PROGRAMMAZIONE II (A, B) - a.a. 2015-16
Secondo Appello — Febbraio 2016

Esercizio 1

Si consideri il tipo di dati astratto modificabile `NewCollection<E>` che rappresenta una collezione di elementi generici (non `null`) caratterizzati da una specifica posizione nella collezione. In altri termini il primo elemento nella collezione si trova alla posizione 0, il secondo alla posizione 1, etc. Supponiamo che la definizione del tipo di dato `NewCollection<E>` contenga, tra gli altri, i metodi

```
/* inserisce l'elemento elem nella collezione alla posizione p,
   spostando l'elemento contenuto in precedenza in una posizione libera */
public void insertAt(E elem, int p);

/* restituisce l'elemento alla posizione p nella collezione */
public E get(int p);

/* restituisce la posizione della prima occorrenza dell'elemento elem,
   altrimenti -1 se elem non appartiene alla collezione */
public int indexOf(E elem);
```

1. Assumendo di adottare una strategia di programmazione difensiva, si completi la specifica dei metodi `insertAt`, `get` e `indexOf` definendo le clausole REQUIRES, MODIFIES e EFFECTS, indicando le eccezioni eventualmente lanciate e se sono checked o unchecked.

   Si veda il file NewCollection.java.

2. Si consideri la seguente struttura di implementazione per la classe `NewCollection<E>`

   ```
   private Object[] elementData; // array che memorizza gli elementi della collezione
   private int size;     // numero di elementi attualmente contenuti nella collezione
   ```

   Si definiscano la funzione di astrazione e l'invariante di rappresentazione per l'implementazione di `NewCollection<E>`.

   Si veda il file NewCollectionImpl.java.

3. Si fornisca l'implementazione dei metodi `insertAt` e `indexOf` e si dimostri che le implementazioni proposte preservano l'invariante di rappresentazione.

   Si veda il file NewCollectionImpl.java.

Esercizio 2

Si consideri il linguaggio didattico funzionale, e si estenda tale linguaggio con una nuova nozione di blocco della forma **use** x = E1 **or** E2 **in** E **when** G, dove E1, E2 e E sono espressioni generiche mentre G è una espressione a valori booleani detta guardia. La valutazione del blocco **use** avviene nel modo seguente

- si valuta la guardia;
- se la guardia viene valutata a true si prosegue con la valutazione di E nell'ambiente esteso con il legame tra x e il valore di E1;
- se la guardia viene valutata a false si prosegue con la valutazione di E nell'ambiente esteso con il legame tra x e il valore di E2.

1. Assumendo scoping statico, si mostri come deve essere modificato l'interprete del linguaggio didattico funzionale.

Sintassi

```
type exp = ...
   | Use of ide * exp * exp * exp * exp
```

Controllo tipi

```
xxxx
```

Semantica

```
let rec sem e r = match e with
   ...
   | Use(i, e1, e2, e, g) -> match sem(g, r) with
                               true -> let v = sem(e1, r) in sem(e, bind(i, v, r))
                             | false -> let v = sem(e2, r) in sem(e, bind(i, v, r))
                             | _ -> failwith "wrong exp"
```

Esercizio 3

Si consideri il seguente programma Java

```
public class A {
    protected String local;
    public A() { local = new String(); }
    public void add(String param) {
        local = local.concat(param);                          // 1
    }
    public void ntimes(int n) {
        for (int i=1; i<=n; i++) {
           local.concat(local);
           System.out.println(local);
       }
    }
}

public class B extends A {
    private Integer val;
    public B(int n) {
        super();
        val = new Integer(n);
    }
    public int addInt(Integer param) {
        val = new Integer(param.intValue() + val.intValue());
        return val.intValue();
    }
    public void ntimes(int n) {
        for (int i=1; i<=n; i++) {
           local.concat(local);
           val = new Integer(val.intValue() + val.intValue());
       }
       System.out.println(local);
       System.out.println(val);
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        A a = new A();
        B b1 = new B(1);
        B b2 = new B(2);
        b1.add("abba");
        a = b1;
        a.ntimes(b2.addInt(2));
    }
}
```

1. Si dica quali sono i valori stampati dal programma. *`"abba"` e `16`.*
2. Si descriva la struttura del run-time (stack e heap) nella valutazione del programma.
3. Quali valori stampa il programma se rimuoviamo il comando di assegnamento alla riga 1 lasciando solo l'invocazione del metodo `local.concat(param);` ? *`""` (stringa vuota) e `16`.*