Objectgeorienteerd Programmeren 25-06-2007

1 Opgave 2

*

Listing 1: Bag.

```
1  class Bag {
2      BagNode head;  // kop van de lijst
3
4      public Bag() { // default constructor
5    head = null;
6      }
7
8
9      public int size() {
10   return size(head);
11     }
12
13     private int size(BagNode t) {
14   int res = 0;
15   while (t != null) {
16       res += t.mult;
17       t = t.next;
18   }
19   return res;
20     }
21
22
23     public boolean contains(int v) {
24   return contains(head, v);
25     }
26
27     private boolean contains(BagNode t, int v) {
28   while (t != null) {
29       if (t.val == v) {
30      return true;
31       }
32       t = t.next;
33   }
34   return false;
35     }
36
37
38     public void add(int v) {
39   head = add(head, v);
40     }
41
42     private BagNode add(BagNode t, int v) {
43   if (t == null) {
44       return new BagNode(v);
45   } else if (t.val < v) {
```

```
46          t.next = add(t.next, v);
47      } else if (t.val == v) {
48          t.mult++;
49      } else { // t.val > v
50          BagNode u = new BagNode(v);
51          u.next = t;
52          return u;
53      }
54      return t;
55       }
56
57
58       public void remove(int v) {
59      head = remove(head, v);
60       }
61
62       private BagNode remove(BagNode t, int v) {
63      if (t == null) {
64          return null;
65      } else if (t.val < v) {
66          t.next = remove(t.next, v);
67      } else if (t.val == v) {
68          t.mult--;
69          if (t.mult == 0) {
70         return t.next;
71          }
72      } else { // t.val > v, v not present
73          // no action needed
74      }
75      return t;
76       }
77
78
79       public String toString() {
80      return toString(head);
81       }
82
83       private String toString(BagNode t) {
84      String res = "<";
85      while (t != null) {
86          res += t.toString();
87          if (t.next != null) {
88         res += ", ";
89          }
90          t = t.next;
91      }
92      res += ">";
93      return res;
94       }
95
96  }
```

Listing 2: BagNode.

```
1  class BagNode {
2      int val;       // de opgeslagen waarde
3      int mult;      // multipliciteit: aantal voorkomens van waarde val
4
5      BagNode next; // het volgende element in de bag
6
7      BagNode() {   // default constructor
8      val = 0;
9      mult = 1;
10     next = null;
11      }
12
```

```
13     BagNode(int v) {
14    this();
15    val = v;
16     }
17
18     public String toString() {
19    return "(" + val + "," + mult + ")";
20     }
21 }
```

Listing 3: Main.

```
1 import java.util.*;
2
3 public class Main {
4
5     public static void main(String[] args) {
6
7    try {
8        Scanner sc = new Scanner(System.in);
9
10       System.out.println();
11       System.out.println("Bag test programma");
12       System.out.println();
13
14       Bag b = new Bag();
15       while (true) {
16
17      System.out.println("Geef getal om in de bag te stoppen");
18      System.out.println("Geef een negatief getal ter beeindiging");
19      System.out.print("Geef getal: ");
20
21      int i = sc.nextInt();
22      if (i < 0) {
23          break;
24      }
25      b.add(i);
26
27      System.out.println();
28      System.out.println("De bag na toevoegen van " + i + " : " + b);
29      System.out.println("De bag bevat " + b.size() + " element" + (b.size() > 1 ? "en" : ""
            ));
30      System.out.println();
31       }
32
33       System.out.println();
34       System.out.println("Invoegen gestopt");
35       System.out.println();
36
37       while (b.size() > 0) {
38
39      System.out.println("Geef getal om uit de bag te halen");
40      System.out.println("Geef een negatief getal ter beeindiging");
41      System.out.print("Geef getal: ");
42
43      int i = sc.nextInt();
44      if (i < 0) {
45          break;
46      }
47      b.remove(i);
48
49      System.out.println();
50      System.out.println("De bag na verwijderen van " + i + " : " + b);
51      System.out.println("De bag bevat " + b.size() + " element" + (b.size() > 1 ? "en" : ""
            ));
52      System.out.println();
```

```
53          }
54
55      } catch (Exception e) {
56          System.out.println();
57          System.out.println("Er is een fout opgetreden: " + e);
58      } finally {
59          System.out.println();
60          System.out.println("Einde programma.");
61      }
62
63
64       }
65
66
67  }
```

2 Opgave 3

*

Listing 4: Main.

```
1  package opgave3;
2
3  import opgave3.tokenizer.*;
4  import opgave3.parser.*;
5
6  public class Main {
7      public static void main(String[] args) {
8      Tokenizer tok = new Tokenizer(System.in);
9      if (Program.tryParse(tok)) {
10         System.out.println("GOED");
11     } else {
12         System.out.println("FOUT");
13     }
14      }
15 }
```

2.1 Tokenizer

Listing 5: Token.

```
1  package opgave3.tokenizer;
2
3  public abstract class Token {
4
5      private String originalString;
6
7      public Token(String s) {
8          originalString = s;
9      }
10
11     public Token(char ch) {
12                 originalString = Character.toString(ch);
13     }
14
15     public String toString() {
16         return originalString;
17     }
18
```

```
}
```

Listing 6: Tokenizer.

```
package opgave3.tokenizer;

import java.io.*;

/**
 * Deze klasse is een tokenizer voor rekenmachine-expressies
 * Een expressies wordt opgebroken in elementaire stukjes die gerepresenteerd worden door
 * subklassen van de klasse Token.
 *
 * De tokenizer heeft een 'wijzer' die 'op' een bepaald token staat. Met getCurrent() kan
 * het huidige token worden opgevraagd. Met moveNext() gaat de tokenizer naar het volgende
 * token. Als alle tokens op zijn worden uitsluitend nog EOFToken-objecten opgeleverd.
 */
public class Tokenizer {

   private StreamTokenizer st;
   private Token current;

   /**
    * Maakt een nieuwe Tokenizer die leest uit de gegeven InputStream.
    * Gebruik bijvoorbeeld System.in om te lezen uit de standaard invoer.
    */
   public Tokenizer(InputStream is) {
      st = new StreamTokenizer(new InputStreamReader(is));
      setup();
   }

   /**
    * Maakt een nieuwe Tokenizer die leest uit de gegeven Reader leest
    */
        public Tokenizer(Reader r) {
                st = new StreamTokenizer(r);
                setup();
        }

   /**
    * Maakt een nieuwe Tokenizer die leest uit de gegeven String.
    */
   public Tokenizer(String s) {
      st = new StreamTokenizer(new StringReader(s));
      setup();
   }

   /**
    * Levert het huidige token als resultaat.
    */
   public Token getCurrent(){
      return current;
   }

   /**
    * Schuift de tokenizer door naar het volgende token.
    */
   public void moveNext() {
       current = makeNext();
   }

   private void setup() {
           /* getallen moeten als token beschouwd worden */
       st.parseNumbers();
       /* / en - niet als speciaal beschouwen */
       st.ordinaryChar('/');
```

```
63          st.ordinaryChar('-');
64          moveNext();
65      }
66
67      private Token makeNext() {
68          try {
69              st.nextToken();
70          } catch (IOException e) {
71              return new InvalidToken("<IOException>");
72          }
73          switch (st.ttype) {
74              case StreamTokenizer.TT_EOF:
75                  return new EOFToken();
76                   case StreamTokenizer.TT_NUMBER:
77                  return new NumberToken(st.nval);
78              case StreamTokenizer.TT_WORD:
79                  if (st.sval.equals("INC")) {
80                 return new IncToken(st.sval);
81                  } else if (st.sval.equals("DEC")) {
82                 return new DecToken(st.sval);
83                  } else if (st.sval.equals("LET")) {
84                                  return new LetToken(st.sval);
85                  } else if (st.sval.equals("WHILE")) {
86                 return new WhileToken(st.sval);
87                  } else if (st.sval.equals("DO")) {
88                 return new DoToken(st.sval);
89                  } else if (st.sval.equals("END")) {
90                                  return new EndToken(st.sval);
91                  }
92                  return new VarToken(st.sval);
93              default:
94                  char ch = (char) st.ttype;
95                  switch (ch) {
96                    case ';' : return new SemicolonToken(ch);
97                    case '=' : return new AssignmentToken(ch);
98                    default  : return new InvalidToken(ch);
99                                  }
100
101         }
102     }
103
104 }
```

Listing 7: AssignmentToken.

```
1 package opgave3.tokenizer;
2
3 public class AssignmentToken extends Token {
4         public AssignmentToken(char op) {
5                 super(op);
6         }
7 }
```

Listing 8: DecToken.

```
1
2 package opgave3.tokenizer;
3
4 public class DecToken extends Token {
5
6         public DecToken(String s) {
7                 super(s);
8         }
9
10 }
```

Listing 9: DoToken.

```
1
2 package opgave3.tokenizer;
3
4 public class DoToken extends Token {
5
6         public DoToken(String s) {
7                 super(s);
8         }
9
10 }
```

Listing 10: EndToken.

```
1
2 package opgave3.tokenizer;
3
4 public class EndToken extends Token {
5
6         public EndToken(String s) {
7                 super(s);
8         }
9
10 }
```

Listing 11: EOFToken.

```
1 package opgave3.tokenizer;
2
3 public class EOFToken extends Token {
4
5    public EOFToken() {
6       super("");
7    }
8
9    public String toString() {
10      return "<EOF>";
11   }
12
13 }
```

Listing 12: IncToken.

```
1
2 package opgave3.tokenizer;
3
4 public class IncToken extends Token {
5
6         public IncToken(String s) {
7                 super(s);
8         }
9
10 }
```

Listing 13: InvalidToken.

```
1 package opgave3.tokenizer;
2
3 public class InvalidToken extends Token {
4
5    public InvalidToken(String s) {
6       super(s);
7    }
8
9    public InvalidToken(char ch) {
```

```
10        super(ch);
11    }
12
13 }
```

Listing 14: LetToken.

```
1
2 package opgave3.tokenizer;
3
4 public class LetToken extends Token {
5
6         public LetToken(String s) {
7                 super(s);
8         }
9
10 }
```

Listing 15: NumberToken.

```
1 package opgave3.tokenizer;
2
3 public class NumberToken extends Token {
4
5    private double value;
6
7    public NumberToken(double v) {
8       super(Double.toString(v));
9       value = v;
10   }
11
12   public double getValue() {
13      return value;
14   }
15
16 }
```

Listing 16: SemicolonToken.

```
1 package opgave3.tokenizer;
2
3 public class SemicolonToken extends Token {
4         public SemicolonToken(char op) {
5                 super(op);
6         }
7 }
```

Listing 17: VarToken.

```
1 package opgave3.tokenizer;
2
3 public class VarToken extends Token {
4
5    private String name;
6
7    public VarToken(String s) {
8       super(s);
9       name = s.toLowerCase();
10   }
11
12   public String getName() {
13      return name;
14   }
15
16 }
```

Listing 18: WhileToken.

```
1
2 package opgave3.tokenizer;
3
4 public class WhileToken extends Token {
5
6         public WhileToken(String s) {
7                 super(s);
8         }
9
10 }
```

2.2 Parser

Listing 19: Parser.

```
1 package opgave3.parser;
2 import opgave3.tokenizer.*;
3 //import opgave3.tokenizer.Tokenizer;
4 //import opgave3.tokenizer.Token;
5
6 public abstract class Parser {
7
8     public static boolean tryParse(Tokenizer tok) {
9    reportError("nog te implementeren");
10   return false;
11    }
12
13    protected static void reportError(String message) {
14   System.out.println("FOUT");
15   System.err.println(message);
16   System.exit(0);
17    }
18
19 }
```

Listing 20: Program.

```
1 package opgave3.parser;
2 import opgave3.tokenizer.*;
3
4 public class Program extends Parser {
5
6     public static boolean tryParse(Tokenizer tok) {
7    if (StatementList.tryParse(tok) && tok.getCurrent() instanceof EOFToken) {
8        return true;
9    } else {
10       reportError("EOF verwacht, maar " + tok.getCurrent() + " gevonden");
11       return false;
12   }
13    }
14
15 }
```

Listing 21: StatementList.

```
1 package opgave3.parser;
2 import opgave3.tokenizer.*;
3
4 public class StatementList extends Parser {
5
6     public static boolean tryParse(Tokenizer tok) {
7    while (Statement.tryParse(tok)) {
8        if (tok.getCurrent() instanceof SemicolonToken) {
```

```
9         tok.moveNext();
10         } else {
11        reportError("; verwacht, maar " + tok.getCurrent() + " gevonden");
12         }
13    }
14    return true;
15     }
16
17 }
```

Listing 22: Statement.

```
1 package opgave3.parser;
2 import opgave3.tokenizer.*;
3
4 public class Statement extends Parser {
5
6     public static boolean tryParse(Tokenizer tok) {
7    if (Increment.tryParse(tok)) {
8        return true;
9    } else if (Decrement.tryParse(tok)) {
10       return true;
11   } else if (Assignment.tryParse(tok)) {
12       return true;
13   } else {
14       return Repetition.tryParse(tok);
15   }
16    }
17
18 }
```

Listing 23: Repetition.

```
1 package opgave3.parser;
2 import opgave3.tokenizer.*;
3
4 public class Repetition extends Parser {
5
6     public static boolean tryParse(Tokenizer tok) {
7    if (tok.getCurrent() instanceof WhileToken) {
8        tok.moveNext(); // weglezen WHILE
9        if (tok.getCurrent() instanceof VarToken) {
10       tok.moveNext(); // variabele weglezen
11       if (tok.getCurrent() instanceof DoToken) {
12           tok.moveNext(); // weglezen DO
13                     if (StatementList.tryParse(tok)) {
14          if (tok.getCurrent() instanceof EndToken) {
15              tok.moveNext(); // weglezen END
16              return true;
17          } else {
18              reportError("END verwacht");
19          }
20           } else {
21          reportError("statements verwacht");
22           }
23       } else {
24           reportError("DO verwacht");
25       }
26        } else {
27       reportError("variabele verwacht");
28        }
29   }
30   return false;
31    }
32
33 }
```

Listing 24: Assignment.

```
package opgave3.parser;
import opgave3.tokenizer.*;

public class Assignment extends Parser {

    public static boolean tryParse(Tokenizer tok) {
   if (tok.getCurrent() instanceof LetToken) {
       tok.moveNext(); // weglezen LET
       if (tok.getCurrent() instanceof VarToken) {
      tok.moveNext(); // variabele weglezen
      if (tok.getCurrent() instanceof AssignmentToken) {
          tok.moveNext(); // weglezen =
          if (tok.getCurrent() instanceof VarToken ||
         tok.getCurrent() instanceof NumberToken) {
         tok.moveNext(); // weglezen variable / getal
         return true;
          } else {
         reportError("variabele / getal verwacht");
          }
      } else {
          reportError("= verwacht");
      }
       } else {
      reportError("variable verwacht");
       }
   }
   return false;
    }

}
```

Listing 25: Decrement.

```
package opgave3.parser;
import opgave3.tokenizer.*;

public class Decrement extends Parser {

    public static boolean tryParse(Tokenizer tok) {
   if (tok.getCurrent() instanceof DecToken) {
       tok.moveNext();
       if (tok.getCurrent() instanceof VarToken) {
      tok.moveNext(); // variabele weglezen
      if (tok.getCurrent() instanceof VarToken ||
                  tok.getCurrent() instanceof NumberToken) {
          tok.moveNext(); // optionele variable / getal weglezen
      }
      return true;
       } else {
      reportError("variable verwacht");
       }
   }
   return false;
    }

}
```

Listing 26: Increment.

```
package opgave3.parser;
import opgave3.tokenizer.*;

public class Increment extends Parser {

```

```
6     public static boolean tryParse(Tokenizer tok) {
7    if (tok.getCurrent() instanceof IncToken) {
8        tok.moveNext();
9        if (tok.getCurrent() instanceof VarToken) {
10       tok.moveNext(); // variabele weglezen
11       if (tok.getCurrent() instanceof VarToken ||
12                   tok.getCurrent() instanceof NumberToken) {
13           tok.moveNext(); // optionele variable / getal weglezen
14       }
15       return true;
16        } else {
17       reportError("variable verwacht");
18        }
19   }
20   return false;
21    }
22
23 }
```

3 Opgave 4

*

Listing 27: Main.

```
1 package opgave4;
2 import opgave4.parser.*;
3 import opgave3.tokenizer.*; // reuse code
4 import java.io.*;
5
6 public class Main {
7
8     public static void main(String[] args) {
9    // controle aantal argumenten
10   if (args.length != 1) {
11      System.out.println("usage: java opgave4.Main <file>");
12      System.exit(0);
13   }
14
15   // hoofd
16   try {
17
18       BufferedReader r = new BufferedReader(new FileReader(args[0]));
19       Tokenizer t = new Tokenizer(r);
20       String p = Program.tryParse(t,0);
21       System.out.println(p);
22
23   } catch (IOException ie) {
24       System.out.println("Fout bij openen / lezen bestand\n" + ie);
25   } catch (ParseException pe) {
26       System.out.println("Fout tijdens parseren\n" + pe);
27   }
28    }
29 }
```

3.1 Parser

Listing 28: ParseException.

```
1 package opgave4.parser;
2
```

```
3 public class ParseException extends Exception {
4
5     public ParseException(String msg) {
6    super("Error during parsing: " + msg);
7     }
8
9 }
```

Listing 29: Parser.

```
1 package opgave4.parser;
2 import opgave3.tokenizer.*; // reuse code
3
4 public abstract class Parser {
5
6     public static String tryParse(Tokenizer tok, int level) throws ParseException {
7    reportError("tryParse must be implemented");
8    return "";
9     }
10
11     protected static void reportError(String message) throws ParseException {
12    throw new ParseException(message);
13     }
14
15
16     protected static String indent(int level) { // protected, dus ook voor subklassen
           beschikbaar
17    String res = "";
18    while (level > 0) {
19        res += "  "; // 2 spaties
20        level--;
21    }
22    return res;
23     }
24
25 }
```

Listing 30: Program.

```
1 package opgave4.parser;
2 import opgave3.tokenizer.*;
3
4 public class Program extends Parser {
5
6     public static String tryParse(Tokenizer tok, int level) throws ParseException {
7    String stmts = StatementList.tryParse(tok, 0); // begin zonder inspringniveau
8    if (tok.getCurrent() instanceof EOFToken) {
9        // skip
10    } else {
11        reportError("EOF verwacht, maar " + tok.getCurrent() + " gevonden");
12    }
13    return stmts;
14     }
15
16 }
```

Listing 31: StatementList.

```
1 package opgave4.parser;
2 import opgave3.tokenizer.*;
3
4 public class StatementList extends Parser {
5
6     public static String tryParse(Tokenizer tok, int level) throws ParseException {
7    String stmts = "";
```

```
8     String statement = Statement.tryParse(tok,level);
9     while(!statement.equals("")) { // herhaal tot we geen statement meer vinden
10        if (tok.getCurrent() instanceof SemicolonToken) {
11       stmts += statement + tok.getCurrent() + "\n";
12       tok.moveNext(); // weglezen ;
13       statement = Statement.tryParse(tok,level);
14        } else {
15       reportError("; verwacht, maar " + tok.getCurrent() + " gevonden");
16        }
17    }
18    return stmts;
19     }
20
21 }
```

Listing 32: Statement.

```
1 package opgave4.parser;
2 import opgave3.tokenizer.*;
3
4 public class Statement extends Parser {
5
6     public static String tryParse(Tokenizer tok, int level) throws ParseException {
7    String s;
8    s = Increment.tryParse(tok, level);
9    if (!s.equals("")) {
10       return s;
11   }
12   s = Decrement.tryParse(tok, level);
13   if (!s.equals("")) {
14       return s;
15   }
16   s = Assignment.tryParse(tok, level);
17   if (!s.equals("")) {
18       return s;
19   }
20   s = Repetition.tryParse(tok, level);
21   return s;
22    }
23
24 }
```

Listing 33: Repetition.

```
1 package opgave4.parser;
2 import opgave3.tokenizer.*;
3
4 public class Repetition extends Parser {
5
6     public static String tryParse(Tokenizer tok, int level) throws ParseException {
7    String res = "";
8    if (tok.getCurrent() instanceof WhileToken) {
9        res += indent(level) + tok.getCurrent(); // spring in
10       tok.moveNext(); // weglezen WHILE
11       if (tok.getCurrent() instanceof VarToken) {
12      res += " " + tok.getCurrent();
13      tok.moveNext(); // variabele weglezen
14      if (tok.getCurrent() instanceof DoToken) {
15          res += " " + tok.getCurrent();
16          tok.moveNext(); // weglezen DO
17          res += "\n";
18          String stmts = StatementList.tryParse(tok, level + 1);
19          if (!stmts.equals("")) {
20         res += stmts;
21         if (tok.getCurrent() instanceof EndToken) {
```

```
22              res += indent(level) + tok.getCurrent(); // vergeet END niet op juiste plek in
                    te springen
23              tok.moveNext(); // weglezen END
24          } else {
25              reportError("END verwacht");
26          }
27           } else {
28          reportError("statements verwacht");
29           }
30       } else {
31            reportError("DO verwacht");
32       }
33        } else {
34       reportError("variabele verwacht");
35        }
36    }
37    return res;
38     }
39
40 }
```

Listing 34: Assignment.

```
1  package opgave4.parser;
2  import opgave3.tokenizer.*;
3
4  public class Assignment extends Parser {
5
6      public static String tryParse(Tokenizer tok, int level) throws ParseException {
7     String res = "";
8     if (tok.getCurrent() instanceof LetToken) {
9         res += indent(level) + tok.getCurrent(); // spring in
10        tok.moveNext(); // weglezen LET
11        if (tok.getCurrent() instanceof VarToken) {
12       res += " " + tok.getCurrent();
13       tok.moveNext(); // variabele weglezen
14       if (tok.getCurrent() instanceof AssignmentToken) {
15           res += " " + tok.getCurrent();
16           tok.moveNext(); // weglezen =
17           if (tok.getCurrent() instanceof VarToken ||
18          tok.getCurrent() instanceof NumberToken) {
19          res += " " + tok.getCurrent();
20          tok.moveNext(); // weglezen variable / getal
21           } else {
22          reportError("variabele / getal verwacht");
23           }
24       } else {
25           reportError("= verwacht");
26       }
27        } else {
28       reportError("variable verwacht");
29        }
30    }
31    return res;
32     }
33
34 }
```

Listing 35: Decrement.

```
1  package opgave4.parser;
2  import opgave3.tokenizer.*;
3
4  public class Decrement extends Parser {
5
```

```
6     public static String tryParse(Tokenizer tok, int level) throws ParseException {
7    String res = "";
8    if (tok.getCurrent() instanceof DecToken) {
9        res += indent(level) + tok.getCurrent(); // spring in
10       tok.moveNext();
11       if (tok.getCurrent() instanceof VarToken) {
12      res += " " + tok.getCurrent();
13      tok.moveNext(); // variabele weglezen
14      if (tok.getCurrent() instanceof VarToken ||
15                  tok.getCurrent() instanceof NumberToken) {
16          res += " " + tok.getCurrent();
17          tok.moveNext(); // optionele variable / getal weglezen
18      }
19       } else {
20      reportError("variable verwacht");
21       }
22   }
23   return res;
24    }
25
26 }
```

Listing 36: Increment.

```
1 package opgave4.parser;
2 import opgave3.tokenizer.*;
3
4 public class Increment extends Parser {
5
6     public static String tryParse(Tokenizer tok, int level) throws ParseException {
7    String res = "";
8    if (tok.getCurrent() instanceof IncToken) {
9        res += indent(level) + tok.getCurrent(); // spring in
10       tok.moveNext();
11       if (tok.getCurrent() instanceof VarToken) {
12      res += " " + tok.getCurrent();
13      tok.moveNext(); // variabele weglezen
14      if (tok.getCurrent() instanceof VarToken ||
15                  tok.getCurrent() instanceof NumberToken) {
16          res += " " + tok.getCurrent();
17          tok.moveNext(); // optionele variable / getal weglezen
18      }
19       } else {
20      reportError("variable verwacht");
21       }
22   }
23   return res;
24    }
25
26 }
```