

# PROGRAMMAZIONE II (A,B) - a.a. 2018-19

## Primo Appello – 14 gennaio 2019 Traccia Soluzione

### Domande di base

1. Si consideri il seguente frammento Java:

```
class A {  
}  
  
class B extends A {  
}  
  
class C extends A {  
}  
  
B b = new B();  
A a = b;  
C c = b;
```

Si dica che il frammento Java supera o meno la fase di compilazione. Si motivi la risposta.

*Il programma non compila dato che c ha tipo statico C e tipo dinamico B. In questo caso B non è sottotipo di C.*

2. Quale è il ruolo del *root-set* nel caso dell'algoritmo di garbage collection denominato *Mark and Sweep*.
3. Descrivere le caratteristiche del passaggio di parametri *per riferimento*.

### Esercizio 1

Si consideri il tipo di dato *buffer* che è un contenitore di dati generici. Un buffer è una sequenza lineare e finita di elementi di uno specifico tipo generico. Le proprietà strutturali di un buffer sono la sua *capacità*, il *limite* e la *posizione*.

- La capacità di un buffer è il numero di elementi che contiene. La capacità di un buffer non è mai negativa e non cambia mai.
- Il limite di un buffer è l'indice del primo elemento che non può essere letto o scritto. Il limite di un buffer non è mai negativo e non è mai superiore alla sua capacità.
- La posizione di un buffer è l'indice dell'elemento successivo da leggere o scrivere. La posizione di un buffer non è mai negativa e non è mai superiore al suo limite.

Supponiamo che la definizione del tipo di dato `Buffer<T>` contenga tra gli altri i metodi

```
interface Buffer<T> {  
  
    /* rende il buffer disponibile per una nuova sequenza di operazioni di lettura o scrittura */  
    void clear();  
  
    /* rende il buffer disponibile per rileggere-riscrivere i dati che contiene */  
    void rewind();  
  
    /* inserisce l'intero contenuto dell'array di elementi generici nel buffer*/  
    void put(T[] src);  
}
```

```

/* trasferisce gli elementi del buffer nell'array generico*/
T[] get();
}

```

1. Assumendo di adottare una strategia di programmazione difensiva, si completi il progetto del tipo di dato astratto `Buffer<T>`, definendo le clausole `REQUIRES`, `MODIFIES`, e `EFFECTS` di ogni metodo, indicando le eccezioni eventualmente lanciate e se sono checked o unchecked.

### Traccia soluzione

```

Overview: Sequenza modificabile di elementi ti tipo T.
Typical element: <cap, limit, pos> <bf@0, bf@1, ... bf@pos, .. bf@limit, .. bf@cap>

Specifica metodi

@requires
@modify this
@effect pos = 0 && forall i, <= 0 < limit bf@i = null
public void clear();

@requires
@modify this
@effect pos = 0 && forall i, <= 0 < limit bf@i != null
public void rewind();

@requires src !=null & src.size() < limit - pos
@modify this
@throws NullPointerException if src = null
@throws IllegalArgumentException if src.size() > limit - pos
@effect forall i, pos <= i < limit buf@i = scr[i-pos]
@effect pos = pos+ scr.size()
public void put(T[] src);

@requires
@result null if pos = 0
@result [bf@0, bf@1, ... bf@pos-1] otheriwise
public T[] get();

```

2. Si consideri la seguente struttura di implementazione per la classe `MyBuffer<T>`

```

private Vector<T> elems;
private int capacity;
private int limit;
private int position;

```

Si definisca l'invariante di rappresentazione per l'implementazione di `MyBuffer<T>`.

### Traccia soluzione

```

REP-INV:
elems != null &&
capacity > 0 &&
limit > 0 &&
pos >= 0 &&
pos < limit <= capacity &&
forall i 0<=i < pos elem[i] != null

```

3. Si fornisca l'implementazione del costruttore e dei metodi `rewind`, `clear` e `put` e si dimostri che le implementazioni proposte preservano l'invariante di rappresentazione.

#### Traccia soluzione

```
public myBuffer<T> (int cap, int lim) {
    if (cap <= 0 || lim > cap)
        throw new IllegalArgumentException();
    capacity = cap;
    limit = lim;
    position = 0;
    Vector<T> elems = new Vector(cap);
}

@requires
@modify this
@effect pos = 0 && forall i, <= 0 < limit bf@i = null
public void clear() {
    position = 0;
    elems = new Vector(capacity);
}

@requires
@modify this
@effect pos = 0 && forall i, <= 0 < limit bf@i != null
public void rewind(){
    position = 0;
}

@requires src !=null & src.size() < limit - pos
@modify this
@throws NullPointerException if src = null
@throws IllegalArgumentException if src.size() > limit - pos
@effect forall i, pos <= i < limit buf@i = scr[i-pos]
@effect pos = pos+ scr.size()

public void put(T[] src){
    if (src = null)
        throw new NullPointerExceptionn();
    if (position + src.length> limit)
        throw new IllegalArgumentException();
    elems.AddAll(position, src) //opportunamente modificato dalle API;
    position = position + src.size();
}
```

4. Si consideri una classe `AlwaysReadWrite<T>` che estende `myBuffer<T>` permettendo di leggere-scrivere sempre gli elementi del buffer. Giustificando la risposta, si dica come deve essere modificata la struttura di implementazione del buffer.

#### Traccia soluzione

```
Refinement: AlwaysReadWrite<T>
    Modificare il costruttore in modo che capacity = limit
```

## 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.

1. Si mostri 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 init = 5;;

let startAt x =
    let incrementBy y = x + y
    in incrementBy;;

let rec applytwice f l = match l with
[] -> []
| x::xs -> f x x :: applytwice f xs;;

applytwice startAt [init+6; init+8];;
```

1. Si simuli la valutazione del programma mostrando la struttura della pila dei record di attivazione. *Vedere file RT.pdf*
2. Si determini il valore calcolato dal programma. *Il valore calcolato è [22;26]*