INF2030

Løsningsforslag

Uke 38 2022

Oppgave 1

(Denne koden finnes også i ~inf2100/e/e4/.)

E.java

```
class E {
    public static void main(String arg[]) {
        Scanner s = new Scanner(arg[0]);
        Expression e = Expression.parse(s);
    }
}
```

ESyntax.java

```
abstract class ESyntax {

}
```

Expression.java

```
abstract class Expression extends ESyntax {
    static Expression parse(Scanner s) {
        Expression e = Term.parse(s);
        return e;
    }
}
```

Term.java

```
import java.util.ArrayList;

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

    static Term parse(Scanner s) {
        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));
        }
        return t;
    }
}
```

Factor.java

```
import java.util.ArrayList;

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

    static Factor parse(Scanner s) {
        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));
        }
        return f;
    }
}
```

Atom.java

```
abstract class Atom extends ESyntax {
    static Atom parse(Scanner s) {
        Atom a;
        if (s.curToken().kind == TokenKind.leftParToken)
            a = InnerExpr.parse(s);
        else
            a = Number.parse(s);
        return a;
    }
}
```

Number.java

```
class Number extends Atom {
    int val;

    static Number parse(Scanner s) {
        Number n = new Number();
        n.val = s.curToken().numVal;
        s.readNextToken();
        return n;
    }
}
```

InnerExpr.java

```
class InnerExpr extends Atom {
    Expression expr;

    static InnerExpr parse(Scanner s) {
        InnerExpr ie = new InnerExpr();
        s.readNextToken();  // Skip past '('.
        ie.expr = Expression.parse(s);
        s.readNextToken();  // Skip past ')'.
        return ie;
    }
}
```

Scanner.java

```
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();
            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';
    }
}
```

Token.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;
    }
}
```

TokenKind.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;
    }
}
```