PROGRAMMAZIONE II (A, B) - a.a. 2017-18
Prima Valutazione Intermedia – 2 Novembre 2017

Esercizio 1

Si vuole progettare un tipo di dato astratto `Game` che intende rappresentare una gara sportiva. Alla gara partecipano un numero di atleti fissato nel momento in cui la gara viene creata. Ad ogni atleta corrisponde un numero di pettorale $1, 2, \ldots, n$, dove n è il numero totale dei partecipanti alla gara. L'operazione `SetResult(num, res)` indica che l'atleta con il numero di pettorale `num` ha gareggiato ottenendo il risultato `res`. L'operazione `first()` restituisce il numero di pettorale dell'atleta in testa al momento nel quale l'operazione viene eseguita. Se un atleta non ha ancora gareggiato il suo risultato è convenzionalmente fissato a `null`. L'interfaccia di seguito rappresenta una descrizione parziale dell'astrazione `Game`.

```
public interface Game {

    // assegna al concorrente num il risultato res
   public void setResult(int num, V res)

   // restituisce il miglior risultato fra quelli conseguiti fino a quel momento
   public int first()

   // restituisce la lista dei risultati conseguiti fino a quel momento
   public List<V> results();
}
```

1. Assumendo di adottare una strategia di programmazione difensiva, si completi il progetto del tipo di dato astratto `Game`, definendo le clausole REQUIRES, MODIFIES, e EFFECTS di ogni metodo, indicando le eccezioni eventualmente lanciate e se sono checked o unchecked.

   Si veda l'interfaccia `Game.java`

2. Nell'ipotesi in cui la struttura di implementazione concreta del tipo di dato astratto `Game` sia

   ```
   private int firstIndex;
   private V[] ress;
   ```

   si definiscano la funzione di astrazione e l'invariante di rappresentazione.

   Si veda la classe `ArrayGame.java`

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

4. Si vuole ora trattare un caso più generale di gara, nel quale ogni atleta può sostenere più prove, delle quali viene memorizzata quella con il risultato migliore. Si definisca quindi una classe `FullGame`, che estende l'implementazione di `Game` aggiungendo e/o modificando tutti e soli i metodi che si ritengono necessari.

   Si veda la classe `FullGame.java`.

5. Giustificando la risposta, si dica se `FullGame` verifica il principio di sostituzione.

   Assumendo `A` come la condizione che il risultato sia migliore e `B` come l'eventuale modifica di this, per `setResult` abbiamo che $post_{AG}$ = `B` e $post_{FG}$ = `A ==> B`, e dato che `A ==> B` non implica `B` (ovvero, la formula `(A ==> B) ==> B` non è una tautologia), `FullGame` non è sotto-tipo di `ArrayGame`.

Esercizio 2

Si consideri il seguente programma in Java

```
public class B {
  public void foo(B obj) {
    System.out.print("B1 ");
  }
  public void foo(C obj) {
    System.out.print("B2 ");
  }
}

public class C extends B {
  public void foo(B obj) {
    System.out.print("C1 ");
  }
  public void foo(C obj) {
    System.out.print("C2 ");
  }
  public static void main(String[] args) {
    B c = new C();
    B b = new B();
    b.foo(c);
    c.foo(b);
    c.foo(c);
  }
}
```

- Motivando la risposta, si dica quale è il risultato osservabile dell'esecuzione del metodo `main`.

  Il risultato osservabile dell'esecuzione del metodo ***`main`*** *è la stampa* ***`B1 C1 C1`****.*