Linguaggi e Programmazione Orientata agli Oggetti

Prova scritta

a.a. 2015/2016

11 febbraio 2016

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-Z][A-Z_]*)|(([0-9]+[\\.0-9]*)[fF])|<=|<|(\\s+)");
        Matcher m = regEx.matcher("A_B< =3.14F");
        m.lookingAt();
        assert m.group(1).equals("A_B");
        m.region(m.end(), m.regionEnd());
        assert m.lookingAt();
        assert m.group(2) == null;
        m.region(m.end(), m.regionEnd());
        m.lookingAt();
        m.region(m.end(), m.regionEnd());
        assert m.lookingAt();
        m.find();
        assert m.group(2).equals("3.14");
        assert Float.parseFloat(m.group(3)) == Float.parseFloat("3.14");
    }
}
```

   (b) Mostrare che la seguente grammatica è ambigua.

```
Exp  ::=  Exp ? Exp : Exp | - Exp | ( Exp ) | Id
Id   ::= x | y | z
```

   (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. Considerare la funzione `limit : int -> 'a list -> 'a list` così definita:
`limit` n $[e_1;e_2;\ldots;e_k]$ restituisce

   - $[e_1;e_2;\ldots;e_k]$ se $n \geq k$;
   - $[e_1;e_2;\ldots;e_n]$ se $0 < n < k$;
   - `[]` se $n \leq 0$.

   Esempi:

```
# limit (-1) [1;2];;
- : int list = []
# limit 0 [1;2];;
- : int list = []
# limit 1 [1;2];;
- : int list = [1]
# limit 2 [1;2];;
- : int list = [1; 2]
# limit 3 [1;2];;
- : int list = [1; 2]
```

   (a) Definire la funzione `limit` direttamente, senza uso di parametri di accumulazione.

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

3. Considerare la seguente implementazione della sintassi astratta di un semplice linguaggio che include il test di uguaglianza, le espressioni condizionali e gli identificatori di variabile.

```
public interface AST { <T> T accept(Visitor<T> visitor); }
public interface Variable extends AST {
    boolean equals(Object obj);
    int hashCode();
}
public interface Visitor<T> {
    T visitEq(AST left, AST right);
    T visitSimpleVar(SimpleVar var);
    T visitIfThenElse(AST exp, AST thenStmt, AST elseStmt);
}
public class Eq implements AST {
    private final AST left; // obbligatorio
    private final AST right; // obbligatorio
    public Eq(AST left, AST right) { /* da completare */}
    public <T> T accept(Visitor<T> v) { return v.visitEq(left, right); }
}
public class IfThenElse implements AST {
    private final AST exp; // obbligatorio
    private final AST thenStmt; // obbligatorio
    private final AST elseStmt; // opzionale
    public IfThenElse(AST exp, AST thenStmt, AST elseStmt) { /* da completare */ }
    public IfThenElse(AST exp, AST thenStmt) { /* da completare */ }
    public <T> T accept(Visitor<T> visitor) { return visitor.visitIfThenElse(exp, thenStmt, elseStmt); }
}
public class SimpleVar implements Variable {
    private final String name; // obbligatorio, non vuoto
    public SimpleVar(String name) { /* da completare */ }
    public <T> T accept(Visitor<T> visitor) { return visitor.visitSimpleVar(this); }
    public final boolean equals(Object obj) { /* da completare */ }
    public int hashCode() { /* da completare */ }
}
```

(a) Completare le definizioni dei costruttori di tutte le classi.

(b) Completare le definizioni dei metodi `equals(Object obj)` e `hashCode()` della classe `SimpleVar`.

(c) Completare la classe `GetFreeVars` che permette di restituire l'insieme degli identificatori di variabile contenuti in un AST. Per esempio, per l'AST che rappresenta l'espressione **`if`** `x==y` **`then`** `z` **`else`** `y`, il risultato della visita è l'insieme {`x`,`y`,`z`}.

```
public class GetFreeVars implements Visitor<Set<Variable>> {
    private final Set<Variable> vars = new HashSet<>();
    public Set<Variable> visitEq(AST left, AST right) { /* da completare */ }
    public Set<Variable> visitSimpleVar(SimpleVar var) { /* da completare */ }
    public Set<Variable> visitIfThenElse(AST exp, AST thenStmt, AST elseStmt) {
        /* da completare */
    }
}
```

(d) Completare la classe `ApplySubs` che permette di applicare una sostituzione a un AST. Una sostituzione è una mappa finita da variabili ad AST; per esempio, la sostituzione $\sigma = \{x \mapsto w, z \mapsto z==y, u \mapsto s\}$, sostituisce simultaneamente ogni occorrenza di `x` con `w`, di `z` con `z==y` e di `u` con `s`, mentre lascia invariate le occorrenze di tutte le variabili diverse da `x`, `z` e `u`. Quindi, l'applicazione di σ all'AST dell'espressione **`if`** `x==y` **`then`** `z` **`else`** `y` restituisce l'AST dell'espressione **`if`** `w==y` **`then`** `z==y` **`else`** `y`.

```
public interface Subst {
    /** restituisce la sostituzione per var; restituisce var se non esiste sostituzione per var */
    AST apply(Variable var);
    /** aggiorna la sostituzione */
    void update(Variable var, AST value);
}
public class ApplySubs implements Visitor<AST> {
    private final Subst subst; // obbligatorio
    public ApplySubs(Subst subst) {
        this.subst = requireNonNull(subst);
    }
    public AST visitEq(AST left, AST right) { /* da completare */ }
    public AST visitSimpleVar(SimpleVar var) { /* da completare */ }
    public AST visitIfThenElse(AST exp, AST thenStmt, AST elseStmt) {
        /* da completare */
    }
}
```

4. Considerare le seguenti dichiarazioni di classi Java:

```
public class P {
    String m(Integer i) { return "P.m(Integer)"; }
    String m(Long l) { return "P.m(Long)"; }
}
public class H extends P {
    String m(int i) { return super.m(i) + " H.m(int)"; }
    String m(Integer i) { return super.m(i) + " H.m(Integer)"; }
    String m(Object o) { return super.m((Integer) o) + " H.m(Object)"; }
}
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) `p.m(new Long(42))`

(d) `h.m(42L)`

(e) `h.m((byte)42)`

(f) `h.m(42.0)`