PROGRAMMAZIONE II 2017-18
Esercitazione

Esercizio 1

Si consideri la collezione generica `Sequence<E>` che intende rappresentare una sequenza finita di oggetti di tipo `E`. Una sequenza è caratterizzata dalla sua lunghezza e i suoi elementi sono determinati dalla posizione a partire dalla posizione 0. La classe astratta Java presentata di seguito rappresenta una descrizione parziale dell'astrazione `Sequence<E>`.

```
public abstract class Sequence<E> {

   /* restituisce la lunghezza della sequenza */
   public abstract int length();

   /* restituisce la prima posizione nella sequenza il cui elemento corrispondente e'
   uguale a elem */
   public abstract int indexOf(E elem);

   /* restituisce la prima posizione nella sequenza a partire dalla quale gli elementi
   successivi formano la sequenza s */
   public abstract int indexOfSlide(Sequence<E> s);

   /* modifica la sequenza inserendo alla posizione pos l'oggetto elem */
   public abstract void update(int pos, E elem);

   /* controlla se la sequenza iniziale coincide con il parametro s */
   public boolean checkInitial(Sequence<E> s) {
      ...
   }
}
```

1. Si completi la scrittura del metodo `checkInitial`.
2. Assumendo di adottare una strategia di programmazione difensiva, si completi il progetto della collezione `Sequence<E>`, definendo le clausole REQUIRES, MODIFIES e EFFECTS di ogni metodo, incluso i costruttori, indicando le eccezioni eventualmente lanciate e se sono checked o unchecked.
3. Si definiscano la struttura di implementazione concreta, la funzione di astrazione e l'invariante di rappresentazione.
4. Si fornisca l'implementazione del metodo `update` e si dimostri che l'implementazione preserva l'invariante di rappresentazione.
5. Supponiamo di estendere l'astrazione `Sequence<E>`. Si definisca la classe `FilteredSequence<E>` che estende la classe che implementa `Sequence<E>` in modo tale che il metodo `update` inserisca l'elemento solamente se l'elemento non appartiene a una lista di indesiderati. Si fornisca la specifica e la struttura dell'implementazione concreta della collezione `FilteredSequence<E>` e si indichi, motivando la risposta, se è soddisfatto o meno il principio di sostituzione.

Esercizio 2

Si considerino le seguenti dichiarazioni

```
class A {
   void f() { System.out.println("A.f()"); }
   void f(int n) { System.out.println("A.f(int)"); }
   void g(int n) { System.out.println("A.g(int)"); }
}
class B extends A {
   void f() { System.out.println("B.f()"); }
   void h() { System.out.println("B.h()"); }
}
class C extends B {
   void g(int n) { System.out.println("C.g(int)"); }
   void g(String s) { System.out.println("C.g(String)"); }
   void h(int n) { System.out.println("C.h(int)"); }
}
```

Si consideri i seguenti frammenti di codice

(a) `A a = new C(); a.f();`

(b) `A a = new C(); a.g(10);`

(c) `A a = new C(); a.g("x");`

Per ogni frammento si indichi, motivando la risposta, se frammento viene compilato correttamente e nel caso quale è il risultato della esecuzione.

Esercizio 3

Si consideri il metodo Java descritto di seguito

```
public static List<Integer> removeHigher(List<Integer> lst, Integer low) {
   List<Integer> tmp = new ArrayList<Integer>;
   for(int i = 0; i < lst.size(); i++)
      if (lst.get(i) > low)
         tmp.add(lst.remove(i));
   return tmp;
}
```

Si modifichi il codice del metodo rispettando il suo comportamento funzionale in modo da renderlo maggiormente *generico* e pertanto utilizzabile da un maggior numero di clienti. Si motivi la risposta.