Linguaggi e Programmazione Orientata agli Oggetti

Prova scritta

a.a. 2018/2019

5 giugno 2019

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("('[^']*')|([$%][a-zA-Z0-9]*)|(\\s+)");
        Matcher m = regEx.matcher("$S3 'hello world'");
        m.lookingAt();
        assert m.group(2).equals("$");
        assert m.group(0).equals("%S3");
        m.region(m.end(), m.regionEnd());
        m.lookingAt();
        assert m.group(3).equals("'");
        assert m.group(0).equals("'hello world'");
        m.region(m.end(), m.regionEnd());
        m.lookingAt();
        assert m.group(1).equals("'");
        assert m.group(0).equals("'hello");
    }
}
```

   (b) Mostrare che la seguente grammatica è ambigua.

```
Exp ::= fst Exp | Exp snd | ( Exp , Exp ) | Id
Id ::= a | b
```

   (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 `gen_sum : ('a -> int) -> 'a list -> int` la funzione così specificata:

   `gen_sum` f `[`x_1`; `x_2`; ... ; `x_n`]` $= f(x_1) + f(x_2) + \ldots + f(x_n)$, con $n \geq 0$.

   Esempi:

```
# gen_sum (fun x->x*x) []
- : int = 0
# gen_sum (fun x->x*x) [1]
- : int = 1
# gen_sum (fun x->x*x) [1;2]
- : int = 5
# gen_sum (fun x->x*x) [1;2;3]
- : int = 14
```

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

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

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

3. Completare la seguente classe di iteratori `Multiples` per generare in ordine crescente la sequenza dei multipli di un numero. Per esempio, il seguente codice

```
for (int n : new Multiples(3, 4)) // genera i primi 4 multipli di 3
    System.out.println(n);
```

stampa la sequenza

```
3
6
9
12
```

```
import java.util.Iterator;

public class Multiples implements Iterator<Integer>, Iterable<Integer> {
        private final int step; // passo tra un elemento della sequenza e il suo seguente
        private final int max; // ultimo numero della sequenza
        private int next; // prossimo numero della sequenza

        /* parametro step: passo tra un elemento della sequenza e il suo seguente
           parametro items: numero totale di elementi della sequenza
           pre-condizione: step > 0 && items >= 0 */
        public Multiples(int step, int items) {
                // completare
        }
        public boolean hasNext() {
                // completare
        }
        public Integer next() {
                // completare
        }
        public Iterator<Integer> iterator() { // restituisce se stesso
                // completare
        }
}
```

4. Considerare le seguenti dichiarazioni di classi Java:

```
public class P {
    String m(Object... o) {
        return "P.m(Object...)";
    }
    String m(Object o) {
        return "P.m(Object)";
    }
}
public class H extends P {
    String m(Integer i) {
        return super.m(i) + " H.m(Integer)";
    }
    String m(Double d) {
        return super.m(d) + " H.m(Double)";
    }
}
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(42.0)`

(e) `p2.m(42.0)`

(f) `h.m(42.0)`