PROGRAMMAZIONE II (A,B) - a.a. 2019-20
Primo Appello – 17 Gennaio 2020–Traccia Soluzione

Domande di base

1. Si consideri il seguente programma (scritto con una sintassi Java-like)

```
class A  {public void m1() {...};}
class B extends A {public void m2() {...};}
class C extends B {public void m1() {...};}

\\ main
A a = new B();
A b = new C();
b=a;
b.m1();
b.m2();
```

   Si dica se il `main` supera il controllo statico dei tipi. Motivare la risposta.

   Il programma non supera il controllo statico dei tipi dato che la variabile b ha tipo statico A. In particolare la chiamata del metodo m2 non supera il controllo dei tipi: m2 viene dichiarato nella classe B che è sottotipo di A .

2. Si dica se nel caso di *blocchi in-line* le regole di *scope* statico e dinamico si comportano nello stesso modo. Motivare la risposta.

   Le due regole si comportano nello stesso modo dato che i blocchi in-line *non contengono dichiarazioni di funzioni e di conseguenza neanche chiamate di funzioni. Di conseguenza in questo caso i blocchi seguono l'ordine di annidamento standard di tipo LIFO.*

3. Si descriva il ruolo del *contatore dei referimenti* nella tecnica di garbage collection detta *reference counting*. Questa tecnica consente di determinare come garbage strutture dati con cicli interni? Motivare la risposta.

Esercizio 1

Si consideri un tipo di dato astratto *GenericContainer* introdotto per rappresentare una collezione di oggetti generici. Gli elementi della collezione sono disposti in ordine qualunque e sono di molteplicità qualunque (sono ammesse ripetizioni). Elementi di valore null non devono fare parte della collezione. L'interfaccia generica `GenericContainer <T>` include, tra gli altri, i seguenti metodi

- `public void insert(T elem)` *il cui effetto è quello di inserire l' elemento generico* `elem` *nella collezione*
- `public int delete(T elem)` *il cui effetto è quello di rimuovere una occorrenza dell'elemento* `elem` *dalla collezione e di restituire il numero degli elementi uguali a* `elem` *ancora presenti nella collezione*
- `public int delOccurrences(T elem, int n)` *il cui effetto è quello di rimuovere un numero* `m` *di occorrenze dell'elemento* `elem` *dalla collezione, dove* $m \leq n$*, e di restituire il numero degli elementi uguali a* `elem` *effettivamente rimossi (ovvero* `m`*)*
- `public int size ()` *restituisce il numero degli elementi presenti nella collezione*
- `public int getOccurrences(T elem)` *il cui effetto è quello di restituire il numero delle occorrenze dell' elemento* `elem` *nella collezione*

- `public T minOccurrences()` *il cui effetto è quello di restituire l' elemento della collezione avente il minor numero di occorrenze*

1. Si definisca la specifica dell'interfaccia `GenericContainer<T>`, indicando l'elemento tipico e per ogni metodo le clausole REQUIRES, MODIFY ed EFFECTS, il valore restituito e le eventuali eccezioni lanciate in dipendenza dei parametri attuali.

```
public interface GenericContainer<T> {
   // Overview: Tipo modificabile dei multi-insiemi di elementi generici di tipo T
   // Typical Element: {<e_1, occ_1>, <e _2, occ_2>, ..., <e_k, occ_k>}
   // IR = (for all i. 1 <= i <= k  ==> occ_i > 0  )
   //      && (for all i, j. 1 <= i < j <= k ==> (!e_i.equals(e_j)))


  public void insert(T elem) throws NullPointerException;
   // REQUIRES: elem!= null
   // THROWS: se elem = null lancia NullPointerException
   // MODIFIES: this
   // EFFECTS: incrementa la seconda componente dell'unica coppia <e_i, occ_i>
   // per la quale elem.equals(e_i); se la coppia non esiste aggiunge <elem, 1>

  public int delete (T elem) throws NullPointerException;
   // REQUIRES: elem != null
   // THROWS: se elem = null lancia  NullPointerException
   // MODIFIES: this
   // EFFECTS: decrementa di 1 la seconda componente dell'unica coppia <e_i, occ_i>
   // per la quale elem.equals(e_i) e  restituisce occ_i-1;
   // se la coppia non esiste restituisce 0.

  public int delOccurrences (T elem, int n) throws
                    NullPointerException, NegativeOccException;
    // REQUIRES: elem!= null && n > 0
     // THROWS: se elem = null lancia NullPointerException
      //        se num <= 0 lancia NegativeOccException
     // MODIFIES: this
     // EFFECTS:decrementa di n la seconda componente dell'unica coppia <e_i, occ_i>
     //  per la quale elem.equals(e_i), rimuovendola se e_i - n <= 0,
     //   e restituisce il minimo fra occ_i e n altrimenti;
    //  se la coppia non esiste restituisce 0.

 public int size();
 //EFFECTS: restituisce la somma della seconda componente delle coppie presenti
 // nella collezione

 public int getOccurrences(T elem) throws NullPointerException;
   // REQUIRES: elem != null
   // THROWS: se elem = null lancia NullPointerException
  // EFFECTS: restituisce la seconda componente dell'unica coppia <e_i, occ_i>
  //  per la quale elem.equals(e_i); se la coppia non esiste restituisce 0

 public T minOccurrences() throws EmptyException;
  // REQUIRES: this non  vuoto
  // THROWS: se this  vuoto lancia EmptyException
 //  EFFECTS: restituisce la prima componente e_i della coppia <e_i, occ_i>
 // tale che occ_i e' minore o uguale delle molteplicita' di ogni altro elemento
 // del multi-insieme
```

2. Si assuma di implementare la classe `MyGenericContainer<T>` che implementa `GenericContainer<T>` con le seguenti variabili d'istanza:

   `private ArrayList<Pair<T>> elts;`

```
private static class Pair<T> {
    //Overview: Tipo immutabile rappresentante una coppia (T,int)
          T t;
          int i;
          Pair(T y, int j) {t = y; i = j;}
          boolean contains(T x) { return (x.equals(t));}
          int occurrences( ) { return i;}
       }
```

(a) Si definiscano la funzione di astrazione e l'invariante di rappresentazione.

$$AF = \{< elts.get(i).t, elts.get(i).occurrences() > | 0 <= i < elts.size()\}$$

$$IR = elts! = null \;\&\&\; (\text{for all } \; i.\; 0 <= i < elts.size() ==> elts.get(i)! = null \;\&\&$$
$$elts.get(i).t! = null \;\&\&\; elts.get(i).occurrences() > 0) \;\&\&$$
$$(\text{ for all } i, j.\; 0 <= i < j < elts.size() ==> (!elts.get(i).t.equals(elts.get(j).t)))$$

(b) Si implementino il costruttore ed i metodi `insert` e `delOccurrences` e si dimostri che l'implementazione proposta preserva l'invariante di rappresentazione.

```
public MyGenericContainer<T>() {
                        elts= new ArrayList<Pair<T>>();
                        }

 public void insert(T elem) throws NullPointerException{
   if (elem== null) throw new NullPointerException();
   int tmp= this. getOccurrences(elem);
   if (tmp ==0) then {elts.add(new Pair(elem,1));}
   else {
            for(int i= 0; i < elts.size(); i++)
               if (elts.get(i).contains(elem))
               then
               {elts.set(i, new Pair(elem, tmp+ 1)); return;}
            }
         }

  public int delOccurrences (T elem, int n) throws
                     NullPointerException, NegativeOccException{

    if (elem == null) throw new NullPointerException();
    if (n <= 0) throw new NegativeOccException();
    int tmp= this. getOccurrences(elem);
    if (tmp==0) then {return 0;}
    for(int i = 0; i < elts.size(); i++)
        if (elts.get(i).contains(elem)) then
           {if (tmp > n)then {
             elts.set(i, new Pair(elem, tmp -n));
             return n;}
             else
            {elts.removeElementAt(i);
            return tmp;}
            }
    }
```

3. Si consideri la classe `OrderedGenericContainer<T>` in cui gli elementi della collezione sono ordinati in modo decrescente rispetto al numero di occorrenze. La classe `OrderedGenericContainer<T>` può essere sottotipo di `MyGenericContainer<T>` secondo il principio di sostituzione di Liskov? Motivare le proprie risposte.
La classe `OrderedGenericContainer<T>` può essere sottotipo di `MyGenericContainer<T>` secondo il principio di sostituzione di Liskov. Infatti il caso ordinato è un caso particolare di non ordinato ovvero ordinato in modo arbitrario.

Esercizio 2

Si consideri il seguente programma OCAML

```
let equalfun f g =
  let rec equalfrom x = f  x = g  x && equalfrom (x+1)
     in  equalfrom 1;;
let mult_sum (x, y) =
    let z = x + y in
       fun w -> w * z;;
let z = 4;;
let mf2 x = x*z;;
let f = mult_sum (3, 4);;
equalfun f mf2;;
```

Si simuli la valutazione del programma mostrando la struttura della pila dei record di attivazione. *Il programma termina restituendo false.*

Esercizio 3

Estendere il linguaggio didattico funzionale con una nuova forma di astrazione funzionale `both-fun` che consente di applicare una coppia di funzioni non ricorsive ad un valore producendo come risultato una coppia di valori. Per esempio in sintassi concreta l'astrazione `both-fun(fun x= x+1, fun x=x*2)` applicata al valore `5` produce come risultato la coppia `<6,10>`.

- Definire le regole di valutazione dell'astrazione `both-fun` e estendere consistentemente la struttura dell'interprete del linguaggio didattico funzionale.

```
type exp = ... BothFun of  ide * exp  * ide * exp
(* definisce le coppie di funzioni *)

(* tipi esprimibili*)
type evT = .........  BothFunVal of evFunP  | Pair of evT * evT
and evFunP = ide * exp *  ide * exp * evT env

let rec eval (e : exp) (r : evT env) : evT = match e with
   ...
   BothFun(x,a,y,b) -> BothFunVal(x,a,y,b,r)
 | FunCall(f,arg)  ->
        let fClosure = (eval f r) in
           match fClosure with
                              ...................
            BothFunVal(x1,body1,x2,body2,fDecEnv) ->
                   let par=(eval arg r)  in
                       let val1= eval body1 (bind fDecEnv x1 par) in
                            let val2= eval body2 (bind fDecEnv x2 par)
                                in  Pair(val1,val2)

       _ -> failwith("non functional value"))
```