

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

## Primo Appello – 14 gennaio 2019

### 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.

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.
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>`.

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.
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.

## 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.

## 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.
2. Si determini il valore calcolato dal programma.