INF2100

Løsningsforslag til oppgaver uke 40 og 41 2014

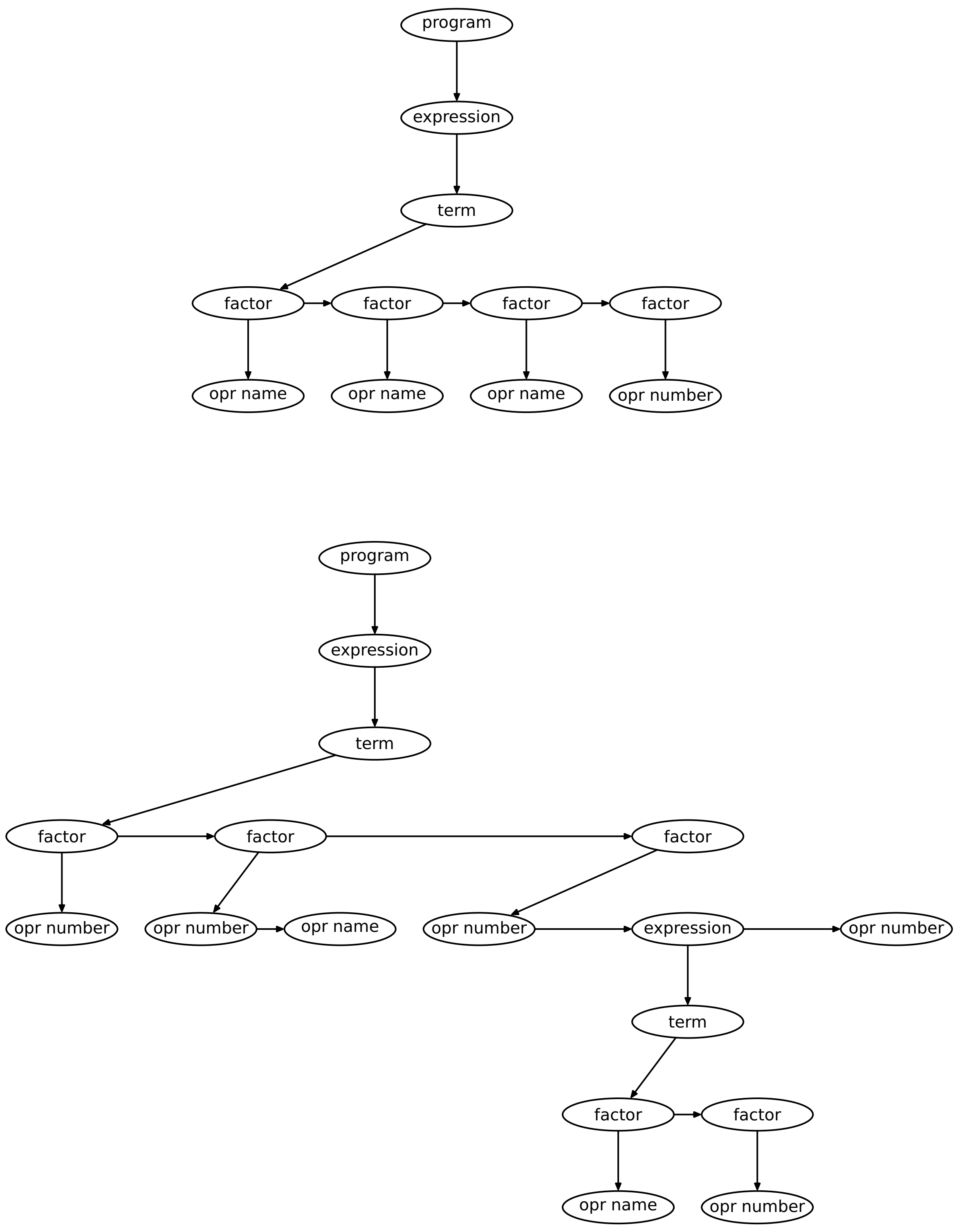
Oppgave 1

Se figur 1 på neste side.

Oppgave 2–4

Her er det mange mulige løsninger — her er én:

```
import java.io.*;
import java.util.*;

class E {
    public static void main(String arg[]) {
        Scanner.init();

        Program p = Program.parse();
        if (Scanner.curToken != Token.eofToken)
            Error.error("Syntax error: Illegal "+Scanner.curToken);
        p.printTree();  Log.writeln();

        System.out.println("The value is "+p.eval());
    }
}

abstract class SyntaxUnit {
    abstract long eval();
    abstract void printTree();
}

class Program extends SyntaxUnit {
    Expression e;

    @Override long eval() {
        return e.eval();
    }

    static Program parse() {
        Program p = new Program();

        Log.enterParser("<program>");
        p.e = Expression.parse();
        Log.leaveParser("</program>");
        return p;
    }

    @Override void printTree() {
        e.printTree();
    }
}
```



Figur 1: Svar på oppgave 1

```
class Expression extends SyntaxUnit {
    Term t = new Term();

    @Override long eval() {
        return t.eval();
    }

    static Expression parse() {
        Expression e = new Expression();

        Log.enterParser("<expression>");
        e.t = Term.parse();
        Log.leaveParser("</expression>");
        return e;
    }

    @Override void printTree() {
        t.printTree();
    }
}

class Term extends SyntaxUnit {
    List<Factor> factors = new ArrayList<Factor>();
    List<Token> opers = new ArrayList<Token>();

    @Override long eval() {
        long v = factors.get(0).eval();

        for (int i = 1;  i < factors.size();  ++i) {
            if (opers.get(i-1) == Token.plusToken)
                v += factors.get(i).eval();
            else
                v -= factors.get(i).eval();
        }
        return v;
    }

    static Term parse() {
        Term t = new Term();

        Log.enterParser("<term>");
        t.factors.add(Factor.parse());
        while (Scanner.curToken==Token.plusToken ||
               Scanner.curToken==Token.minusToken) {
            t.opers.add(Scanner.curToken);  Scanner.readNext();
            t.factors.add(Factor.parse());
        }
        Log.leaveParser("</term>");
        return t;
    }

    @Override void printTree() {
        factors.get(0).printTree();
        for (int i = 1;  i < factors.size();  ++i) {
            if (opers.get(i-1) == Token.plusToken)
                Log.write("+");
            else
                Log.write("-");
            factors.get(i).printTree();
        }
    }
}

class Factor extends SyntaxUnit {
    List<Operand> operands = new ArrayList<Operand>();
    List<Token> opers = new ArrayList<Token>();

    @Override long eval() {
        long v = operands.get(0).eval();

        for (int i = 1;  i < operands.size();  ++i) {
            if (opers.get(i-1) == Token.mulToken) {
                v *= operands.get(i).eval();
            } else {
                long dv = operands.get(i).eval();
                if (dv == 0)
                    Error.error("Division by zero!");
                v /= dv;
            }
        }
        return v;
    }

    static Factor parse() {
        Factor f = new Factor();

        Log.enterParser("<factor>");
        f.operands.add(Operand.parse());
        while (Scanner.curToken==Token.mulToken ||
               Scanner.curToken==Token.divToken) {
            f.opers.add(Scanner.curToken);  Scanner.readNext();
            f.operands.add(Operand.parse());
        }
        Log.leaveParser("</factor>");
        return f;
    }

    @Override void printTree() {
        operands.get(0).printTree();
        for (int i = 1;  i < operands.size();  ++i) {
            if (opers.get(i-1) == Token.mulToken)
                Log.write("*");
            else
                Log.write("/");
            operands.get(i).printTree();
        }
    }
}

abstract class Operand extends SyntaxUnit {
    static Operand parse() {
        if (Scanner.curToken == Token.nameToken)
            return OperandName.parse();
        if (Scanner.curToken == Token.numberToken)
            return OperandNumber.parse();
        if (Scanner.curToken == Token.leftParToken)
            return OperandExpr.parse();
        Error.error("Syntax error; found a "+Scanner.curToken);
        return null;  // Required by the compiler.
    }
}
```

```
class OperandName extends Operand {
    char id;

    @Override long eval() {
        if (id == 'M') return 1000;
        if (id == 'C') return 100;
        if (id == 'X') return 10;
        Error.error("Unknown name: "+id+".");
        return 0;  // Required by the compiler!
    }

    static OperandName parse() {
        OperandName o = new OperandName();

        Log.enterParser("<operand> (a name)");
        o.id = Scanner.curName;  Scanner.readNext();
        Log.leaveParser("</operand>");
        return o;
    }

    @Override void printTree() {
        Log.write(""+id);
    }
}

class OperandNumber extends Operand {
    long n;

    @Override long eval() {
        return n;
    }

    static OperandNumber parse() {
        OperandNumber o = new OperandNumber();

        Log.enterParser("<operand> (a number)");
        o.n = Scanner.curNumber;  Scanner.readNext();
        Log.leaveParser("</operand>");
        return o;
    }

    @Override void printTree() {
        Log.write(""+n);
    }
}

class OperandExpr extends Operand {
    Expression e;

    @Override long eval() {
        return e.eval();
    }

    static OperandExpr parse() {
        OperandExpr e = new OperandExpr();

        Log.enterParser("<operand> (an inner expression)");
        Scanner.readNext();
        e.e = Expression.parse();
        if (Scanner.curToken != Token.rightParToken)
```

```
            Error.error("A ) expected after an expression.");
        Scanner.readNext();
        Log.leaveParser("</operand>");
        return e;
    }

    @Override void printTree() {
        Log.write("(");  e.printTree();  Log.write(")");
    }
}

enum Token { nameToken, numberToken, plusToken, minusToken, mulToken,
        divToken, leftParToken, rightParToken, eofToken }

class Scanner {
    public static Token curToken;
    public static char curName;
    public static int curNumber;

    private static LineNumberReader f;

    public static void init() {
        f = new LineNumberReader(new InputStreamReader(System.in));
        readNext();
    }

    public static void readNext() {
        curToken = null;
        while (curToken == null) {
            int c = '?';
            try {
                c = f.read(); // Read one character
            } catch (IOException e) {
                Error.error("Read error!");
            }

            if (c < 0) {
                curToken = Token.eofToken;
            } else if (c == '+') {
                curToken = Token.plusToken;
            } else if (c == '-') {
                curToken = Token.minusToken;
            } else if (c == '*') {
                curToken = Token.mulToken;
            } else if (c == '/') {
                curToken = Token.divToken;
            } else if (c == '(') {
                curToken = Token.leftParToken;
            } else if (c == ')') {
                curToken = Token.rightParToken;
            } else if ('A'<=c && c<='Z' || 'a'<=c && c<='z') {
                curToken = Token.nameToken;  curName = (char)c;
            } else if (Character.isDigit(c)) {
                curToken = Token.numberToken;  curNumber = c-'0';
            } else if (Character.isWhitespace(c)) {
                // Ignore space
            } else {
                Error.error("Illegal character: '"+(char)c+"'!");
            }
        }
        // For testing:
```

```
        // System.out.println("Scanner: Read a "+curToken);
    }
}

class Error {
    static void error(String message) {
        System.err.println("ERROR: "+message);
        System.exit(1);
    }
}

class Log {
    public static boolean doLogParser = true, doLogTree = true;
    private static int parseLevel = 0;

    public static void enterParser(String symbol) {
        if (! doLogParser) return;
        for (int i = 1;  i <= parseLevel;  ++i)
            System.out.print("  ");
        System.out.println(symbol);
        ++parseLevel;
    }

    public static void leaveParser(String symbol) {
        if (! doLogParser) return;
        --parseLevel;
        for (int i = 1;  i <= parseLevel;  ++i)
            System.out.print("  ");
        System.out.println(symbol);
    }

    public static void write(String s) {
        if (! doLogTree) return;
        System.out.print(s+" ");
    }

    public static void writeln() {
        if (! doLogTree) return;
        System.out.println();
    }
}
```

Tilleggsspørsmål 2a

Metodene eval og printTree vil ikke lenger være virtuelle, så vi ville få feilmeldinger for alle @Override-ene våre.

Om vi også fjernet alle @Override, ville vi stadig få feilmeldinger, for eksempel om vi utfører i Factor.eval: operands.get(i).eval() siden kompilatoren ikke kunne vite at Operand inneholder en eval-metode.