Linguaggi e Programmazione Orientata agli Oggetti

Prova scritta

a.a. 2017/2018

12 febbraio 2018

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

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class MatcherTest {
   public static void main(String[] args) {
      Pattern regEx = Pattern.compile("(0[bB][0-1]+)|(0|[1-9][0-9]*)|(\\s+)");
      Matcher m = regEx.matcher("0b1 0");
      m.lookingAt();
      assert m.group(1).equals("0b1");
      assert m.group(0) != null;
      m.region(m.end(), m.regionEnd());
      m.lookingAt();
      assert !m.group(3).equals("0");
      assert m.group(1) == null;
      m.region(m.end(), m.regionEnd());
      m.lookingAt();
      assert m.group(2).equals("0");
      assert m.group(0).length() == 1;
   }
}
```

   (b) Mostrare che la seguente grammatica è ambigua.

```
Type  ::= Type * Type | Type -> Type  | ( Type ) | Id
Id    ::= a | b | c
```

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

2. Sia `insert_after : ('a -> bool) -> 'a -> 'a list -> 'a list` la funzione così specificata: `insert_after` p e l aggiunge l'elemento e nella lista l immediatamente dopo ogni suo elemento che soddisfa il predicato p. Esempio:

```
# insert_after (fun x->x>3) 0 []
- : int list = []

# insert_after (fun x->x>3) 0 [1;4;2;5]
- : int list = [1;4;0;2;5;0]
```

   (a) Definire la funzione `insert_after` senza uso di parametri di accumulazione.

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

   (c) Definire la funzione `insert_after` come specializzazione della funzione `it_list` così definita:

```
let rec it_list f a = function x::l -> it_list f (f a x) l | _ -> a;;
val it_list : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a = <fun>
```

3. Considerare le seguenti classi Java dichiarate nello stesso package.

```
public class P {
    String m(double d) {
        return "P.m(double)";
    }

    String m(float f) {
        return "P.m(float)";
    }
}
public class H extends P {
    String m(double d) {
        return super.m(d) + " H.m(double)";
    }

    String m(int i) {
        return super.m(i) + " 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.

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(42.0)`

(e) `p2.m(42.0)`

(f) `h.m(42.0)`