

PROGRAMMAZIONE II (A,B) - a.a. 2017-18

Secondo Appello – 5 febbraio 2018

Esercizio 1

Si consideri il tipo `BoundedSortedMap<K,V>`, semplificazione del tipo analogo presente nella libreria Java, che rappresenta una mappa nella quale le chiavi sono ordinate e devono essere più piccole di una chiave massima fissata al momento della creazione.

Fra gli altri, l'interfaccia ha a disposizione i seguenti metodi

```
public interface BoundedSortedMap<K, V> {  
    /* restituisce true se la mappa ha un valore associato alla chiave key */  
    boolean containsKey(Object key);  
  
    /* elimina la chiave key; restituisce il valore precedentemente associato alla chiave,  
       se presente, null altrimenti */  
    V remove(Object key);  
  
    /* associa il valore value alla chiave key; restituisce il valore precedentemente  
       associato alla chiave, se presente, null altrimenti */  
    V put(K key, V value);  
  
    /* restituisce la piu' piccola chiave che ha associato un valore */  
    K firstKey( );  
  
    /* restituisce una mappa ristretta alla chiavi piu' grandi o uguali a fromKey */  
    BoundedSortedMap tailMap(K fromKey);  
}
```

1. Assumendo di adottare una strategia di programmazione difensiva, si completi la specifica dei metodi, definendo le clausole `REQUIRES`, `MODIFIES` e `EFFECTS`, indicando le eccezioni eventualmente lanciate e se sono `checked` o `unchecked`.
2. Si consideri la seguente struttura di implementazione per la classe `BoundedSortedMap`

```
private K[] keys;  
private V[] values;
```

Si definiscano sia la funzione di astrazione che l'invariante di rappresentazione per l'implementazione `MyBoundedSortedMap` di `BoundedSortedMap`.

3. Si fornisca l'implementazione dei metodi `firstKey` e `put` e si dimostri che preservano l'invariante di rappresentazione.

Si veda il file `MyBoundedSortedMap.java`.

4. Si consideri la classe `MyIntervalSortedMap` che estende `MyBoundedSortedMap` in modo che, oltre a una chiave massima, il dominio abbia anche una chiave minima, sempre fissata al momento della creazione.

Si descriva l'invariante di rappresentazione della nuova classe, si modifichino i metodi necessari e, giustificando la risposta, si dica se l'estensione proposta verifica il principio di sostituzione.

Si veda il file `MyIntervalSortedMap.java`. Non vale il principio di sostituzione perché la regola dei metodi non è soddisfatta.

Esercizio 2

Si estenda il linguaggio didattico funzionale con il costrutto `CodaLimitata` per la definizione di code con lunghezza massima prefissata. In aggiunta, il linguaggio è esteso con le operazioni primitive `insert` e `remove`, che rispettano la politica FIFO, e `peek`, che restituisce l'elemento in cima alla coda.

1. Si mostri come deve essere modificato l'interprete del linguaggio didattico funzionale.

Una soluzione minimale è riportata di seguito. Una soluzione di stile più funzionale al posto della `peek` avrebbe introdotto dei costruttori per gestire un risultato più complesso dell'operazione di rimozione, che intuitivamente dovrebbe restituire una coppia.

```
type exp = ...
  | CodaLimitata of exp * queue
  | Insert of exp * exp
  | Remove of exp
  | Peek of exp
and queue = Empty | Item of exp * queue

type evT = ...
  | CodaLimVal of evT * (evT list)

let rec eval (e : exp) (r : evT env) : evT = match e with
...
  | CodaLimitata(i, q) ->
    let iv = (eval i r) in
    let ql = (evalQ q r) in
    (match iv with
      Int ivv -> if ((List.length ql) <= ivv)
                  then CodaLimVal(iv, ql)
                  else failwith("wrong limit") |
      _ -> failwith("not a limit"))
...
  | Insert(e1, e2) ->
    let q = (eval e1 r) in
    (match q with
      CodaLimVal(lim, ql) -> if ((List.length ql) < lim)
                              then CodaLimVal(lim, ql@[eval e2 r])
                              else failwith("out of bound") |
      _ -> failwith("not a queue"))
  | Remove(e1) ->
    let q = (eval e1 r) in
    (match q with
      CodaLimVal(lim, v::ql) -> CodaLimVal(lim, ql) |
      CodaLimVal(_, []) -> failwith("empty queue") |
      _ -> failwith("not a queue"))
  | Peek(e1) ->
    let q = (eval e1 r) in
    (match q with
      CodaLimVal(_, v::ql) -> v |
      CodaLimVal(_, []) -> failwith("empty queue") |
      _ -> failwith("not a queue"))
...
and let rec evalQ (q : queue) (r : evT env) : evT list = match q with
  Empty -> []
  | Item (e, q1) -> (eval e r)::(evalQ q1 r)
```

Esercizio 3

Si consideri il seguente programma OCaml

```
let rec fold2 f g l =
  match l with
  | [] -> false
  | x::[] -> g x
  | x::ls -> f (fold2 f g ls) (g x);;

let imp n m =
  if n then m
  else true;;

let last x = imp true (x > 0);;

fold2 imp last [3;-4;5];;
```

1. Si determini il tipo inferito dall'interprete OCaml per gli identificatori di funzione (`fold2`, `imp` e `last`) che compaiono nel programma scelto.

```
val fold2 : (bool -> bool -> bool) -> ('a -> bool) -> 'a list -> bool = <fun>
val imp : bool -> bool -> bool = <fun>
val last : int -> bool = <fun>
```

2. Si simuli la valutazione del programma mostrando la struttura della pila dei record di attivazione.
Si veda in fondo al testo.

3. Si determini il valore calcolato dal programma. `- : bool = true`

Esercizio 4. Si consideri il seguente programma scritto in una sintassi C-like.

```
int i = 1;
void foo(int f, int g) {
    g = f;
    f = g - i;
    g = i;
}

int main( ) {
    int a[] = {1, 2, 0};
    foo(i, a[i]);
    printf("%d %d %d %d\n", i, a[0], a[1], a[2]);
}
```

1. Motivando le risposte, si dica quali sono i valori stampati dal programma nel caso di passaggio di parametri per

(a) call-by-value `1 1 2 0`

(b) call-by-reference `0 1 0 0`

(c) call-by-name `0 0 1 0`

A	SL 0	CL 0	F0	code_fold2	A
	fold2	F0			
B	SL A	CL A	F1	code_imp	A
	imp	F1			
C	SL B	CL B	F2	code_last	B
	last	F2			
D	fold2	SL A	CL C	*****	
	imp	f	F1		
	last	g	F2		
	[3;-4;5]	l	[3;-4;5]		
		x	3		
		ls	[-4;5]		
	f(fold2 ...)g(x)	res	true		
E	fold2	SL A	CL D	***	
	f	f	F1		
	g	g	F2		
	ls	l	[-4;5]		
		x	-4		
		ls	[5]		
	f(fold2 ...)g(x)	res	false		
F	fold2	SL A	CL E	*	
	f	f	F1		
	g	g	F2		
	ls	l	[5]		
		x	5		
	g x	res	true		
G	g	SL B	CL F	last	
	x	x	5		
	imp ... (x>0)	res	true		
H	imp	SL A	CL G	*	
	true	n	true		
	x > 0	m	true		
		res	true		
F1	g	SL B	CL E	last	
	x	x	-4		
	imp ... (x>0)	res	false		
G1	imp	SL A	CL F1	**	
	true	n	true		
	x > 0	m	false		
		res	false		
F2	f	SL A	CL E	imp * ** ***	
	fold2 ...	n	true		
	g x	m	false		
		res	false		
E1	g	SL B	CL D	last	
	x	x	3		
	imp ... (x>0)	res	last		
F1	imp	SL A	CL E1	****	
	true	n	true		
	x > 0	m	true		
		res	true		
E2	f	SL A	CL D	imp *** **** *****	
	fold2 ...	n	false		
	g x	m	true		
		res	true		