Linguaggi e Programmazione Orientata agli Oggetti

Prova scritta

a.a. 2012/2013

12 luglio 2013

1. (a) Data la seguente linea di codice Java

```
Pattern p = Pattern.compile("[0-9]+\\.?[0-9]*([eE][\\+-]?[0-9]+)?");
```

Indicare quali delle seguenti asserzioni falliscono, motivando la risposta.

i. `assert assert p.matcher("42").matches();`

ii. `assert assert p.matcher("42.").matches();`

iii. `assert assert p.matcher("42.42").matches();`

iv. `assert assert !p.matcher("42eE42").matches();`

v. `assert assert !p.matcher("42E.-42").matches();`

vi. `assert assert p.matcher("42E2").matches();`

(b) Mostrare che la seguente grammatica è ambigua.

```
Exp  ::= Id | Exp Exp | Exp + Exp | + Exp
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` **rimanga invariato**.

2. Considerare la funzione `swap : 'a list -> 'a list` così definita: `swap` l scambia l'elemento x di posizione i con l'elemento y di posizione $i+1$ se $x > y$, per ogni i che va da 0 ad $n-2$ (dove n è la lunghezza di l).

Esempi:

```
# swap [4;4;3;3;6;6;5;5];;
- : int list = [4; 3; 3; 4; 6; 5; 5; 6]
# swap [4;3;2;1;6;5;8;7];;
- : int list = [3; 2; 1; 4; 5; 6; 7; 8]
# swap [3; 2; 1; 4; 5; 6; 7; 8];;
- : int list = [2; 1; 3; 4; 5; 6; 7; 8]
# swap [2; 1; 3; 4; 5; 6; 7; 8];;
- : int list = [1; 2; 3; 4; 5; 6; 7; 8]
# swap [1; 2; 3; 4; 5; 6; 7; 8];;
- : int list = [1; 2; 3; 4; 5; 6; 7; 8]
```

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

(b) Definire la funzione `swap` direttamente, usando un parametro di accumulazione in modo che la ricorsione sia di coda.

(c) Definire la funzione `swap_no_dup` direttamente, senza uso di parametri di accumulazione e così definita: `swap_no_dup` l scambia l'elemento x di posizione i con l'elemento y di posizione $i+1$ se $x > y$, e cancella l'elemento x di posizione i se l'elemento y di posizione $i+1$ è tale che $x = y$, per ogni i che va da 0 ad $n-2$ (dove n è la lunghezza di l).

Esempi:

```
# swap_no_dup [4;4;3;3;6;6;5;5;4];;
- : int list = [3; 3; 4; 5; 5; 4; 6]
# swap_no_dup [3; 3; 4; 5; 5; 4; 6];;
- : int list = [3; 4; 4; 5; 6]
# swap_no_dup [3; 4; 4; 5; 6];;
- : int list = [3; 4; 5; 6]
```

3. Considerare le seguenti dichiarazioni di classi Java:

```
public class P {
        public String m(Number n) {
                return "P.m(Number)";
        }
        public String m(Double d) {
                return "P.m(Double)";
        }
        public String m(int i) {
                return "P.m(int)";
        }
}
public class H extends P {
        public String m(Number n) {
                return "H.m(Number) " + super.m(n);
        }
        public String m(double d) {
                return "H.m(double) " + super.m(d);
        }
        public String m(int i) {
                return "H.m(int) " + super.m(i);
        }
}

public class Test {
         public static void main(String[] args) {
                 H h = new H();
                 P p = h;
                 System.out.println(...);
         }

}
```

Dire, per ognuno dei casi sotto elencati, 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(1)`

(b) `p.m((Integer) 1)`

(c) `h.m((Integer) 1)`

(d) `h.m(4.2)`

(e) `h.m((Double) 4.2)`

(f) `h.m(1,2)`

4. Considerare i package `ast` e `visitor` che implementano abstract syntax tree e visite su di essi per espressioni aritmetiche formate a partire da variabili, literal interi, l'operatore binario di addizione e quello unario di sottrazione.

```
package ast;
import visitor.Visitor;
public interface Exp {
        Iterable<Exp> getChildren();
        void accept(Visitor v);
}
```

```
package ast;
public interface Variable extends Exp {
        String getName();
}
```

```
package ast;
import static java.util.Arrays.asList;
public abstract class AbsExp implements Exp {
        private final Iterable<Exp> children;
        protected AbsExp(Exp... children) {
                // completare
        }
        @Override
        public Iterable<Exp> getChildren() {
                // completare
        }
}
```

```
package ast;
import visitor.Visitor;
public class IdentExp extends AbsExp implements Variable {
        private final String name;
        public IdentExp(String name) {
                this.name = name;
        }
        public String getName() {
                return name;
        }
        public void accept(Visitor v) {
                // completare
        }
}
```

```
package ast;
import visitor.Visitor;
public class NumLit extends AbsExp {
        final private int value;
        public NumLit(int value) {
                this.value = value;
        }
        public int getValue() {
                return value;
        }
        public void accept(Visitor v) {
                // completare
        }
}
```

```
package ast;
import visitor.Visitor;
public class AddExp extends AbsExp {
        public AddExp(Exp exp1, Exp exp2) {
                // completare
        }
        public void accept(Visitor v) {
                // completare
        }
}
```

```
package ast;
import visitor.Visitor;
public class MinusExp extends AbsExp {
        public MinusExp(Exp exp) {
                // completare
        }
        public void accept(Visitor v) {
                // completare
        }
}
```

(a) Completare le definizioni delle classi `AbsExp`, `IdentExp`, `NumLit`, `AddExp` e `MinusExp`.

(b) Date le seguenti dichiarazioni di classe e interfaccia, completare la definizione delle classi `ContainVarVisitor` e `PrefixVisitor`.

```
package visitor;
import ast.*;
public interface Visitor {
        void visit(IdentExp e);
        void visit(NumLit e);
        void visit(AddExp e);
        void visit(MinusExp e);
}
```

```
package visitor;
public abstract class AbstractVisitor<T> implements Visitor {
        protected T result;
        public T getResult() {
                return result;
        }
}
```

```
package visitor;
import java.util.Iterator;
import ast.*;
public class ContainVarVisitor extends AbstractVisitor<Boolean> {
        private Variable var;
        public void setVar(Variable var) {
                this.var = var;
        }
        public ContainVarVisitor(Variable var) {
                this.var = var;
        }
        @Override
        public void visit(AddExp exp) {
                // completare
        }

        @Override
        public void visit(IdentExp exp) {
                // completare
        }

        @Override
        public void visit(MinusExp exp) {
                // completare
        }

        @Override
        public void visit(NumLit exp) {
                // completare
        }
}
```

```
package visitor;
import java.util.Iterator;
import ast.*;
public class PrefixVisitor extends AbstractVisitor<String> {
        public void visit(AddExp exp) {
                // completare
        }
        public void visit(IdentExp exp) {
                // completare
        }
        public void visit(MinusExp exp) {
                // completare
        }
        public void visit(NumLit exp) {
                // completare
        }
}
```

- la classe `ContainVarVisitor` controlla se l'espressione visitata contiene una data variabile.
  Esempio:

```
Exp exp = new AddExp(new AddExp(new NumLit(0), new IdentExp("z")),
        new MinusExp(new AddExp(new IdentExp("x"), new IdentExp("y"))));
ContainVarVisitor cv = new ContainVarVisitor(new IdentExp("x"));
exp.accept(cv);
assert cv.getResult();
cv.setVar(new IdentExp("z"));
exp.accept(cv);
assert cv.getResult();
cv.setVar(new IdentExp("w"));
exp.accept(cv);
assert !cv.getResult();
```

- la classe `PrefixVisitor` genera la stringa corrispondente alla forma polacca prefissa della espressione visitata.
  Esempio:

```
Exp exp = new AddExp(new AddExp(new NumLit(0), new IdentExp("z")),
        new MinusExp(new AddExp(new IdentExp("x"), new IdentExp("y"))));
PrefixVisitor pv = new PrefixVisitor();
exp.accept(pv);
assert pv.getResult().equals("+ + 0 z - + x y");
```