Linguaggi e Programmazione Orientata agli Oggetti

Prova scritta

a.a. 2018/2019

19 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("(\\s+|//.*)|([a-zA-Z][a-zA-Z0-9]*(\\.[a-zA-Z][a-zA-Z0-9]*)*)");
        Matcher m = regEx.matcher("java.util // example");
        m.lookingAt();
        assert m.group(2).equals("java.util");
        assert m.group(0) != null;
        m.region(m.end(), m.regionEnd());
        m.lookingAt();
        assert m.group(0) != null;
        assert m.group(1) != null;
        m.region(m.end(), m.regionEnd());
        m.lookingAt();
        assert m.group(1).equals("// example");
        assert m.group(2) != null;
    }
}
```

   (b) Mostrare che la seguente grammatica è ambigua.

```
Exp ::= size Exp | Exp + Exp | [ Exps ] | Id
Exps ::= Exp | Exp , Exps
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_prod : ('a -> int) -> 'a list -> int` la funzione così specificata:

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

   Esempi:

```
# gen_prod (fun x->x+2) []
- : int = 1
# gen_prod (fun x->x+2) [1]
- : int = 3
# gen_prod (fun x->x+2) [1;2]
- : int = 12
# gen_prod (fun x->x+2) [1;2;3]
- : int = 60
```

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

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

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

3. Completare la seguente classe di iteratori `Powers` per generare la sequenza di potenze di un numero intero in ordine crescente di esponente a partire da 1. Per esempio, il seguente codice

```
// genera la sequenza 3^1, 3^2, 3^3, 3^4
for (int n : new Powers(3, 4))
    System.out.println(n);
```

stampa la sequenza

```
3
9
27
81
```

```
import java.util.Iterator;

public class Powers implements Iterator<Integer>, Iterable<Integer> {

    private final int base; // base dell'esponente
    private int items; // numero di elementi ancora da generare
    private int next; // prossimo elemento da restituire

    // precondizione: items >= 0
    public Powers(int base, int items) {
        // completare
    }
    public boolean hasNext() {
        // completare
    }
    public Integer next() {
        // completare
    }
    // restituisce se stesso
    public Iterator<Integer> iterator() {
        // completare
    }
}
```

4. Considerare le seguenti dichiarazioni di classi Java:

```
public class P {
    String m(Number... o) {
        return "P.m(Number...)";
    }
    String m(Number o) {
        return "P.m(Number)";
    }
}
public class H extends P {
    String m(Short s) {
        return super.m(s) + " H.m(Short)";
    }
    String m(Float f) {
        return super.m(f) + " H.m(Float)";
    }
}
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((short) 42)`

(b) `p2.m((short) 42)`

(c) `h.m((short) 42)`

(d) `p.m(42.0f)`

(e) `p2.m(42.0f)`

(f) `h.m(42.0f)`