
interface M { M m(); }
interface N { }

class A implements M {
   public M m() { return new B(); }
}
class B extends A {
   public M m() { return new A(); }
}
class C extends A implements N {
   public M m() { return this; }
}
```

Indicate il tipo statico ed il tipo dinamico per ciascuna (sotto) espressione nei seguenti frammenti di codice.

Determinate il risultato della compilazione e, nel caso la compilazione non dia errori, dell'esecuzione.

1. `N x = new C(); M y = x.m();`

2. `M x = new A(); B y = (B)x.m();`

3. `M x = new B(); B y = (B)x.m();`