PROGRAMMAZIONE II (A,B) - a.a. 2018-19
Terzo Appello – 25 Giugno 2019– Traccia Soluzione

Domande di base

1. Si descriva brevemente il meccanismo di caricamento dinamico delle classi in Java.
2. Cosa è una *chiusura* e quale è il ruolo delle chiusure nella implementazione del linguaggi di programmazione.
3. Descrivere il modo in cui gli algoritmi di garbage collection *mark and sweep* e *reference counting* trattano strutture dati circolari allocate sullo heap.

Esercizio 1

Si consideri un tipo di dato astratto *HDSet*, *History Dependent Set* che rappresenta un insieme che memorizza anche tutti gli elementi che ha contenuto, anche quelli eventualmente rimossi. La classe `HDSet<E>` è generica rispetto al tipo `E` deli elementi. La classe `HDSet<E>` deve includere, tra gli altri, i seguenti metodi

- `public boolean contained(E el)` il cui effetto è quello di verificare se `el` è fra gli elementi che sono stati contenuti nell'insieme, inclusi quelli che lo sono attualmente.
- `public boolean newInsert(E el)` il cui effetto è quello di inserire `el` nell'insieme se `el` non appartiene all'insieme e non è fra gli elementi che sono stati contenuti.
- `public boolean wasContained(E el)` il cui effetto è quello di verificare se `el` è fra gli elementi che sono stati contenuti nell'insieme, ma non lo sono attualmente.
- `public boolean remove(E el)` il cui effetto è quello di rimuovere `el` nel caso sia presente fra gli elementi che sono attualmente contenuti nell'insieme.

1. Si definisca la specifica dei metodi descritti in precedenza, indicando per ogni metodo **a)** le clausole REQUIRES, MODIFY ed EFFECTS, e **b)** il valore restituito e le eventuali eccezioni lanciate in dipendenza dei parametri attuali.

   Overview della classe. Il tipo `HDSet<E>` descrive un insieme di elementi di tipo E che tiene traccia anche degli elementi che ha contenuto. Typical element: $\langle\{x_0, \ldots, x_n\}, \{y_0, \ldots, y_k\}\rangle$ dove $\{x_0, \ldots, x_n\}$ descrive l'insieme degli elementi attualmente contenuti mentre $\{y_0, \ldots, y_k\}$ tiene traccia degli elementi che sono stati contenuti. Ovviamente abbiamo che $\{x_0, \ldots, x_n\} \cap \{y_0, \ldots, y_k\} = \emptyset$.

   Specifica dei metodi

```
THROWS: se el == null lancia  NullPointerException (eccezione unchecked di  Java)
EFFECTS: se el  appartiene  o  e' stato contenuto  nell'insieme   restituisce true,
    false  altrimenti
public boolean contained(E el)

THROWS: se el == null lancia  NullPointerException (eccezione unchecked di  Java)
MODIFIES:this
EFFECTS: se el non appartiene all'insieme e non   e' mai stato contenuto
allora lo inserisce e restituisce true, false  altrimenti
public boolean newInsert(E el)

THROWS: se el == null lancia  NullPointerException (eccezione unchecked di  Java)
EFFECTS: se el  non appartiene all'insieme ma e'  stato contenuto
 allora  restituisce true, false  altrimenti
public boolean wasContained(E el)

THROWS: se el == null lancia  NullPointerException (eccezione unchecked di  Java)
MODIFIES:this
EFFECTS: se el  appartiene all'insieme allora lo rimuove  e restituisce true,
    false  altrimenti
public boolean remove(E el)
```

2. Si assuma di implementare la classe `HDSet<E>` con la struttura dati

```
private int size;
private ArrayList<E> elts;
private ArrayList<Boolean> presences; // altri parametri
public HDSet<E>( ) { // parametri ignorati
   size = 0;
   elts = new ArrayList<E>();
   presences = new ArrayList<Boolean>();
}
```

- Si definiscano la funzione di astrazione e l'invariante di rappresentazione. La funzione di astrazione mappa la rappresentazione in un elemento astratto quindi in una coppia di insiemi. Il primo insieme contiene gli elementi di *elts* che hanno true nella posizione corrispondente di *presences* mentre il secondo insieme contiene gli elementi di *elts* che hanno false nella posizione corrispondente di *presences*.
  FUNZIONE DI ASTRAZIONE:

$$\langle \{elts.get(i) \mid \ 0 <= i < size \wedge presences.get(i) = true\}, \\ \{elts.get(j) \mid \ 0 <= j < size \wedge presences.get(j) = false\} \rangle$$

  REP-INV:

$$elts! = null \wedge presences! = null \ \wedge \\ size = elts.size() = presences.size() \ \wedge \\ \forall i.\ 0 <= i < size \Longrightarrow (elts.get(i)! = null \wedge presences.get(i)! = null \ \wedge \\ \forall j.(0 <= j < size \ \wedge \ i! = j) \Longrightarrow elts.get(i)! = elts.get(j))$$

- Si implementi il metodo `newInsert` verificando che preservi l'invariante di rappresentazione.

```
THROWS: se el == null lancia  NullPointerException (eccezione unchecked di  Java)
MODIFIES:this
EFFECTS: se el non appartiene all'insieme e non   e' mai stato contenuto
allora lo inserisce e restituisce true, false  altrimenti
public boolean newInsert(E el) {
if (el == null) throw new NullPointerException();
 if   (elts.indexOf(el) != -1) then return false;
elts.add(el) ;
presences.add(true);
size++;
return true;       }
```

Esercizio 2

Si estenda il linguaggio didattico funzionale con il costrutto `StaticFun of ide * exp` per introdurre funzioni non ricorsive *statiche*. Una funzione statica è una funzione che non elimina l'ambiente locale della funzione: l'ambiente locale viene preservato per l' invocazione successiva. Si noti che l'espressione `exp` è definita da una sequenza di costrutti *let* annidati e dal corpo della funzione.

(a) Si discuta come deve essere modificato l'interprete del linguaggio didattico funzionale.

Traccia soluzione

```
Modifica della sintassi astratta.
Tenendo conto della testo dell' esercizio sulla struttura delle funzioni statiche.


type letseq  = Empty | LetItem * ide * exp * letseq
type bodyexp = exp senza il costruttore Let

type exp = .... | StaticFun of ide * letseq * bodyexp

type evT = ..... | StaticFunVal of ide * bodyexp * evT env


Modifica della funzione di valutazione

let rec eval  (e:exp) (r: evT env): evT =
                     match e with
                           :
           | StaticFun(i,l,body) -> StaticFunVal(i, body, (evalLetSeq l r))

           | FunCall(e1,e2) -> match eval e1 r with
                                        :
                                | StaticFunVal(i,b, envstatic) ->
                                      evalbody b (bind envstatic i (eval e2 r))

 dove
   let rec evalLetSeq (e:letseq) (r: evT env): evT env = match e with
      | Empty -> r
      | LetItem(i, e1, es) -> let v = eval e1 r in  evalLetSeq es (bind r i v)

 e evalbody = eval (senza le clausole per valutazione Let)
```

Esercizio 3

Si consideri il seguente programma OCaML

```
let z =2;;

let f1 = fun x y  -> x + y *z;;

let rec apply_n_times f n x =
  if n<=0 then x
  else apply_n_times f (n-1) (f x);;

let rec map_n_times g n = function
  | [] -> []
  | h1::ls -> (apply_n_times g n h1) :: map_n_times g n ls;;

let z = 3;;

let ff = f1 1;;

map_n_times ff z [10;20];;
```

(a) Si simuli la valutazione del programma mostrando la struttura della pila dei record di attivazione.

(b) Si determini il valore calcolato dal programma. *Il valore calcolato è* $[87; 167]$