Programmazione ad oggetti – Modulo A
Prova scritta 3 Giugno 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

1 Esercizio

Considerate il seguente sistema di classi per rappresentare un orologio digitale.

```
public class DigitalWatch
{
   private WatchState state = new TimeState();

   public void pressButton() { state.pressButton(this);  }

   public String toString() {  return state.toString(this);  }

   void setState(WatchState state) { this.state = state;   }
}

abstract class WatchState
{
   abstract void pressButton(DigitalWatch context);

   abstract String toString(DigitalWatch context);
}
```

Definite due sottoclassi di `WatchState`, `TimeState`, `AlarmState` che realizzano i due possibili stati di un orologio. In particolare, dato `w:DigitalWatch`:

quando `w` si trova nello stato `TimeState`:

- `w.toString()` restituisce ``tic tac'';
- `w.pressButton()` porta `w` nello stato `AlarmState`;

quando `w` si trova nello stato `AlarmState`:

- `w.toString()` restituisce ``sveglia!!'';
- `w.pressButton()` porta `w` nello stato `TimeState`.

2 Esercizio

Siano date le seguenti definizioni. Completate l'implementazione della classe `Payroll` definindo il codice del metodo `aboveAverageIterator()` e della classe `AboveAverageIterator` in modo che l'iteratore restituito da `aboveAverageIterator()` enumeri i dipendenti (`Employee`) che hanno uno stipendio (`salary`) superiore alla media degli stipendi di tutti gli impiegati nel libro paga (`Payroll`).

```
class Employee
{
  private String name;
  private double salary;
  public Employee(String initName, double initSalary)
  {
    name = initName; salary = initSalary;
  }

  public String toString() { return name + ", \$" + salary; }
}

class Payroll
{
  public List<Employee> employees = new ArrayList<Employee>();

  public Iterator<Employee> aboveAverageIterator()
  {
     // COMPLETARE



  }

  class AboveAverageIterator implements Iterator<Employee>
  {
     // COMPLETARE









  }
```

3 Esercizio

Sia data la seguente specifica della classe `CircList<T>` che realizza il tipo di dato *Lista Circolare*, ovvero una lista in cui il successore dell'ultimo elemento è nuovamente il primo elemento della lista).

```
class CircList<T extends Comparable<T>> {
// OVERVIEW: una CircList e` una lista circolare di Integers.
// Elemento tipico = [x1,...,xn,x1,...,xn, ....]

public CircList<T>()
// POST: costruisce una CircList vuota

public T first() throws EmptyException
// POST: se this e` vuota solleva EmptyException, altrimenti
// restituisce il primo elemento di this

public boolean empty()
 // POST: se this e` vuota ritorna true, altrimenti ritorna false.

public void insert (T x) throws NullPointerException
 // POST: se x e` null solleva NullPointerexception, altrimenti
 // modifica this aggiungendo x come primo elemento

public T delete() throws EmptyException
 // POST se this e` vuota solleva EmptyException, altrimenti rimuove il
 // primo elemento di this, e lo restituisce

public void rightRotate() throws EmptyException;
 // EFFECTS: se this e` vuota solleva EmptyException, altrimenti
 // modifica this ruotando la lista a destra. Ovvero,
 // se this = [x1,...,xn,x1,...,xn]
 // this_post = [xn,x1,...,xn-1, xn,x1,...,xn-1]

public void leftRotate() throws EmptyException;
 // EFFECTS: se this e` vuota solleva EmptyException, altrimenti
 // modifica this ruotando la lista a sinistra. Ovvero,
 // se this = [x1,...,xn,x1,...,xn]
 // this_post = [x2,...,xn,x1, x2,...,xn,x1]
}
```

Completate la definizione della classe `CircList`, ovvero:

a) definite la rappresentazione della classe utilizzando una struttura a lista con nodi semplici

b) fornite e l'invariante di rappresentazione

c) definite l'implementazione del costruttore e di tutti i metodi nella specifica

4 Esercizio

Completate seguente definizione del sottotipo `MinCircList` di `CircList`.

```
class MinCircList<T extends Comparable<T>> extends CircList<T> {
 // OVERVIEW: una CircList con un metodo min() che determina
 // il minimo intero nella lista.

 public T min() throws EmptyException
    // POST: restituisce il minimo elemento contenuto nella lista
}
```

La vostra implementazione deve garantire una complessità costante per `min()`, ridefinendo quindi i metodi `insert()` e `delete()` della superclasse.

5 Esercizio

Considerate la seguente gerarchia di classi:

```
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()"); }
    public void n() { System.out.println("C.n()"); }
}
```

Quale è il risultato della compilazione e della (eventuale, nel caso la compilazione non dia errori) esecuzione dei seguenti frammenti? **Motivate le risposte: le risposte non motivate non saranno considerate**

- `I x = new C(); x.m(x);`

- `I x = new C(); x.m((J)x);`

- `I x = new A(); x.m((C)x);`

- `I x = new C(); ((C)x).m((C)x);`

- `B x = new B(); x.m(new C());`