Linguaggi e Programmazione Orientata agli Oggetti

Prova scritta

a.a. 2020/2021

12 luglio 2021

1. (a) Per ogni stringa elencata sotto stabilire, motivando la risposta, se appartiene alla seguente espressione regolare e, in caso affermativo, indicare il gruppo di appartenenza (escludendo il gruppo 0).

```
([0-9]+)|([A-Z][a-zA-Z0-9]*)|(move|store|load)|(\s+)
```

   i. `"Move"`
   ii. `"move42"`
   iii. `"store "`
   iv. `"000"`
   v. `"0a"`
   vi. `" "`

   (b) Mostrare che la seguente grammatica è ambigua.

```
Exp ::= Atom + Exp | Atom - Exp | Atom
Atom ::=  Bit | ( Exp ) | Exp
Bit ::= 0 | 1
```

   (c) Modificare la grammatica definita al punto precedente in modo che **non sia ambigua** e che il linguaggio generato a partire dal non terminale `Exp` **resti invariato**.

2. Sia `swap : ('a * 'b) list -> ('b * 'a) list` la funzione così specificata:

   `swap [`(x_1, y_1)`; ... ;`(x_k, y_k)`] = [`(y_1, x_1)`; ... ;`(y_k, x_k)`]`, con $k \geq 0$.

   Esempi:

```
swap [(1,"one");(2,"two");(3,"three")] = [("one", 1); ("two", 2); ("three", 3)]
swap []=[]
```

   (a) Definire `swap` senza uso di parametri di accumulazione.

   (b) Definire `swap` usando `List.map:('a -> 'b) -> 'a list -> 'b list`.

3. Completare il seguente codice che implementa gli alberi di ricerca binari con nodi etichettati da numeri interi (con la relazione d'ordine usuale) e la ricerca di un elemento nell'albero tramite visitor pattern.

   Esempio:

```
var t = new Node(10,new Leaf(5),new Node(42,new Leaf(31),null));
assert t.accept(new Search(31)); // 31 è un'etichetta contenuta nell'albero t
assert !t.accept(new Search(43)); // 43 non è un'etichetta contenuta nell'albero t
assert !t.accept(new Search(30)); // 30 non è un'etichetta contenuta nell'albero t
```

Definizione di albero binario di ricerca: Un albero binario di ricerca con nodi etichettati da numeri interi soddisfa le seguenti proprietà: ogni nodo v è etichettato da un numero intero i, le etichette dei nodi del sottoalbero sinistro di v sono $\leq i$, le etichette dei nodi del sottoalbero destro di v sono $\geq i$.

Completare costruttori e metodi delle classi `Leaf`, `Node` e `Search`:

```
public interface Visitor<T> {
   T visitLeaf(int value);
   T visitNode(int value, BSTree left, BSTree right);
}

public abstract class BSTree {
   protected final int value;
   protected BSTree(int value) { this.value = value; }
   public abstract <T> T accept(Visitor<T> v);
}

public class Leaf extends BSTree { // nodi foglia
   public Leaf(int value) { /* completare */ }
   @Override public <T> T accept(Visitor<T> v) { /* completare */ }
}

public class Node extends BSTree { // nodi interni
   private final BSTree left, right; // left, right possono contenere null, ma non entrambi
   public Node(int value, BSTree left, BSTree right) { /* completare */ }
   @Override public <T> T accept(Visitor<T> v) { /* completare */ }
}

public class Search implements Visitor<Boolean> { // ricerca l'elemento searchedValue nell'albero
   private final int searchedValue; // elemento da cercare
   public Search(int searchedValue) { /* completare */ }
   @Override public Boolean visitLeaf(int value) { /* completare */ }
   @Override public Boolean visitNode(int value, BSTree left, BSTree right) { /* completare */ }
}
```

4. Considerare le seguenti dichiarazioni di classi Java:

```
public class P {
   String m(long l) { return "P.m(long)"; }
   String m(long[] la) { return "P.m(long[])"; }
}
public class H extends P {
   String m(int i) { return "H.m(int)"; }
   String m(int[] ia) { return "H.m(int[])"; }
}
public class Test {
    public static void main(String[] args) {
        P p = new P();
        H h = new H();
        P p2 = h;
        System.out.println(...);
    }
}
```

Dire, per ognuno dei casi elencati sotto, che cosa succede sostituendo al posto dei puntini nella classe `Test` il codice indicato, assumendo che tutte le classi siano dichiarate nello stesso package.

Per ogni caso fornire due o tre righe di spiegazione così strutturate: se c'è un errore in fase di compilazione, specificare esattamente quale; se invece la compilazione va a buon fine spiegare brevemente perché e descrivere cosa avviene al momento dell'esecuzione, anche qui spiegando brevemente perché.

(a) `p.m(42)`

(b) `p2.m(42)`

(c) `h.m(42)`

(d) `p.m(new long[]{42L, 24L})`

(e) `p2.m(new long[]{42L, 24L})`

(f) `h.m(new long[]{42L, 24L})`