Aufgabe 3.1

(1) Aufgabe

```
1  module changeorientation(degree : Real)
2  var abs : Real;
3  abs := orientation();
4  if degree ≠ abs
5     left(degree − abs);
6  endif
7  endmodule
```

(2) Aufgabe

```
1  module movetopoint(x, y : Real)
2  var currOrientation, currX, currY : Real;
3  currOrientation := orientation();
4  currX := xpos();
5  currY := ypos();
6  changeorientation(0);
7  move(x − currX);
8  changeorientation(90);
9  move(y − currY);
10 changeorientation(currOrientation);
11 endmodule
```

Aufgabe 3.2

```
1  import sun.rmi.runtime.Log;
2
3  import java.awt.*;
4  import java.awt.event.*;
5
6  public class Robot extends Frame {
7      Graphics g;
8      /* Offset for painting area, such that (0,0) is in the middle  */
9      final static int offset = 360;
10     //     final static int xoffset = 510;
11     final static int scalefactor = 100;
12
13     /* Constructor */
14     public Robot() {
15         setTitle("Picture-Drawing Robot");
16         setSize(700, 700);
17         addWindowListener(new WindowAdapter() {
18             @Override
19             public void windowClosing(WindowEvent e) {
20                 System.exit(0);
21             }
22         });
23     }
24
25     private int convert(double x) {
```

```
26          return new Double(offset + x * scalefactor).intValue();
27      }
28
29      private void drawL(double x1, double y1, double x2, double y2) {
30          g.drawLine(convert(x1), convert(y1 * -1), convert(x2), convert(y2 * -1));
31      }
32
33
34      /* State of the Picture-Drawing Robot */
35      private double orientation = 0;
36      private double xpos = 0;
37      private double ypos = 0;
38      private boolean down = false;
39
40      /* Operations on the Robot */
41      public double orientation() {
42          return orientation;
43      }
44
45      public double xpos() {
46          return xpos;
47      }
48
49      public double ypos() {
50          return ypos;
51      }
52
53      public void move(double x) {
54          double newx, newy;
55          newx = xpos + Math.cos(orientation * Math.PI / 180) * x;
56          newy = ypos + Math.sin(orientation * Math.PI / 180) * x;
57          if (down) {
58              drawL(xpos, ypos, newx, newy);
59          }
60          xpos = newx;
61          ypos = newy;
62      }
63
64      public void left(double x) {
65          orientation += x;
66          if (orientation >= 360)
67              orientation -= 360;
68          //System.out.println("New Orientation is: "+orientation);
69      }
70
71      public void right(double x) {
72          orientation -= x;
73          if (orientation < 0)
74              orientation += 360;
75          //System.out.println("New Orientation is: "+orientation);
76      }
77
78      public void raisepen() {
79          down = false;
80      }
81
82      public void lowerpen() {
83          down = true;
84      }
85
86      /* In the methods below we will only make use
87         of the following methods and methods defined
88         using them.
89         - orientation()
90         - xpos()
91         - ypos()
92         - move(double x)
93         - left(double x)
94         - right(double x)
95         - raisepen()
```

```
96          - lowerpen()
97      */
98
99      /* Drawpolygon example from the lecture */
100     public void drawpolygon(double size, int n) {
101         lowerpen();
102         for (int i = 0; i < n; i++) {
103             move(size);
104             left(360 / n);
105         }
106         raisepen();
107     }
108
109     /* Implementation of the algorithm of exercise task 3.1 (1) */
110     /*
111      * First of make x between 0 and 360, then check the difference in degrees
112      * between the current state and the end state, and turn accordingly.
113      * to make it less time consuming (considering it is a real robot) we consider two cases
114      * Either it is faster to turn to the left or right.
115      */
116     public void changeorientation(double x) {
117         double abs = orientation();
118         if (x != abs) {
119           left(x - abs);
120         }
121     }
122
123     /* Implementation of the algorithm of exercise task 3.1 (2) */
124     /*
125      * This function is supposed to let the point move to the given coordinates without
126      * changing orientation.
127      * First of we set it facing the X coordinates and we move to the desired point, then
128      * we do the same for the Y coordinates. (P.S. this would work even if the point is lower).
129      */
130     public void movetopoint(double x, double y) {
131         double currentOrientation = orientation();
132         double currentX = xpos(), currentY = ypos();
133         changeorientation(0);
134         move(x - currentX);
135         changeorientation(90);
136         move(y - currentY);
137         changeorientation(currentOrientation);
138     }
139
140     /* Implementation of the algorithm of tutorial task 2.1 */
141     public void nikolaus(double x) {
142         /* TODO: Enter your solution here */
143     }
144
145
146     @Override
147     public void paint(Graphics g) {
148         /* Initialize Robot */
149         this.g = g;
150         orientation = 0;
151         xpos = 0;
152         ypos = 0;
153         down = false;
154         /* Initialize Coordinate System*/
155         drawL(-3.1, 0, 3.1, 0);
156         drawL(0, -3.1, 0, 3.1);
157         for (int i = -3; i <= 3; i++) {
158             drawL(i, 0.05, i, -0.05);
159             drawL(0.05, i, -0.05, i);
160         }
161
162         /* Test programm, drawing one shape in each sector of
163            the coordinate system. */
164         movetopoint(1, 1);
165         drawpolygon(1, 5);
```

```
166
167         movetopoint(-2, 1);
168         drawpolygon(1.5, 3);
169
170         movetopoint(-2, -2);
171         drawpolygon(1.5, 4);
172
173         movetopoint(1, -2);
174         drawpolygon(1, 6);
175     }
176
177     public static void main(String[] args) {
178         new Robot().setVisible(true);
179     }
180 }
```

Aufgabe 3.3

(1) statement ::= assignment | sequence | conditional | iteration | return expression

(2) expression ::= arithmeticexpr | booleanexpr | modulecall

(3) modulecall ::= name(expressionlist) | name(expression)
expressionlist ::= expression | expression, expressionlist