Programmazione ad oggetti – Modulo A
Prova scritta 4 Settembre 2013

Nome: _____ Matricola: _____

Istruzioni

- Scrivete il vostro nome sul primo foglio.
- Scrivete le soluzioni nello spazio riservato a ciascun esercizio.
- 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 semaforo.

```
public class TrafficLights extends Thread
{
   private LightsState state = new GreenState();

   public void change(){ state.changeState(this); }

   public void show(){ System.out.println(state.color()); }

   public void wait(){ sleep(state.length()); }

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

   public static void main(String[] argv)
   {
     TrafficLights tl = new TrafficLights();
     while (true) { tl.show(); tl.wait(); tl.change(); }
   }
}
```

Definite l'interfaccia `LightState` e tre implementazioni della stessa interfaccia – `GreenState`, `RedState` e `YellowState` – che realizzano i 3 possibili stati di un semaforo. In particolare eseguendo il metodo `main` della classe `TrafficLights`:

- quando `tl` entra nello stato `GreenState`, visualizza `''green''` e dopo 30 secondi passa nello stato `YellowState`;
- quando `tl` entra nello stato `YellowState`, visualizza `''yellow''` e dopo 10 secondi passa nello stato `RedState`;
- quando `tl` entra nello stato `RedState`, visualizza `''red''` e dopo 40 secondi passa nello stato `GreenState`;

2 Esercizio

Considerate l'interfaccia la seguente interfaccia che descrive una generica lista di `T`.

```
public interface List<T>
{
    boolean empty();
    List<T> zip(List<T> l);
}
```

Definite il codice di due classi `MTList<T>` (la lista sempre vuota) e `NEList<T>` (una lista sempre non vuota) che implementano l'interfaccia `List<T>` e realizzando i metodi `empty()` e `zip()`. In particolare, `empty()` restituisce `true` se la lista è vuota, `false` altrimenti, mentre `zip()` è definito come segue:

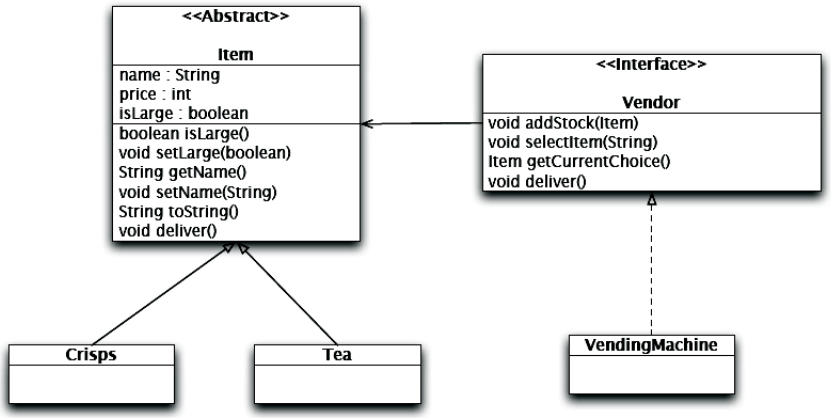
- se `l1:MTList<T>`, oppure `l2:MTList<T>`, allora `l1.zip(l2)` restituisce la lista la vuota;
- altrimenti, `l1.zip(l2)` costruisce la lista il cui i primi due elementi sono rispettivamente il primo elemento di `l1` e il primo elemento di `l2`, e il resto è ottenuto dallo `zip` del resto di `l1` con il resto di `l2`.

Chiariamo il comportamento del metodo `zip()` con due esempi:

- se `l1 = [1,5,3]` e `l2 = [4,2]`, allora `l1.zip(l2) = [1, 4, 5, 2]`.
- se `l1 = []` allora `l1.zip(l2) = []` per qualunque `l2`.

3 Esercizio

Il diagramma qui di seguito rappresenta lo schema di realizzazione di un distributore automatico di vivande o (`Vending Machine`).



Il distributore vende solamente patatine `Crisps` e the `Tea`. Il progetto, peraltro, è studiato in modo da poter essere esteso per vendere un qualunque articolo definito dalla classe astratta `Item`.

Dovete implementare la classe `VendingMachine` e la classe `Tea`, seguendo le direttive descritte qui di seguito:

1. definite i campi necessari per la rappresentazione dei dati e dello stato della `VendingMachine`. In particolare, una `VendingMachine` ha una lista di items in vendita, il suo `stock`. La macchina può avere un item selezionato, che rappresenta la scelta corrente `currentChoice`. Inizialmente la scelta corrente è nulla.
2. implementate i metodi `addStock()` e `selectItem()`. `addStock(Item item)` aggiunge `item` allo stock. `selectItem(String name)` cerca un item con nome `name` nello `stock`; se lo trova, questo diventa la scelta corrente, altrimenti lancia un'eccezione `NoSuchElementException`;
3. implementate il metodo `deliver()`: il metodo controlla che la scelta corrente non sia nulla e nel caso, invoca il metodo `deliver` dell'item corrispondente e resetta la scelta corrente. In caso la scelta corrente sia nulla, il metodo scrive in output la stringa `"please select an item first"`.
4. Implementate la classe `Tea`: il metodo deliver di questa classe restituisce la stringa `"delivering a large cup of tea"` o `"delivering a small cup of tea"` a seconda del valore restituito da `isLarge()`.

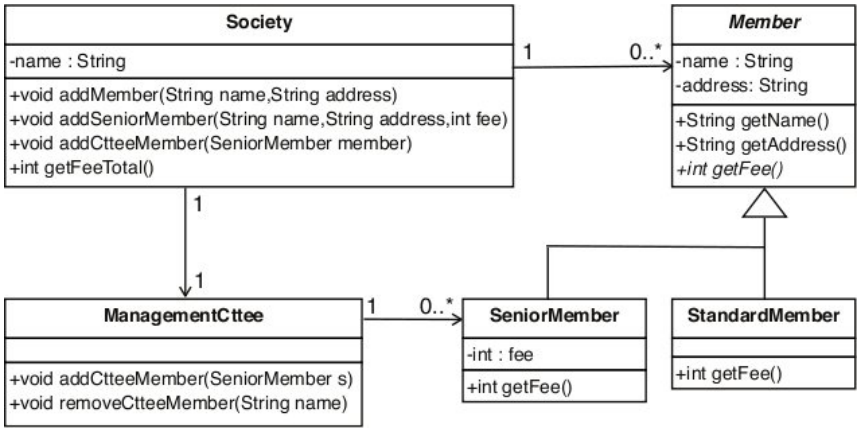
A seguire, trovate trovate il codice dell'interfaccia `Vendor` e della classe `Item`

```
public interface Vendor
{
    void addStock(Item item);
    void selectItem(String name);
    void selectItem(String name, boolean isLarge);
    Item getCurrentChoice();
    void deliver();
}
```

```
public abstract class Item
{
    private int price;
    private String name;
    private boolean isLarge;

    public Item(int price)   { this.price = price; }

    public boolean isLarge() { return isLarge;     }

    public void setLarge(boolean isLarge)
    {
        this.isLarge = isLarge;
        if (isLarge) price = price * 2;
    }

    public String getName()  { return name;       }

    public void setName(String name) { this.name = name; }

    public String toString()
    {
        String size = "small";
        if (isLarge())  size = "large";
        String s = String.format("Item = %s, Size = %s, Price = %d",
                                 name, size, price);
        return s;
    }

    public abstract void deliver();
}
```

4 Esercizio

Considerate il seguente diagramma che descrive parte di una applicazione per gestire informazioni relative all'appartenenza ad una società professionale.



Una società può avere $n \geq 0$ associati, di tipo `Member`, ed un comitato di gestione, di tipo `ManagementCttee`, il quale a sua volta è formato da 0 o più `SeniorMember`. I campi ed i metodi di ciascuna classe sono illustrati nel diagramma (con − indichiamo un livello di visibilità `private`, con + il livello `public`).

1. Scrivete il codice Java della classe `ManagementCttee`, realizzandone i metodi indicati, definendone i campi che ritenete opportuni, e definendo un costruttore `public ManagementCttee()`
2. Scrivete il codice Java della classe `Member`, realizzandone i metodi indicati. La classe è dotata di un costruttore `public Member(String name, String address)` ed è abstract in quanto il metodo `getFee()` è abstract.
3. Scrivete il codice Java della classe `StandardMember`, realizzandone i metodi, dotandola di un costruttore
   `public StandardMember(String name, String address)`
   e assumendo che ogni `StandardMember` abbia una tassa di iscrizione, (`fee`), di 50 (Euro).
4. Scrivete il codice Java della classe `SeniorMember`, realizzandone i metodi e dotandola di un costruttore
   `public SeniorMember(String name, String address, int fee)`
   Scrivete il codice Java della classe `Society`, realizzandone i metodi, defininendo i campi necessari e dotandola di un costruttore `public Society(String name)`.

5 Esercizio

Completate l'implementazione del metodo seguente come richiesto dalla specifica:

```
  /**
   * Restituisce un iteratore che enumera in sequenza tutti gli elementi della
   * lista l che occorrono in posizione pari. La vostra implementazione non
   * deve creare una nuova lista, n\'e modificare la  lista l passata come.
   * argomento. Non \'e necessario implementare il metodo remove delliteratore
   */

public static <T> Iterator<T> iterator(List<T> l)
{
    // Completare come da specifica









}
```