Uitwerkingen Tentamen Objectgeorienteerd Programmeren 23-04-2007

Opgave 1

a. Een klasse beschrijft een verzameling van gelijksoortige objecten. De beschrijving legt vast welke eigenschappen / velden en methoden elke object dat tot de klasse behoort heeft. Voorts beschrijft het een (of meerdere) constructor(en) waarmee een instantie van de klasse, d.w.z.\ een concreet object gemaakt kan worden. De constructor voorziet velden evt. van een waarde.

   Een object is dus een (specifieke) instantie van een klasse, en een object heeft z'n eigen waarden van de velden onafhankelijk van andere objecten. Op een object kan je methoden aanroepen waardoor het object deze waarden eventueel kan gaan aanpassen.

b. Inheritance / overerving is het structureren van de klassen in een hierarchy. Een (sub)klasse is een uitbreiding op een reeds bestaande (super)klasse. Een object van de subklasse bevat alle functionaliteit (toegankelijke velden en methoden) die een object van de superklasse ook heeft, en voegt daar extra functionaliteit aan toe, en/of past enige overgeerfde functionaliteit aan (overriding).

   Als B een subklasse is van een klasse A, dan mag overal waar een object van klasse A verwacht / gevraagd wordt, ook een object van klasse B gegeven worden; omdat B tenminste alle methoden en velden die A beschikbaar stelt, ook beschikbaar stelt. (Impliciete upcasting)

c. Een natuurlijk criterium om na te gaan of er overerving / inheritance toegepast kan worden is de "is-een" / "is-a" relatie. Als klasse B een specifiekere vorm is van een algemenere klasse A, d.w.z. objecten van B zijn eigenlijk net als objecten van klasse A, maar ...(voegen extra zaken toe enz.), dan is B een subklasse van klasse A.

Opgave 2

De uiteindelijke klassen Heap, HeapNode en Main:

```
class Heap {
   HeapNode root;

   public Heap() { // default constructor
         root = null;
   }

   public boolean contains(int v) {
         return contains(v, root);
   }

   private boolean contains(int v, HeapNode t) {
         if (t == null) {
```

```
            return false;
        } else if (t.val >= v) {
            /* rest van de boom bevat waarden groter gelijk aan t.val */
            return t.val == v;
        } else { // t != null && t.val < v
        return contains(v, t.left) || contains(v, t.right);
        }
}

public boolean checkConsistency() {
        return checkHeap(root) && checkBalans(root);
}


private boolean checkHeap(HeapNode t) {
        if (t == null) {
            return true;
        } else {
            return checkHeap(t.left, t.val) && checkHeap(t.right, t.val);
        }
}


private boolean checkHeap(HeapNode t, int v) {
        if (t == null) {
            return true;
        } else {
            return t.val >= v && checkHeap(t.left, t.val) && checkHeap(t.right, t.val);
        }
}

private boolean checkBalans(HeapNode t) {
        if (t == null) {
            return true;
        } else {
            boolean balans = checkBalans(t.left) && checkBalans(t.right);
            if (balans) {
                    int l = size(t.left);
                    int r = size(t.right);
                    // controle van het balance veld
                    if (t.balance) {
                        return l == r;
                    } else {
                        return l == r + 1;
                    }
            } else {
                    return false;
            }
        }
}
```

```
public int size() {
	return size(root);
}


private int size(HeapNode t) {
	if (t == null) {
	   return 0;
	} else {
	   return 1 + size(t.left) + size(t.right);
	}
}

private void swapValues(HeapNode a, HeapNode b) {
	int tmpVal = a.val; a.val = b.val; b.val = tmpVal;
}


public void add(int v) {
	root = add(v, root);
}


private HeapNode add(int v, HeapNode t) {
	if (t == null) {
	   return new HeapNode(v);
	} else {
	   if (t.balance) {
		t.left = add(v, t.left);
		if (t.left.val < t.val) {
		   swapValues(t, t.left);
		}
	   } else {
		t.right = add(v, t.right);
		if (t.right.val < t.val) {
		   swapValues(t, t.right);
		}
	   }
	   t.balance = !t.balance;
	   return t;
	}
}

public int deleteMinimum() {
	if (root == null) {
	   throw new RuntimeException();
	}
	// root != null
```

```
        int minimum = root.val;
        HeapNode t = root;
        HeapNode p = null;
        // t != null;
        while (t.left != null) {     // t != null && t.left != null
            p = t; // houd parent bij.
            if (p.balance) { // balance & t.left != null => t.right != null
                  // laatst toegevoegd aan rechts
                  t = t.right;
            } else {
                  t = t.left;
            }
            // t != null
            // op dit pad verdwijnt een knoop
            p.balance = !p.balance;
        }
        // t = laatst toegevoegde knoop, p = t's parent
        if (p == null) {
            // t == root != null
            root = null;
            return minimum;
        }
        // verwijder laatst toegevoegde knoop
        if (p.left == t) {
            p.left = null;
        } else {
            p.right = null;
        }
        root.val = t.val;
        heapify(root);
    return minimum;
}

/** herstelt de heapeigenschap */
private void heapify(HeapNode t) {
        /* pre:
         knoop t voldoet aan de balans eigenschap,
             knoop t voldoet niet aan heap-eigenschap,
             kinderen van t voldoen wel aan heap-eigenschap.
        */
    while (t != null) {
            // in knoop t moet minimale waarde komen te staan.
            HeapNode min = null;
            if (t.left  != null &&   t.val >  t.left.val) {
                  min = t.left;
            }
            if (t.right != null && min != null && min.val > t.right.val) {
                  min = t.right;
            }
            if (min != null) {
```

```
                // min wijst naar knoop met kleinste kind
                swapValues(t, min);
            }
            t = min;
        }
    }

    // methoden voor het afdrukken

    public String toString() {
        return toString(root);
    }

    private String toString(HeapNode t) {
        if (t == null) {
            return "";
        } else {
            String s = "" + t.val;
            if (t.left != null) {
                s += "[ " + toString(t.left) + ", " + toString(t.right) + "]";
            }
            return s;
        }
    }

}

class HeapNode {
  /** de opgeslagen waarde */
    int val;

    /** de balans: 0 = links en rechts evenveel kinderen
                   1 = links heeft 1 kind meer dan rechts */
    boolean balance;


    HeapNode left  = null;
    HeapNode right = null;

    HeapNode() {
        balance = true;
    }

    HeapNode(int v) {
        this();
        val = v;
    }
}

import java.util.*;
```

```
public class Main {

   public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        Heap h = new Heap();
        while (true) {
           int i = sc.nextInt();
           if (i < 0) {
                break;
           }
           h.add(i);
           System.out.println(h);
           if (!h.checkConsistency()) {
                System.out.println("FOUT: heap schendt de gestelde eigenschappen!");
                System.exit(0);
           }
        }

        while (h.size() > 0) {
           System.out.println(h.deleteMinimum());
           System.out.println(h);
           if (!h.checkConsistency()) {
                System.out.println("FOUT: heap schendt de gestelde eigenschappen!");
                System.exit(0);
           }
        }

   }


}
```

Opgave 3
De uiteindelijke klassen Tokenizer en verschillende Tokens klassen:

1. 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();
	}

	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('/');
	   st.ordinaryChar('-');
	   moveNext();
	}
```

```
        private Token makeNext() {
                try {
                        st.nextToken();
                } catch (IOException e) {
                        return new InvalidToken("<IOException>");
                }
                switch (st.ttype) {
                        case StreamTokenizer.TT_EOF:
                           return new EOFToken();
                      case StreamTokenizer.TT_NUMBER:
                           return new NumberToken(st.nval);
                        case StreamTokenizer.TT_WORD:
                           if (st.sval.equals("SET")) {
                                 return new SetToken(st.sval);
                           } else if (st.sval.equals("GET")) {
                                 return new GetToken(st.sval);
                      }
                           return new VarToken(st.sval);
                        default:
                           char ch = (char) st.ttype;
                           switch (ch) {
                             case ';' : return new SemicolonToken(ch);
                             case '=' : return new AssignmentToken(ch);
                             case '+' : return new AddToken(ch);
                             case '-' : return new SubtToken(ch);
                       case '*' : return new MultToken(ch);
                       case ':' : return new DivToken(ch);
                                 default  : return new InvalidToken(ch);
                      }

                }
        }

}
```

```
package opgave3.tokenizer;

public class AddToken extends OpToken {

      public AddToken(char ch) {
            super(ch);
      }

}
```

2.
```
package opgave3.tokenizer;

public class AssignmentToken extends Token {
      public AssignmentToken(char op) {
```

```
            super(op);
        }
}
```

3. package opgave3.tokenizer;

```
public class DivToken extends OpToken {

        public DivToken(char ch) {
            super(ch);
        }

}
```

4. package opgave3.tokenizer;

```
public class EOFToken extends Token {

            public EOFToken() {
                    super("");
            }

            public String toString() {
                    return "<EOF>";
            }

}
```

5. package opgave3.tokenizer;

```
public class GetToken extends Token {

        public GetToken(String s) {
            super(s);
        }

}
```

6. package opgave3.tokenizer;

```
public class InvalidToken extends Token {

            public InvalidToken(String s) {
                    super(s);
            }

            public InvalidToken(char ch) {
                    super(ch);
```

```
        }

}
```

7. package opgave3.tokenizer;

```
public class MultToken extends OpToken {

    public MultToken(char ch) {
        super(ch);
    }

}
```

8. package opgave3.tokenizer;

```
public class NumberToken extends Token {

        private double value;

        public NumberToken(double v) {
                super(Double.toString(v));
                value = v;
        }

        public double getValue() {
                return value;
        }

}
```

9. package opgave3.tokenizer;

```
public class OpToken extends Token {

    public OpToken(char ch) {
        super(ch);
    }

}
```

10. package opgave3.tokenizer;

```
public class SemicolonToken extends Token {
    public SemicolonToken(char op) {
        super(op);
    }
}
```

11. package opgave3.tokenizer;

```
public class SetToken extends Token {

	public SetToken(String s) {
		super(s);
	}

}
```

12. package opgave3.tokenizer;

```
public class SubtToken extends OpToken {

	public SubtToken(char ch) {
		super(ch);
	}

}
```

13. package opgave3.tokenizer;

```
public abstract class Token {

	private String originalString;

	public Token(String s) {
		originalString = s;
	}

	public Token(char ch) {
	    originalString = Character.toString(ch);
	}

	public String toString() {
		return originalString;
	}

}
```

14. package opgave3.tokenizer;

```
public class VarToken extends Token {

	private String name;

	public VarToken(String s) {
		super(s);
```

```
            name = s.toLowerCase();
        }

        public String getName() {
            return name;
        }

}
```

De uiteindelijke Parser klassen:

1.
```
package opgave3.parser;
import opgave3.tokenizer.*;
//import opgave3.tokenizer.Tokenizer;
//import opgave3.tokenizer.Token;

public abstract class Parser {

   public static boolean tryParse(Tokenizer tok) {
        reportError("nog te implementeren");
        return false;
   }

   protected static void reportError(String message) {
        System.out.println("FOUT");
        System.err.println(message);
        System.exit(0);
   }

}
```

2.
```
package opgave3.parser;
import opgave3.tokenizer.*;

public class Assignment extends Parser {

   public static boolean tryParse(Tokenizer tok) {
        if (tok.getCurrent() instanceof SetToken) {
           tok.moveNext(); // weglezen SET
           if (tok.getCurrent() instanceof VarToken) {
                tok.moveNext(); // variabele weglezen
                if (tok.getCurrent() instanceof AssignmentToken) {
                   tok.moveNext(); // weglezen =
                   if (Expr.tryParse(tok)) {
                        return true;
                   } else {
                        reportError("expressie verwacht");
                        return false;
                   }
                } else {
```

```
                    reportError("= verwacht");
                }
            } else {
                reportError("variable verwacht");
            }
        }
        return false;
    }

}
```

3.
```
package opgave3.parser;
import opgave3.tokenizer.*;

public class Query extends Parser {

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

}
```

4.
```
package opgave3.parser;
import opgave3.tokenizer.*;

public class StatementList extends Parser {

    public static boolean tryParse(Tokenizer tok) {
        while (Statement.tryParse(tok)) {
            if (tok.getCurrent() instanceof SemicolonToken) {
                tok.moveNext();
            } else {
                reportError("; verwacht, maar " + tok.getCurrent() + " gevonden");
            }
        }
        return true;
    }

}
```

5.
```
package opgave3.parser;
```

```
import opgave3.tokenizer.*;

public class Expr extends Parser {

   public static boolean tryParse(Tokenizer tok) {
         if (Value.tryParse(tok)) {
            if (tok.getCurrent() instanceof OpToken) {
                  tok.moveNext(); // weglezen operator
                  return Value.tryParse(tok);
            }
            return true;
         }
         reportError("<expr> verwacht");
         return false;

   }

}
```

6.
```
package opgave3.parser;
import opgave3.tokenizer.*;

public class Program extends Parser {

   public static boolean tryParse(Tokenizer tok) {
         if (StatementList.tryParse(tok) && tok.getCurrent() instanceof EOFToken) {
            return true;
         } else {
            reportError("EOF verwacht, maar " + tok.getCurrent() + " gevonden");
            return false;
         }
   }

}
```

7.
```
package opgave3.parser;
import opgave3.tokenizer.*;

public class Statement extends Parser {

   public static boolean tryParse(Tokenizer tok) {
         if (Assignment.tryParse(tok)) {
            return true;
         } else {
            return Query.tryParse(tok);
         }
   }

}
```

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

public class Value extends Parser {

```
   public static boolean tryParse(Tokenizer tok) {
         if (tok.getCurrent() instanceof VarToken ||
            tok.getCurrent() instanceof NumberToken) {
            tok.moveNext(); // weglezen Var of Number
            return true;
         }
         return false;
   }

}
```

De klasse Main:

```
package opgave3;

import opgave3.tokenizer.*;
import opgave3.parser.*;

public class Main {
   public static void main(String[] args) {
         Tokenizer tok = new Tokenizer(System.in);
         if (Program.tryParse(tok)) {
            System.out.println("GOED");
         } else {
            System.out.println("FOUT");
         }
   }
}
```

Opgave 4
De Parser:

1. package opgave4.parser;

```
import opgave4.tokenizer.*;

abstract class ParseTreeNode {

         protected static void reportError(String message) throws ParseException {
           throw new ParseException(message);
         }

}
```

2. package opgave4.parser;

```
public class ParseException extends Exception {

   public ParseException(String msg) {
         super("Error during parsing: " + msg);
   }

}
```

3. package opgave4.parser;

```
public class UninitializedVariableException extends EvaluationException {

         public UninitializedVariableException() {
                  super("uninitialized variable exception");
         }

       public UninitializedVariableException(String msg) {
            super(msg);
         }

}
```

4. package opgave4.parser;

```
public class DivByZeroException extends EvaluationException {
         public DivByZeroException() {
                  super("division by zero exception");
         }
}
```

5. package opgave4.parser;

```
public class EvaluationException extends Exception {

   public EvaluationException(String msg) {
         super(msg);
   }

}
```

6. package opgave4.parser;

```
import opgave4.tokenizer.*;

class Assignment extends Statement {

   private String    var;
   private ExprNode expr;
```

```
    Assignment(String v, ExprNode e) {
        var = v; expr = e;
    }

    static Assignment tryParse(Tokenizer tok) throws ParseException {

        if (tok.getCurrent() instanceof SetToken) {
           tok.moveNext(); // weglezen SET
           if (tok.getCurrent() instanceof VarToken) {
                String v = ((VarToken) tok.getCurrent()).getName();
                tok.moveNext(); // variabele weglezen
                if (tok.getCurrent() instanceof AssignmentToken) {
                   tok.moveNext(); // weglezen =
                   ExprNode e = Expr.tryParse(tok);
                   if (e != null) {
                        return new Assignment(v, e);
                   } else {
                        reportError("expression expected");
                   }
                } else {
                   reportError("= expected");
                }
           } else {
                reportError("variable expected");
           }
        }
        return null;
    }

    void execute(State s) throws EvaluationException {
        s.assign(var, expr.evaluate(s));
    }

}
```

7.
```
package opgave4.parser;
import opgave4.tokenizer.*;

class Expr extends ExprNode {

    private Value left;
    private OpToken op;
    private Value right;

    Expr(Value l, OpToken o, Value r) {
        left = l; op = o; right = r;
    }

    static ExprNode tryParse(Tokenizer tok) throws ParseException {
        Value l = Value.tryParse(tok);
```

```
        if (l != null) {
           if (tok.getCurrent() instanceof OpToken) {
                 OpToken o = (OpToken) tok.getCurrent();
                 tok.moveNext(); // weglezen operator
                 Value r = Value.tryParse(tok);
                 if (r != null) {
                    return new Expr(l,o,r);
                 } else {
                    reportError("value expected");
                 }
           } else {
                 return l; // a Value
           }
        }
        return null;
   }

   double evaluate(State s) throws EvaluationException {
        double lval = left.evaluate(s);
        double rval = right.evaluate(s);
        switch (op.getOperator()) {
          case '+' : return lval + rval;
          case '-' : return lval - rval;
          case '*' : return lval * rval;
          case ':' : if (rval != 0) {
                        return lval / rval;
                     }
                     throw new DivByZeroException();
       default:   throw new EvaluationException("unknown operator");
        }
   }

}
```

8.
```
package opgave4.parser;
import opgave4.tokenizer.*;

public abstract class ExprNode extends ParseTreeNode {

   abstract double evaluate(State s) throws EvaluationException;

}
```

9.
```
package opgave4.parser;

import java.util.*;

public class HashtableState implements State {

  Hashtable<String, Double> ht;
```

```
  public HashtableState() {
     ht = new Hashtable<String, Double>();
  }

  public void assign(String var, double value) {
    ht.put(var, value);
  }

  public double query(String var) throws UninitializedVariableException {
    if (ht.containsKey(var)) {
      return ht.get(var).doubleValue();
    }
    // else: variable not associated with value
    throw new UninitializedVariableException(var + " has not been assigned a value");
  }

}
```

10.

```
package opgave4.parser;
import opgave4.tokenizer.*;

class Num extends Value {

   private double num;

   Num(double n) {
         num = n;
   }

   static Num tryParse(Tokenizer tok) {
         if (tok.getCurrent() instanceof NumberToken) {
            NumberToken nt = (NumberToken) tok.getCurrent();
            tok.moveNext();
            return new Num(nt.getValue());
         }
         return null;
   }

   double evaluate(State s) {
         return num;
   }

}
```

11.

```
package opgave4.parser;
import opgave4.tokenizer.*;

public class Program extends ParseTreeNode {
   private StatementList statementList;
```

```
    Program(StatementList sl) {
        statementList = sl;
    }

    public static Program tryParse(Tokenizer tok) throws ParseException {
        StatementList sl = StatementList.tryParse(tok);
        if (sl != null && tok.getCurrent() instanceof EOFToken) {
            return new Program(sl);
        }
        reportError("EOF expected, but found " + tok.getCurrent());
        return null;
    }

    public void execute(State s) throws EvaluationException {
        statementList.execute(s);
    }

}
```

12.
```
package opgave4.parser;
import opgave4.tokenizer.*;

class Query extends Statement {
    private String var;

    Query(String v) {
        var = v;
    }

    static Query tryParse(Tokenizer tok) throws ParseException {

        if (tok.getCurrent() instanceof GetToken) {
            tok.moveNext(); // weglezen GET
            if (tok.getCurrent() instanceof VarToken) {
                Query q = new Query(((VarToken) tok.getCurrent()).getName());
                tok.moveNext(); // variabele weglezen
                return q;
            } else {
                reportError("variable expected");
            }
        }
        return null;
    }

    void execute(State s) throws UninitializedVariableException {
        System.out.println(var + " = " + s.query(var));
    }

}
```

13. package opgave4.parser;

```
public interface State {

  public void assign(String var, double value);
  public double query(String var) throws UninitializedVariableException;

}
```

14. package opgave4.parser;

```
import opgave4.tokenizer.*;

abstract class Statement extends ParseTreeNode {

   static Statement tryParse(Tokenizer tok) throws ParseException {
         Statement s;
         s = Assignment.tryParse(tok);
         if (s != null) {
            return s;
         }
         s = Query.tryParse(tok);
         if (s != null) {
            return s;
         }
         return null;
   }

   abstract void execute(State s) throws EvaluationException;
   // implemented by Assigment and Query

}
```

15. package opgave4.parser;

```
import opgave4.tokenizer.*;
import java.util.*;

class StatementList extends ParseTreeNode { // class access only

   private List<Statement> statements;

   StatementList() { // constructor
         statements = new LinkedList<Statement>();
   }

   private void addStatement(Statement s) { // private method
         statements.add(s);
   }
```

```
    static StatementList tryParse(Tokenizer tok) throws ParseException {
        StatementList sl = new StatementList();
        Statement s = null;
        s = Statement.tryParse(tok);
        while (s != null && tok.getCurrent() instanceof SemicolonToken) {
            tok.moveNext();
            sl.addStatement(s);
            s = Statement.tryParse(tok);
        }
        if (s != null && !(tok.getCurrent() instanceof SemicolonToken)) {
            reportError("; expected, but found " + tok.getCurrent());
        }
        return sl;
    }

    void execute(State s) throws EvaluationException {
        for (Statement stmt : statements) {
            stmt.execute(s);
        }
    }

}
```

16.
```
package opgave4.parser;
import opgave4.tokenizer.*;

abstract class Value extends ExprNode {

    static Value tryParse(Tokenizer tok) {
        Var v = Var.tryParse(tok);
        if (v != null) {
            return v;
        }
        Num n = Num.tryParse(tok);
        if (n != null) {
            return n;
        }
        return null;
    }

    // evaluate is overgeerfd.


}
```

17.
```
package opgave4.parser;
import opgave4.tokenizer.*;

class Var extends Value {
```

```
    private String var;

    Var(String v) {
        var = v;
    }

    static Var tryParse(Tokenizer tok) {
        if (tok.getCurrent() instanceof VarToken) {
            VarToken vt = (VarToken) tok.getCurrent();
            tok.moveNext();
            return new Var(vt.getName());
        }
        return null;
    }

    double evaluate(State s) throws UninitializedVariableException {
        return s.query(var);
    }

}
```

De Tokenizer:

```
1. package opgave4.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();
        }
```

```
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('/');
   st.ordinaryChar('-');
   moveNext();
}

private Token makeNext() {
        try {
                st.nextToken();
        } catch (IOException e) {
                return new InvalidToken("<IOException>");
        }
        switch (st.ttype) {
                case StreamTokenizer.TT_EOF:
                   return new EOFToken();
              case StreamTokenizer.TT_NUMBER:
                   return new NumberToken(st.nval);
                case StreamTokenizer.TT_WORD:
                   if (st.sval.equals("SET")) {
                        return new SetToken(st.sval);
```

```
                        } else if (st.sval.equals("GET")) {
                            return new GetToken(st.sval);
                    }
                        return new VarToken(st.sval);
                    default:
                        char ch = (char) st.ttype;
                        switch (ch) {
                          case ';' : return new SemicolonToken(ch);
                          case '=' : return new AssignmentToken(ch);
                          case '+' : return new AddToken(ch);
                          case '-' : return new SubtToken(ch);
                  case '*' : return new MultToken(ch);
                  case ':' : return new DivToken(ch);
                          default  : return new InvalidToken(ch);
                 }

            }
        }

}
```

2. package opgave4.tokenizer;

```
public abstract class Token {

        private String originalString;

        public Token(String s) {
                originalString = s;
        }

        public Token(char ch) {
            originalString = Character.toString(ch);
        }

        public String toString() {
                return originalString;
        }

}
```

3. package opgave4.tokenizer;

```
public class VarToken extends Token {

        private String name;

        public VarToken(String s) {
                super(s);
                name = s.toLowerCase();
```

```
        }

        public String getName() {
                return name;
        }

}
```

4.

```
package opgave4.tokenizer;

public class SubtToken extends OpToken {

      public SubtToken(char ch) {
            super(ch);
      }

}
```

5.

```
package opgave4.tokenizer;

public class SetToken extends Token {

      public SetToken(String s) {
            super(s);
      }

}
```

6.

```
package opgave4.tokenizer;

public class OpToken extends Token {

   private char operator;

   public OpToken(char ch) {
         super(ch);
         operator = ch;
   }

   public char getOperator() {
         return operator;
   }

}
```

7. package opgave4.tokenizer;

```
public class SemicolonToken extends Token {
	public SemicolonToken(char op) {
		super(op);
	}
}
```

8.
```
package opgave4.tokenizer;

public class NumberToken extends Token {

	private double value;

	public NumberToken(double v) {
		super(Double.toString(v));
		value = v;
	}

	public double getValue() {
		return value;
	}

}
```

9.
```
package opgave4.tokenizer;

public class MultToken extends OpToken {

	public MultToken(char ch) {
		super(ch);
	}

}
```

10.
```
package opgave4.tokenizer;

public class GetToken extends Token {

	public GetToken(String s) {
		super(s);
	}

}
```

11.
```
package opgave4.tokenizer;

public class InvalidToken extends Token {

	public InvalidToken(String s) {
```

```
		super(s);
	}

	public InvalidToken(char ch) {
		super(ch);
	}

}
```

12.
```
package opgave4.tokenizer;

public class EOFToken extends Token {

	public EOFToken() {
		super("");
	}

	public String toString() {
		return "<EOF>";
	}

}
```

13.
```
package opgave4.tokenizer;

public class DivToken extends OpToken {

	public DivToken(char ch) {
		super(ch);
	}

}
```

14.
```
package opgave4.tokenizer;

public class AddToken extends OpToken {

	public AddToken(char ch) {
		super(ch);
	}

}
```

15.
```
package opgave4.tokenizer;

public class AssignmentToken extends Token {
	public AssignmentToken(char op) {
```

```
            super(op);
        }
}
```

De klasse Main:

```
package opgave4;

import opgave4.tokenizer.*;
import opgave4.parser.*;

import java.io.*; // vergeet niet de i/o klassen te importeren

public class Main {
    public static void main(String[] args) {

        if (args.length != 1) {
            System.out.println("Usage: java opgave4.Main <programfile>");
            System.exit(-1);
        }

        try {
            BufferedReader br = new BufferedReader(new FileReader(args[0]));

            Tokenizer tok = new Tokenizer(br);
            // Tokenizer tok = new Tokenizer(System.in);

            Program p = Program.tryParse(tok);
            if (p != null) {
                State s = new HashtableState();
                p.execute(s);
            } else {
                throw new ParseException("Program expected");
            }
        } catch (IOException ie) {
            System.out.println("Error during I/O \n " + ie);
        } catch (ParseException pe) {
            System.out.println("Exception during parsing \n" + pe);
        } catch (EvaluationException ee) {
            System.out.println("Exception during execution \n" + ee);
        }
    }
}
```