IN2030

Løsningsforslag

Uke 39 2020

Oppgave 1

(Disse filene finnes også i mappen ~inf2100/e/e2/.)

```
1  class E {
2      public static void main(String arg[]) {
3          Scanner s = new Scanner(arg[0]);
4          Expression e = Expression.parse(s);
5          e.prettyPrint();  System.out.println();
6      }
7  }
```

```
1  abstract class ESyntax {
2      abstract void prettyPrint();
3
4      static int parseLevel = 0;
5
6      static void enterParser(String nonterm) {
7          for (int i = 1;  i <= parseLevel;  ++i)
8              System.out.print("  ");
9          System.out.println("<"+nonterm+">");
10         parseLevel++;
11     }
12
13     static void leaveParser(String nonterm) {
14         parseLevel--;
15         for (int i = 1;  i <= parseLevel;  ++i)
16             System.out.print("  ");
17         System.out.println("</"+nonterm+">");
18     }
19 }
```

```
1  abstract class Expression extends ESyntax {
2      static Expression parse(Scanner s) {
3          enterParser("expression");
4
5          Expression e = Term.parse(s);
6
7          leaveParser("expression");
8          return e;
9      }
10 }
```

```
1  import java.util.ArrayList;
2
3  class Term extends Expression {
4      ArrayList<Factor> operands = new ArrayList<>();
5      ArrayList<Token> oprs = new ArrayList<>();
6
7      static Term parse(Scanner s) {
8          enterParser("term");
```

1

```
 9
10          Term  t  =  new  Term ( ) ;
11          t . operands . add ( Factor . parse ( s ) ) ;
12          while  ( s . curToken ( ) . kind  ==  TokenKind . addToken  | |
13                  s . curToken ( ) . kind  ==  TokenKind . subtractToken )  {
14              t . oprs . add ( s . curToken ( ) ) ;
15              s . readNextToken ( ) ;
16              t . operands . add ( Factor . parse ( s ) ) ;
17          }
18
19          leaveParser ( "term" ) ;
20          return  t ;
21      }
22
23      @Override
24      void  prettyPrint ( )  {
25          operands . get ( 0 ) . prettyPrint ( ) ;
26          for  ( int  i  =  1 ;    i  <  operands . size ( ) ;    i++)  {
27              System . out . print ( "␣" + oprs . get ( i −1). kind  +  "␣" ) ;
28              operands . get ( i ) . prettyPrint ( ) ;
29          }
30      }
31 }
```

```java
 1 import java . util . ArrayList ;
 2
 3 class Factor extends ESyntax {
 4      ArrayList<Atom> operands  =  new  ArrayList <>();
 5      ArrayList<Token> oprs  =  new  ArrayList <>();
 6
 7      static  Factor  parse ( Scanner  s )  {
 8          enterParser ( "factor" ) ;
 9
10          Factor  f  =  new  Factor ( ) ;
11          f . operands . add ( Atom . parse ( s ) ) ;
12          while  ( s . curToken ( ) . kind  ==  TokenKind . multiplyToken  | |
13                  s . curToken ( ) . kind  ==  TokenKind . divideToken )  {
14              f . oprs . add ( s . curToken ( ) ) ;
15              s . readNextToken ( ) ;
16              f . operands . add ( Atom . parse ( s ) ) ;
17          }
18
19          leaveParser ( "factor" ) ;
20          return  f ;
21      }
22
23      @Override
24      void  prettyPrint ( )  {
25          operands . get ( 0 ) . prettyPrint ( ) ;
26          for  ( int  i  =  1 ;    i  <  operands . size ( ) ;    i++)  {
27              System . out . print ( "␣" + oprs . get ( i −1). kind  +  "␣" ) ;
28              operands . get ( i ) . prettyPrint ( ) ;
29          }
30      }
31 }
```

```java
 1 abstract class Atom extends ESyntax {
 2      static  Atom  parse ( Scanner  s )  {
 3          enterParser ( "atom" ) ;
 4
 5          Atom  a ;
 6          if  ( s . curToken ( ) . kind  ==  TokenKind . leftParToken )
 7              a  =  InnerExpr . parse ( s ) ;
 8          else
 9              a  =  Number . parse ( s ) ;
```

```
10
11          leaveParser("atom");
12          return a;
13      }
14  }
```

```
1   class InnerExpr extends Atom {
2       Expression expr;
3
4       static InnerExpr parse(Scanner s) {
5           enterParser("inner_expr");
6
7           InnerExpr ie = new InnerExpr();
8           s.readNextToken();  // Skip past '('.
9           ie.expr = Expression.parse(s);
10          s.readNextToken();  // Skip past ')'.
11
12          leaveParser("inner_expr");
13          return ie;
14      }
15
16      @Override
17      void prettyPrint() {
18          System.out.print("(");  expr.prettyPrint();
19          System.out.print(")");
20      }
21  }
```

```
1   class Number extends Atom {
2       int val;
3
4       static Number parse(Scanner s) {
5           enterParser("number");
6
7           Number n = new Number();
8           n.val = s.curToken().numVal;
9           s.readNextToken();
10
11          leaveParser("number");
12          return n;
13      }
14
15      @Override
16      void prettyPrint() {
17          System.out.print(val);
18      }
19  }
```

```
1   import java.io.*;
2   import java.util.*;
3
4   class Scanner {
5       private LineNumberReader sourceFile = null;
6       private String curFileName;
7       private ArrayList<Token> curLineTokens = new ArrayList<>();
8
9       Scanner(String fileName) {
10          curFileName = fileName;
11          try {
12              sourceFile = new LineNumberReader(
13                              new InputStreamReader(
14                                  new FileInputStream(fileName),
15                                  "UTF-8"));
16          } catch (IOException e) {}
17      }
```

```
18
19    public Token curToken() {
20        while (curLineTokens.isEmpty()) {
21            readNextLine();
22        }
23        return curLineTokens.get(0);
24    }
25
26    void readNextToken() {
27        if (! curLineTokens.isEmpty())
28            curLineTokens.remove(0);
29    }
30
31    private void readNextLine() {
32        curLineTokens.clear();
33
34        // Read the next line:
35        String line = null;
36        try {
37            line = sourceFile.readLine();
38            if (line == null) {
39                sourceFile.close();   sourceFile = null;
40                line = "";
41            }
42        } catch (IOException e) {}
43
44        // Were there any more lines to read?
45        if (sourceFile == null) {
46            curLineTokens.add(new Token(TokenKind.eofToken));
47        }
48
49        // Find all the tokens:
50        int pos = 0;
51        while (pos < line.length()) {
52            char c = line.charAt(pos++);
53
54            if (isDigit(c)) {
55                curLineTokens.add(new Token(Integer.parseInt(""+c)));
56            } else if (c == '+') {
57                curLineTokens.add(new Token(TokenKind.addToken));
58            } else if (c == '-') {
59                curLineTokens.add(new Token(TokenKind.subtractToken));
60            } else if (c == '*') {
61                curLineTokens.add(new Token(TokenKind.multiplyToken));
62            } if (c == '/') {
63                curLineTokens.add(new Token(TokenKind.divideToken));
64            } else if (c == '(') {
65                curLineTokens.add(new Token(TokenKind.leftParToken));
66            } else if (c == ')') {
67                curLineTokens.add(new Token(TokenKind.rightParToken));
68            }
69        }
70        for (Token t: curLineTokens)
71            System.out.println("E scanner: Read a " + t);
72    }
73
74    private boolean isDigit(char c) {
75        return '0'<=c && c<='9';
76    }
77 }
```

```
1  class Token {
2      TokenKind kind;
3      int numVal;
4
5      Token(TokenKind k) {
```

```
 6            kind = k;
 7        }
 8
 9        Token(int nVal) {
10            kind = TokenKind.numberToken;   numVal = nVal;
11        }
12
13        public String toString() {
14            String s = kind.toString();
15            if (kind == TokenKind.numberToken) s += ":"+numVal;
16            return s;
17        }
18 }
```

```
 1 enum TokenKind {
 2        numberToken("number"),
 3        addToken("+"),
 4        subtractToken("-"),
 5        multiplyToken("*"),
 6        divideToken("/"),
 7        leftParToken("("),
 8        rightParToken(")"),
 9        eofToken("e-o-f");
10
11
12        private String image;
13
14        TokenKind(String s) {
15            image = s;
16        }
17
18        public String toString() {
19            return image;
20        }
21 }
```