Linguaggi e Programmazione Orientata agli Oggetti

Prova scritta

a.a. 2018/2019

10 luglio 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("([0-9]+)|(\\s+)|([a-zA-Z][a-zA-Z]*(\\.[a-zA-Z][a-zA-Z]*)*)");
        Matcher m = regEx.matcher("001 a.b");
        m.lookingAt();
        assert m.group(1).equals("001");
        assert m.group(0).equals("001 a.b");
        m.region(m.end(), m.regionEnd());
        m.lookingAt();
        assert m.group(2) != null;
        assert m.group(0).equals(" a.b");
        m.region(m.end(), m.regionEnd());
        m.lookingAt();
        assert m.group(3).equals("a.b");
        assert m.group(0).equals("a.b");
    }
}
```

   (b) Mostrare che la seguente grammatica è ambigua.

```
Exp ::= Exp ∧ Exp | Exp ∨ Exp | { ExpSeq } | Id
ExpSeq ::= Exp | Exp ExpSeq
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_cat : ('a -> string) -> 'a list -> string` la funzione così specificata, dove ^ è l'operatore di concatenazione di stringhe:

   `gen_cat` f `[`s_1`;` s_2`; ... ;` s_n`]` $= f(s_1)\hat{}f(s_2)\hat{}\ldots\hat{}f(s_n)$, con $n \geq 0$.

   Esempio:

```
# gen_cat (fun s -> s^"__") ["one";"two";"three"];;
- : string = "one__two__three__"
```

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

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

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

3. (a) Completare le seguenti classi che implementano i nodi di un AST per espressioni con literal interi positivi e operazione di elevamento a potenza.

```
public interface Visitor<T> {
    T visitNatLit(int val);
    T visitPow(Exp left, Exp right);
}
public interface Exp { <T> T accept(Visitor<T> visitor); }

// nodi AST per literal interi positivi
public class PosLit implements Exp {
    private final int val;

    // precondizione: val > 0
    public PosLit(int val) { /* completare */ }
    public <T> T accept(Visitor<T> visitor) { /* completare */ }
}
// nodi AST per elevamento a potenza
public class Pow implements Exp {
    private final Exp left; // base
    private final Exp right; // potenza

    // precondizione: left!=null, right!=null
    public Pow(Exp left, Exp right) { /* completare */ }
    public <T> T accept(Visitor<T> visitor) { /* completare */ }
}
```

   (b) Completare la classe `Eval` per la valutazione degli AST; la classe `Test` contiene una prova esplicativa.

```
public class Eval implements Visitor<Integer> {
    public Integer visitNatLit(int val) { /* completare */ }
    public Integer visitPow(Exp left, Exp right) { /* completare */ }
}
public class Test {
    public static void main(String[] args) {
        Exp e = new Pow(new PosLit(3), new PosLit(4)); // AST per 3 elevato 4
        assert e.accept(new Eval()) == 81;
    }
}
```

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(Integer i) { return super.m(i) + " H.m(Integer)"; }
    String m(Integer i1, Integer i2) { return super.m(i1, i2) + " H.m(Integer,Integer)"; }
}
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, 42)`

(e) `p2.m(42, 42)`

(f) `h.m(42, 42)`