PROGRAMMAZIONE II (A,B) - a.a. 2015-16
Prima Valutazione Intermedia — Novembre 2015

Esercizio 1

Si consideri il tipo di dati astratto modificabile `BoxOffice` che intende rappresentare i dati relativi alla vendita di biglietti per un evento. L'interfaccia Java presentata di seguito rappresenta una descrizione parziale dell'astrazione `BoxOffice`.

```
public interface BoxOffice {

  /* restituisce la capienza dell'evento */
  public int getCapacity();

  /* restituisce il numero di biglietti ancora disponibili */
  public int ticketAvailable();

  /* restituisce il costo del singolo biglietto */
  public int getPrice();

 /* richiesta di acquisto di biglietti da parte di un rivenditore autorizzato */
  public boolean sellTicket(int num, String seller);

  /* richiesta di biglietti omaggio da parte di un rivenditore autorizzato */
  public boolean getTicket(int num, String seller);
}
```

1. Assumendo di adottare una strategia di programmazione difensiva, si completi la specifica del tipo di dato `BoxOffice`, definendo le clausole REQUIRES, MODIFIES e EFFECTS di ogni metodo, incluso i costruttori, indicando le eccezioni eventualmente lanciate e se sono checked o unchecked.

   Si veda il file `BoxOffice.java`. Si noti come la strategia difensiva obblighi a lanciare le eccezioni nel caso di parametri uguali a null.

2. Si definiscano la struttura di implementazione concreta, la funzione di astrazione e l'invariante di rappresentazione.

   Si veda il file `BoxOfficeList.java`.

3. Si fornisca l'implementazione del metodo `sellTicket` e si dimostri che l'implementazione preserva l'invariante di rappresentazione.

   Le istruzioni condizionali garantiscono che i vincoli su `capienza`, `disponibili` e `prezzo` presenti in `IR` sono rispettati. Considerando `AF`, l'effetto del metodo concreto corrisponde esattamente a quello del metodo astratto.

4. Supponiamo di estendere l'astrazione `BoxOffice`. Si definisca la classe `BrockeredBoxOffice` che estende `BoxOffice` in modo tale che un rivenditore autorizzato possa richiedere un numero massimo di biglietti (comprensivo di quelli omaggio, lo stesso numero per ogni rivenditore). Si fornisca la specifica e l'implementazione del tipo di dati `BrockeredBoxOffice` e si indichi, motivando la risposta, se è soddisfatto il principio di sostituzione.

   Si veda il file `BrockeredBoxOffice.java`. Il principio di sostituzione è soddisfatto, dato che nella sottoclasse i metodi `sellTicket` e `getTicket` hanno la post-condizione rafforzata.

Esercizio 2

Si consideri il seguente frammento di codice Java

```
class Shape { /* ... */ }
class Circle extends Shape { /* ... */ }
class Rectangle extends Shape { /* ... */ }
class Node<T> { /* ... */ }
```

Si verifichi, motivando la risposta, se il codice descritto di seguito supera il controllo dei tipi.

```
Node<Circle> nc = new Node<>();
Node<Shape>  ns = nc;
```

Il codice non supera il controllo. Mentre la prima istruzione è corretta, la seconda no, dato che `Node<Circle>` *non è sotto-tipo di* `Node<Shape>`*.*

Esercizio 3

Si consideri il seguente frammento di codice Java

```
public abstract class C {
  public abstract Set<String> get(String s);
}

public class B extends C {
  public Set<String> get(String s) {
    if (s.equals("pr2") {
      Set<String> result = new TreeSet<String>();
      result.add("pr2");
    }
    return null;
  }
}
```

1. Si indichino il tipo statico e il tipo dinamico di `result`.

   `result` ha tipo statico `Set<String>` *e tipo dinamico* `TreeSet<String>`*.*

2. In quale situazione il codice Java genera una eccezione? E quale tipo di eccezione?

   Nel caso `s` *sia* `null`*, lanciando una* `NullPointerException`*.*

3. Si consideri il codice `main`

   ```
   public static void main(String[] args) {
     C c = new C();
     Set<String> cc = c.get("pr2");
   }
   ```

   Si verifichi, motivando la risposta, se il metodo `main` viene compilato correttamente.

   No, dato che una classe astratta non può essere inizializzata.