Linguaggi e Programmazione Orientata agli Oggetti

Prova scritta

a.a. 2017/2018

24 gennaio 2019

1. (a) Indicare quali delle asserzioni contenute nel seguente codice Java hanno successo e quali falliscono, motivando la risposta.

```
Pattern regEx = Pattern.compile("(String|Float)|(\\s+)|([a-z][\\-a-zA-Z]*)");
Matcher m = regEx.matcher("is-Float String");
m.lookingAt();
assert m.group(3) != null;
assert m.group(0).equals("is-Float");
m.region(m.end(), m.regionEnd());
m.lookingAt();
assert m.group(2) != null;
assert m.group(1) == null;
m.region(m.end(), m.regionEnd());
m.lookingAt();
assert m.group(3) != null;
assert m.group(0).equals("Float");
```

   (b) Mostrare che la seguente grammatica è ambigua.

```
Exp  ::= ref Exp | Exp := Exp | ( Exp ) | Id
Id ::= x | y
```

   (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 `cond_map : ('a -> 'a) -> ('a -> bool) -> 'a list -> 'a list` la funzione così specificata:

   `cond_map` f p l restituisce la lista ottenuta da l applicando, nell'ordine, la funzione f agli elementi di l che soddisfano il predicato p e lasciando invariati i restanti.

   Esempio:

```
# cond_map  sqrt (fun x->x>=0.0) [-1.0;9.0;-4.0;4.0]
- : float list = [-1.0; 3.0; -4.0; 2.0]
```

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

   (b) Definire `cond_map` usando un parametro di accumulazione affinché la ricorsione sia di coda.

   (c) Definire `cond_map` come specializzazione della funzione `it_list` o `List.fold_left`:
   `it_list:('a -> 'b -> 'a) -> 'a -> 'b list -> 'a`

3. (a) Completare le classi `BoolLit` e `And` che rappresentano i nodi di un albero della sintassi astratta corrispondenti, rispettivamente, a literal booleani e all'and logico.

```
public interface AST { <T> T accept(Visitor<T> v); }

public interface Visitor<T> {
   T visitBoolLit(boolean b);
   T visitAnd(AST left, AST right);
}

public class BoolLit implements AST {
   private final boolean value;
   public BoolLit(boolean value) { /* completare */  }
   public <T> T accept(Visitor<T> v) { /* completare */  }
}

public class And implements AST {
   private final AST left, right;
   public And(AST left, AST right) { /* completare */  }
   public <T> T accept(Visitor<T> v) { /* completare */  }
}
```

(b) Completare le classi `Eval` e `ToString` che implementano visitor su oggetti di tipo `AST`.

```
/* Un visitor Eval restituisce il valore dell'espressione,
   calcolato secondo le regole convenzionali;
   la valutazione dell'and logico e' con short-circuit */
public class Eval implements Visitor<Boolean> {
   public Boolean visitBoolLit(boolean b) { /* completare */ }
   public Boolean visitAnd(AST left, AST right) { /* completare */ }
}
/* Un visitor ToString restituisce la rappresentazione in
   notazione polacca postfissa dell'espressione;
   usare String.valueOf per la conversione da boolean a String */
public class ToString implements Visitor<String> {
   public String visitBoolLit(boolean b) { /* completare */ }
   public String visitAnd(AST left, AST right) { /* completare */ }
}
// Classe di prova
public class Test {
   public static void main(String[] args) {
      AST b1 = new BoolLit(true), b2 = new BoolLit(true), b3 = new BoolLit(false);
      AST b1_b2_and_b3_and = new And(new And(b1, b2), b3);
      assert !b1_b2_and_b3_and.accept(new Eval());
      assert b1_b2_and_b3_and.accept(new ToString()).equals("true true && false &&");
   }
}
```

4. Considerare le seguenti dichiarazioni di classi Java:

```
public class P {
   String m(Number n) { return "P.m(Number)"; }
   String m(String s) { return "P.m(String)"; }
}
public class H extends P {
   String m(Number n) { return super.m(n) + " H.m(Number)"; }
   String m(String s) { return super.m(s) + " H.m(String)"; }
}
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(42)`

(e) `p2.m(42)`

(f) `h.m(42)`