IN2030

Løsningsforslag

Uke 41 2023

Oppgave 1

(Disse filene finnes også i mappen ~`inf2100/e/e3/`.)

```java
class E {
    public static void main(String arg[]) {
        Scanner s = new Scanner(arg[0]);
        Expression e = Expression.parse(s);
        e.prettyPrint();   System.out.println();
        System.out.println("Resultatet␣er:␣" + e.eval());
    }
}
```

```java
abstract class ESyntax {
    abstract int eval();
    abstract void prettyPrint();

    static int parseLevel = 0;

    static void enterParser(String nonterm) {
        for (int i = 1;  i <= parseLevel;  ++i)
            System.out.print("␣␣");
        System.out.println("<"+nonterm+">");
        parseLevel++;
    }

    static void leaveParser(String nonterm) {
        parseLevel--;
        for (int i = 1;  i <= parseLevel;  ++i)
            System.out.print("␣␣");
        System.out.println("</"+nonterm+">");
    }
}
```

```java
abstract class Expression extends ESyntax {
    static Expression parse(Scanner s) {
        enterParser("expression");

        Expression e = Term.parse(s);

        leaveParser("expression");
        return e;
    }
}
```

```java
import java.util.ArrayList;

class Term extends Expression {
    ArrayList<Factor> operands = new ArrayList<>();
    ArrayList<Token> oprs = new ArrayList<>();

    static Term parse(Scanner s) {
        enterParser("term");

        Term t = new Term();
        t.operands.add(Factor.parse(s));
        while (s.curToken().kind == TokenKind.addToken ||
               s.curToken().kind == TokenKind.subtractToken) {
            t.oprs.add(s.curToken());
            s.readNextToken();
            t.operands.add(Factor.parse(s));
        }

        leaveParser("term");
        return t;
    }

    @Override
    int eval() {
        int v = operands.get(0).eval();
        for (int i = 1;  i < operands.size();  i++) {
            int v2 = operands.get(i).eval();
            switch (oprs.get(i-1).kind) {
            case addToken:
                v = v + v2;  break;
            case subtractToken:
                v = v - v2;  break;
            }
        }
        return v;
    }

    @Override
    void prettyPrint() {
        operands.get(0).prettyPrint();
        for (int i = 1;  i < operands.size();  i++) {
            System.out.print("␣" + oprs.get(i-1).kind + "␣");
            operands.get(i).prettyPrint();
        }
    }
}
```

```java
import java.util.ArrayList;

class Factor extends ESyntax {
    ArrayList<Atom> operands = new ArrayList<>();
    ArrayList<Token> oprs = new ArrayList<>();

    static Factor parse(Scanner s) {
        enterParser("factor");

        Factor f = new Factor();
        f.operands.add(Atom.parse(s));
        while (s.curToken().kind == TokenKind.multiplyToken ||
               s.curToken().kind == TokenKind.divideToken) {
            f.oprs.add(s.curToken());
            s.readNextToken();
            f.operands.add(Atom.parse(s));
        }

        leaveParser("factor");
        return f;
    }

    @Override
    int eval() {
        int v = operands.get(0).eval();
        for (int i = 1;  i < operands.size();  i++) {
            int v2 = operands.get(i).eval();
            switch (oprs.get(i-1).kind) {
            case multiplyToken:
                v = v * v2;  break;
            case divideToken:
                if (v2 == 0) {
                    System.out.println("FEIL:␣Ulovlig␣med␣deling␣med␣0!");
                    System.exit(1);
                }
                v = v / v2;  break;
            }
        }
        return v;
    }

    @Override
    void prettyPrint() {
        operands.get(0).prettyPrint();
        for (int i = 1;  i < operands.size();  i++) {
            System.out.print("␣" + oprs.get(i-1).kind + "␣");
            operands.get(i).prettyPrint();
        }
    }
}
```

```java
abstract class Atom extends ESyntax {
    static Atom parse(Scanner s) {
        enterParser("atom");

        Atom a;
        if (s.curToken().kind == TokenKind.leftParToken)
            a = InnerExpr.parse(s);
        else
            a = Number.parse(s);

        leaveParser("atom");
        return a;
    }
}
```

```java
class InnerExpr extends Atom {
    Expression expr;

    static InnerExpr parse(Scanner s) {
        enterParser("inner␣expr");

        InnerExpr ie = new InnerExpr();
        s.readNextToken();  // Skip past '('.
        ie.expr = Expression.parse(s);
        s.readNextToken();  // Skip past ')'.

        leaveParser("inner␣expr");
        return ie;
    }

    @Override
    int eval() {
        return expr.eval();
    }

    @Override
    void prettyPrint() {
        System.out.print("(");  expr.prettyPrint();
        System.out.print(")");
    }
}
```

```
class Number extends Atom {
    int val;

    static Number parse(Scanner s) {
        enterParser("number");

        Number n = new Number();
        n.val = s.curToken().numVal;
        s.readNextToken();

        leaveParser("number");
        return n;
    }

    @Override
    int eval() {
        return val;
    }

    @Override
    void prettyPrint() {
        System.out.print(val);
    }
}
```

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

class Scanner {
    private LineNumberReader sourceFile = null;
    private String curFileName;
    private ArrayList<Token> curLineTokens = new ArrayList<>();

    Scanner(String fileName) {
        curFileName = fileName;
        try {
            sourceFile = new LineNumberReader(
                            new InputStreamReader(
                                new FileInputStream(fileName),
                                "UTF-8"));
        } catch (IOException e) {}
    }

    public Token curToken() {
        while (curLineTokens.isEmpty()) {
            readNextLine();
        }
        return curLineTokens.get(0);
    }

    void readNextToken() {
        if (! curLineTokens.isEmpty())
            curLineTokens.remove(0);
    }

    private void readNextLine() {
        curLineTokens.clear();

        // Read the next line:
        String line = null;
        try {
            line = sourceFile.readLine();
```

```java
            if (line == null) {
                sourceFile.close();   sourceFile = null;
                line = "";
            }
        } catch (IOException e) {}

        // Were there any more lines to read?
        if (sourceFile == null) {
            curLineTokens.add(new Token(TokenKind.eofToken));
        }

        // Find all the tokens:
        int pos = 0;
        while (pos < line.length()) {
            char c = line.charAt(pos++);

            if (isDigit(c)) {
                curLineTokens.add(new Token(Integer.parseInt(""+c)));
            } else if (c == '+') {
                curLineTokens.add(new Token(TokenKind.addToken));
            } else if (c == '-') {
                curLineTokens.add(new Token(TokenKind.subtractToken));
            } else if (c == '*') {
                curLineTokens.add(new Token(TokenKind.multiplyToken));
            } if (c == '/') {
                curLineTokens.add(new Token(TokenKind.divideToken));
            } else if (c == '(') {
                curLineTokens.add(new Token(TokenKind.leftParToken));
            } else if (c == ')') {
                curLineTokens.add(new Token(TokenKind.rightParToken));
            }
        }
        for (Token t: curLineTokens)
            System.out.println("E␣scanner:␣Read␣a␣" + t);
    }

    private boolean isDigit(char c) {
        return '0'<=c && c<='9';
    }
}
```

```java
class Token {
    TokenKind kind;
    int numVal;

    Token(TokenKind k) {
        kind = k;
    }

    Token(int nVal) {
        kind = TokenKind.numberToken;   numVal = nVal;
    }

    public String toString() {
        String s = kind.toString();
        if (kind == TokenKind.numberToken) s += ":"+numVal;
        return s;
    }
}
```

```java
enum TokenKind {
    numberToken("number"),
    addToken("+"),
    subtractToken("-"),
    multiplyToken("*"),
    divideToken("/"),
    leftParToken("("),
    rightParToken(")"),
    eofToken("e-o-f");


    private String image;

    TokenKind(String s) {
        image = s;
    }

    public String toString() {
        return image;
    }
}
```