Informatik II 12.05.2007
Andreas Krienke Dominik Deichsel
Matrikel-Nr.: 517903 Matrikel-Nr.: 517946
Internationale Medieninformatik
FHTW Berlin

Lab report X3 – Backtracking and Distributed Computing

Exercice description:

http://www.tfh-berlin.de/~ischmied/Inf2/Exercises/X3_Backtracking_and_Distribution.html

Work description:

Understanding problem solving by backtracking and learning to use distributed computing to trade computing power for complexity.

1. Preperation Questions

1. *2+9=8 … Obviously, it is arithmetically wrong, but it can be corrected by moving a single match. Which one?*

   From the nine we moving the upper right stick to the lower left position. Now it is arithmetically correct.
   2+6=8

2. *In backtracking, a tree-shaped search structure evolves. If you work iteratively, you have to define a data structure to store the search tree. Why is this not required if you work recursively?*

   Each recursive step knows everything about the values to be calculated and goes down to the simplest solution and need not store all values.

3. *Explain how the time complexity of a problem can be reduced if you use distributed computing.*

   Because of the fact that in distributed computing it is possible to run different parts of a program simultaneously the walk through search tree takes less time. And all different parts of the program have their independent result.

Lab experiments

1. *Write a backtracking program to solve a riddle of the form* ***Digit Operator Digit = Digit*** *with the digits and the operator being given as user input.*

1.1 *Start with an application which reads the equation as calling parameters or console input.*

We used the Class Scanner (`java.util.Scanner`), a simple text scanner which can parse primitive types and strings using regular expressions, to reads the equation as console input. After the "read in" we call the DigitRiddle constructor with the entered equation as parameters.

```
Scanner console = new Scanner(System.in);
System.out.println("Please enter an equation as\n <digit> <operator> <digit> = <digit>");

String d1 = console.next();
String op = console.next();
String d2 = console.next();
console.next(); // Gleichheitszeichen überspringen
String d3 = console.next();

dr = new DigitRiddle(d1,op,d2,d3);
```

Figure 1: Usage of Scanner class

1.2 Define and fill a data structure (e.g. array of vectors) which for every digit enumerates the transformations achievable by

-1 : removing a match
0 : moving a match
+1 : adding a match.

We decided to invent a class called `Transformations to build and fill the multidimensional vector with all possible solutions for every number. The Transformations Vector is initialised in the constructor of the DigitRiddle class.`

```
private void fillTransformations()
{
      Vector digit = new Vector();
      Vector transformations = new Vector();

      // START adding transformations for 0
      transformations.add(8);
      digit.add(transformations.clone());
      transformations.clear();

      transformations.add(6);
      transformations.add(9);
      digit.add(transformations.clone());
      transformations.clear();

      transformations.add("");
      digit.add(transformations.clone());
      transformations.clear();

      digitNumbers.add(digit.clone());
      // END adding transformations for 0

      // ...
```

Figure 2: Extract of fillTransformations method

1.3 Find a recursive backtracking algorithm which runs through the (imaginary) search tree

It was very hard work to create a recursive backtracking algorithm. It was a big problem to imagine the recursive search tree algorithm. The first step was to used the hints that was given. We build a search method called searchTree.

```
public boolean searchTree(String[] equation, int position, boolean up, boolean down)
```

Figure 3: recursive method header

The method gets the actual equation, the actual position of the digit which should be worked on and two boolean parameters. One indicates that a match is taken away (up) while the other indicates that a match is added to the equation (down).

The recursion ends when either both boolean values are true (two matches moved) or the position pointer is behind the last digit (e.g. > 4). If the incoming equation is arithmetically correct the method returns true, otherwise false.

```
// two moves done -> check and return
if (up && down) {

	// check equation
	if(checkEquation(equation)) {

		// store correct result in a vector
		storecorrectResult(equation.clone());

		return true;
	}
	return false;
}
```

Figure 4: checking for terminating condition

Checking the equation for being correct we swapped out this fragment into an own method which simply loops through our result vector. The current equation is added only if it is not already contained.

```
private void storecorrectResult(String[] correctEquation) {

	boolean stored = false;

	for (int i = 0; i < this.correctEquations.size(); i++) {

		String[] result = (String[]) this.correctEquations.get(i);

		if (result[0].equals(correctEquation[0]) &&
			result[1].equals(correctEquation[1]) &&
			result[2].equals(correctEquation[2]) &&
			result[3].equals(correctEquation[3])
		) {
			stored = true;
		}
	}

	// store correct result in global vector if it is not
	// already stored
	if (!stored) this.correctEquations.add(correctEquation);
}
```

Figure 5: method for checking of arithmetical correctness of given equation

Back to our recursive method we start checking the states of both boolean values. If for example „up“ is false we can still take one match away from the equation. So, for the current digit at the given position we retrieve all possible transformations from our transformations vector. Getting a transformation the equation will be changed and the new digit is set instead of the original one. Additionally the position is incremented and the boolean values are set respectively.

```
if (!up) {
```

```
// get possible digits for adding a stick
digitVector = ((Vector)digitTransformations.get(2));

// call with all possible equations
for (int k = 0; k < digitVector.size(); k++) {

      if (!digitVector.get(k).toString().equals("")) {

            // store equation
            String s = equation[position];

            // set new equation
            equation[position] = digitVector.get(k).toString();

            // recursive call
            result = searchTree(equation, position+1, true, down);

            if (!result) {
                  equation[position] = s;
                  searchTree(equation, position+1, up, down);
            }
            equation[position] = s;
```

Figure 6: Extract of recursive method – changing digit in equation and recursive call

Then the method calls itself recursively with these new values returning either true if a correct result is found in this branch or false if not.
The most important thing here is that we had to store the current equation temporary before changing it. This was one of our major problems why the program did not work correct until we added this functionality.

Afterwards the prior stored equation has to be set back as we want to use it for the other boolean "down" and for both.

```
Problems | Javadoc | Declaration | Console
<terminated> DigitRiddle (1) [Java Application] /System
Please enter an equation as
 <digit> <operator> <digit> = <digit>
3 + 6 = 0
Starting equation: 3+6=0
Correct equation no.1: 3+5=8
Correct equation no.2: 3+6=9
```

Figure 7: Final console output of results

Voluntary work:

After the mandatory part which took a long time of preparation we tried the voluntary work. Here developing an applet is still some fun work as we can programming the user interface and its interaction.

So we started by adding some comboboxes for the equation input. This is also good in preventing input errors.
After clicking the calculate button an instance of our DigitRiddle class is used to retrieve possible transformations. The result vector then is used to display the first one as matches and the others as simple text at the bottom.

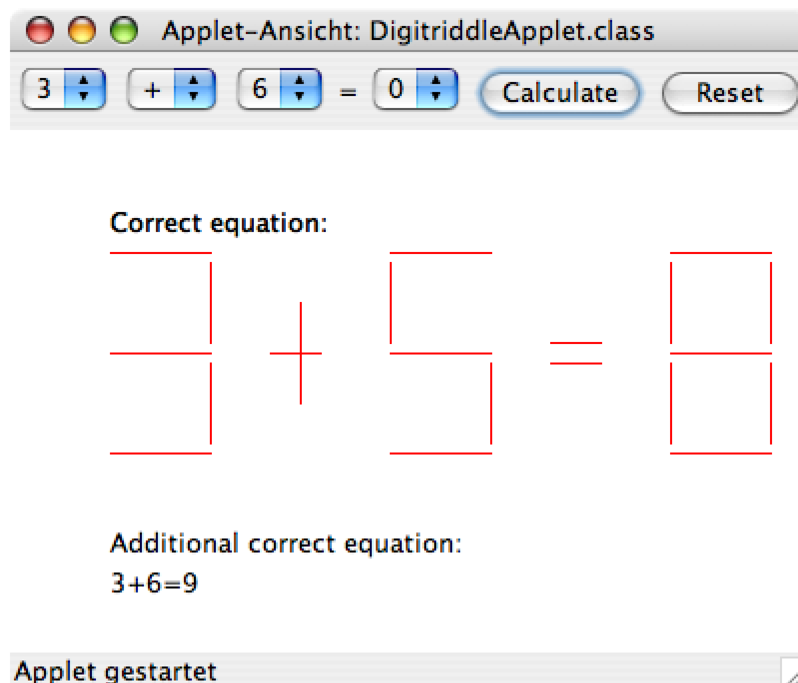

Figure 8: First applet implementation

This was quite easy work straight forward. So we tried to do it in a more sophisticated way by showing the user which match was moved. Therefor our consideration was to draw the starting equation in bold lines while the result one is drawn in small lines. But this looked strange and it was not obvious what should be shown.
So we did a little bit research and found out that the Graphics2D class has some functionality to draw things with a transparent effect. This is what we finally did. Drawing the starting equation in red and the result in green with a kind of transparency we got the following output:

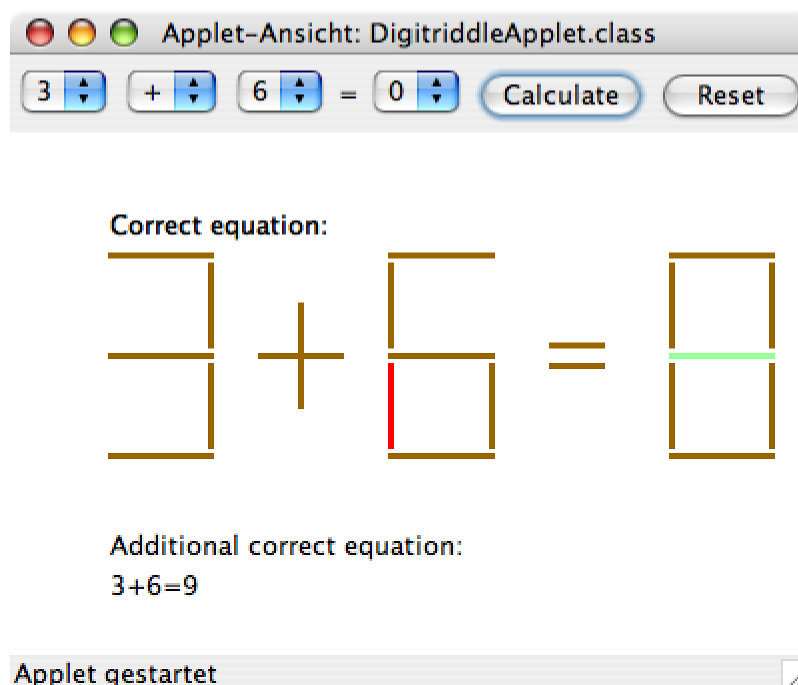

Figure 9: Final DigitRiddleApplet –
starting and result equation with transparency

What we got is a „brown“ equation **plus** the green mach (3+5=8). The red one was taken away.

Time usage:

Also we described that we spend a lot of time for our reports – this time it was a lot more. I cannot explain detailed why but I think it was due to the fact that none of us could simply see what was the key point of lab task 1. Only as you at the end of the lab time and later our tutor Maria explained what the task should be we saw a possible solution and invented it with some minor problems on our own.

But this was also the reason why we did not do task 2 this time.

Source codes:

This time we created a website with the described applet and links to our source code files. It can be found here: http://www.f4.fhtw-berlin.de/~s0517903/Info/lab3/x3_applet.html

Appendix:

Attention:

Please have in mind that the appended source codes are only for completeness reasons. These can also be downloaded from the mentioned website.

1. **Link to website which includes applet**

   http://www.f4.fhtw-berlin.de/~s0517903/Info/lab3/x3_applet.html

2. **DigitRiddle.java**

   Code of DigitRiddle.java:

```
import java.util.Scanner;
import java.util.Vector;

public class DigitRiddle {

	// main vector holding all digit transformations
	Transformations transformations;

	// vector of found correct equations
	Vector correctEquations;

	// given equation, stored for reference
	String[] myFormula;

	// define variables for input digits
	String d1, d2, d3, op;

	public DigitRiddle() {
		this("3","+","6","0");
	}

	public DigitRiddle(String d1, String op, String d2, String d3) {
		this.d1 = d1;
		this.d2 = d2;
		this.op = op;
		this.d3 = d3;

		this.transformations = new Transformations();

		this.correctEquations = new Vector();

		// store as formula
		this.myFormula = new String[]{this.d1,this.op,this.d2,this.d3};
	}

	public static void main(String[] args) {

	   DigitRiddle dr;

	   if (args.length == 5) {
	      // get program arguments as equation parts
	      dr = new DigitRiddle(args[0],args[1],args[2],args[4]);
	   }
```

```
    else {
      // get input from console
      Scanner console = new Scanner(System.in);
      System.out.println("Please enter an equation as\n <digit>
                          <operator> <digit> = <digit>");

      String d1 = console.next();
      String op = console.next();
      String d2 = console.next();
      console.next(); // Gleichheitszeichen überspringen
      String d3 = console.next();

      dr = new DigitRiddle(d1,op,d2,d3);
     }

      // string arrays will be referenced, so get a standalone one
      String[] form = dr.myFormula.clone();

      // call recursive method to found correct equations
      dr.searchTree(form, 0, false, false);

      // print starting equation
      System.out.println("Starting equation: " +
                          dr.myFormula[0] +
                          dr.myFormula[1] +
                          dr.myFormula[2] + " = " +
                          dr.myFormula[3]);

      // print found solutions
      dr.printFoundEquations();
  }

  public void searchTree() {
      this.searchTree(new String[]{this.d1,this.op,this.d2,this.d3},
                       0, false, false);
  }

  public boolean searchTree(String[] equation, int position,
                          boolean up, boolean down)
  {
      int digit;
      Vector digitVector;
      boolean result;

      // two moves done -> check and return
      if (up && down) {

            // check equation
            if(checkEquation(equation)) {

                  // store correct result in a vector
                  storecorrectResult(equation.clone());

                  return true;
            }
            return false;
      }

      // all digits done
      if (position >= 4) {
            return true;
      }
```

```
        // get digit from formula
        digit = getDigitFromEquation(equation.clone(), position);

        // get digit row with all transformations
        Vector digitTransformations = (Vector)
                            transformations.digitNumbers.get(digit);

        if (!up) {

          // get possible digits for adding a stick
          digitVector = ((Vector)digitTransformations.get(2));

          // call with all possible equations
          for (int k = 0; k < digitVector.size(); k++) {

              if (!digitVector.get(k).toString().equals("")) {

                    // store equation
                    String s = equation[position];

                    // set new equation
                    equation[position] = digitVector.get(k).toString();

                    // recursive call
                    result = searchTree(equation, position+1, true, down);
                    if (!result) {
                          equation[position] = s;
                          searchTree(equation, position+1, up, down);
                    }
                    equation[position] = s;
              }
          }
        }

        if (!up && !down) {

          // get possible digits for changing a stick
          digitVector = ((Vector)digitTransformations.get(1));

              // call with all possible equations
              for (int k = 0; k < digitVector.size(); k++) {

                if (!digitVector.get(k).toString().equals("")) {

                    // store equation
                    String s = equation[position];

                    // set new equation
                    equation[position] = digitVector.get(k).toString();

                    // recursive call
                    result = searchTree(equation, position+1, true, true);
                    //System.out.println(result);
                    if (!result) {

                          equation[position] = s;
                          searchTree(equation, position+1, up, down);
                    }
                    equation[position] = s;
                 }
              }
        }
```

```
        if (!down) {

              // get possible digits for removing a stick
              digitVector = ((Vector)digitTransformations.get(0));

              // call with all possible equations
              for (int k = 0; k < digitVector.size(); k++) {

                if (!digitVector.get(k).toString().equals("")) {

                    // store equation
                    String s = equation[position];

                    // set new equation
                    equation[position] = digitVector.get(k).toString();

                    // recursive call
                    result = searchTree(equation, position+1, up, true);

                    if (!result) {
                          equation[position] = s;
                          searchTree(equation, position+1, up, down);
                    }
                    equation[position] = s;
                }
              }
        }

        return false;
  }

  private boolean checkEquation(String[] eq)
  {
    if (eq[1].equals("+")) {

      if(Integer.valueOf(eq[0]) + Integer.valueOf(eq[2]) ==
         Integer.valueOf(eq[3])) {

        return true;
      }
    }
    else if (eq[1].equals("-")){

        if (Integer.valueOf(eq[0]) - Integer.valueOf(eq[2]) ==
            Integer.valueOf(eq[3])) {
         return true;
        }
    }

    return false;
  }

  private void storecorrectResult(String[] correctEquation) {

        boolean stored = false;

        // store correct result in global vector if it is not
        // already stored

        for (int i = 0; i < this.correctEquations.size(); i++) {

              String[] result = (String[]) this.correctEquations.get(i);
```

```
                  if (result[0].equals(correctEquation[0]) &&
                        result[1].equals(correctEquation[1]) &&
                        result[2].equals(correctEquation[2]) &&
                        result[3].equals(correctEquation[3])
                  ) {
                        stored = true;
                  }
            }

            if (!stored) this.correctEquations.add(correctEquation);
      }

      private int getDigitFromEquation(String[] equation, int pos) {

            int digit;

            if (equation[pos].equals("+")) {
                  digit = 10;
            }
            else if (equation[pos].equals("-")) {
                  digit = 11;
            }
            else {
                  digit = Integer.valueOf(equation[pos]);
            }

            return digit;
      }

      public void printFoundEquations() {

            for (int i = 0; i < this.correctEquations.size(); i++) {
                  String[] result = (String[]) this.correctEquations.get(i);
                  System.out.println("Correct equation no."+(i+1)+": "+
result[0]+result[1]+result[2]+"="+result[3]);
            }
      }
}
```

3. **DigitRiddleApplet.java**

Code of DigitriddleApplet.java:

```
public class DigitriddleApplet extends JApplet {

      // gui elements
      JComboBox jcoD1, jcoD2, jcoD3, jcoSign;
      JLabel text1, equalitySign;
      JPanel p1, p2, p3;
      JButton b1, b2;

      String[] items = new String[]{"0","1","2","3","4","5","6","7","8","9"};

      Graphics g;

      public void init() {

            setSize(400,300);

            this.setVisible(true);

            setLayout(new BorderLayout());
```

```
            // north panel
            p1 = new JPanel();

            // add some elements
            jcoD1 = new JComboBox(items);
            jcoSign = new JComboBox(new String[]{"+","-"});
            jcoD2 = new JComboBox(items);
            jcoD3 = new JComboBox(items);

            equalitySign = new JLabel("=");

            b1 = new JButton("Calculate");
            b2 = new JButton("Reset");


            p1.add(jcoD1);
            p1.add(jcoSign);
            p1.add(jcoD2);
            p1.add(equalitySign);
            p1.add(jcoD3);
            p1.add(b1);
            p1.add(b2);

            this.add(p1, BorderLayout.NORTH);

            b1.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent ev) {

                              //repaint();

                              // try to find new equations
                              DigitRiddle dr = new DigitRiddle(
                                          jcoD1.getSelectedItem().toString(),
                                          jcoSign.getSelectedItem().toString(),
                                          jcoD2.getSelectedItem().toString(),

                                          jcoD3.getSelectedItem().toString()
                                          );

                              //DigitRiddle dr = new DigitRiddle();
                              //System.out.println(jcoD1.getSelectedItem().to-
String() + jcoSign.getSelectedItem().toString());

                              // call recursive method
                              dr.searchTree();

                              String[] eq = null;

                              if (dr.correctEquations.size()>0) {
                                    eq = (String[])dr.correctEquations.get(0);
                                    paintEquation(eq.clone(), false);
                              }
                              else {
                                    // no correct equations found
                                    System.out.println("No possible correct
equations found");
                              }

                              // if more than one correct equation was found,
                              // give a notice
                              for (int i = 1; i < dr.correctEquations.size();
i++) {
```

```
                                    eq = (String[])dr.correctEquations.get(i);
                                    paintEquation(eq.clone(), true);
                              }


                        }
                  }
            );

            b2.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent ev) {

                        // reset gui fields
                        jcoD1.setSelectedIndex(0);
                        jcoSign.setSelectedIndex(0);
                        jcoD2.setSelectedIndex(0);
                        jcoD3.setSelectedIndex(0);


                        repaint();
                  }
            }
      );

      }

      public void paint(Graphics g) {
            //super.paint(g);
            this.g = g;
            g.setColor(Color.BLUE);
        g.clearRect (0, 35, 500, 500);
        //g.fillRect (0, 35, 500, 500);

      }

      private void paintEquation(String[] equation, boolean more) {

            Graphics2D g = (Graphics2D) this.getGraphics();

            g.setColor(Color.BLACK);
            g.drawString("Correct equation:",50,90);

            // if boolean more is true paint other equations as text
            if (more) {
                  g.setColor(Color.BLACK);
                  g.drawString("Additional correct equation:",50,250);
                  g.drawString(equation[0]+equation[1]+equation[2]+"="+equa-
tion[3],50,270);
                  return;
            }

            g.setStroke(new BasicStroke(2));

            g.setColor(Color.RED);
            drawDigit(g, jcoD1.getSelectedItem().toString().charAt(0),
"0".charAt(0));
            drawDigit(g, jcoSign.getSelectedItem().toString().charAt(0),
"1".charAt(0));
            drawDigit(g, jcoD2.getSelectedItem().toString().charAt(0),
"2".charAt(0));
            drawDigit(g, "=".charAt(0), "=".charAt(0));
            drawDigit(g, jcoD3.getSelectedItem().toString().charAt(0),
"3".charAt(0));
```

```
		g.setStroke(new BasicStroke(2));
		g.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER,
(float)0.4));
		g.setColor(Color.GREEN);
		drawDigit(g, equation[0].charAt(0), "0".charAt(0));
		drawDigit(g, equation[1].charAt(0), "1".charAt(0));
		drawDigit(g, equation[2].charAt(0), "2".charAt(0));
		drawDigit(g, "=".charAt(0), "=".charAt(0));
		drawDigit(g, equation[3].charAt(0), "3".charAt(0));

	}

	private void drawDigit(Graphics2D g, char digit, char pos) {

		int startX = 50;

		// set x coordinate depending on digit position in equation
		switch (pos) {
		case '0':
			startX = 50;
			break;
		case '1':
			startX = 120;
			break;
		case '2':
			startX = 190;
			break;
		case '=':
			startX = 260;
			break;
		case '3':
			startX = 330;
			break;
		default:
			startX = 50;
			break;
		}

		// define lines for each digit
		switch (digit) {
		case '0':
			g.drawLine(startX,100,startX+50,100);
			g.drawLine(startX,105,startX,145);
			g.drawLine(startX,155,startX,195);
			g.drawLine(startX+50,105,startX+50,145);
			g.drawLine(startX+50,155,startX+50,195);
			g.drawLine(startX,200,startX+50,200);

			break;
		case '1':
			g.drawLine(startX+50,100,startX+50,145);
			g.drawLine(startX+50,155,startX+50,195);

			break;
		case '2':
			g.drawLine(startX,100,startX+50,100);
			g.drawLine(startX+50,105,startX+50,145);
			g.drawLine(startX,150,startX+50,155);
			g.drawLine(startX,155,startX,195);
			g.drawLine(startX,200,startX+50,200);

			break;
```

```
            case '3':
                  g.drawLine(startX,100,startX+50,100);
                  g.drawLine(startX+50,105,startX+50,145);
                  g.drawLine(startX,150,startX+50,150);
                  g.drawLine(startX+50,155,startX+50,195);
                  g.drawLine(startX,200,startX+50,200);

                  break;
            case '4':
                  g.drawLine(startX,105,startX,145);
                  g.drawLine(startX,150,startX+50,150);
                  g.drawLine(startX+50,105,startX+50,145);
                  g.drawLine(startX+50,155,startX+50,195);

                  break;
            case '5':
                  g.drawLine(startX,100,startX+50,100);
                  g.drawLine(startX,105,startX,145);
                  g.drawLine(startX,150,startX+50,150);
                  g.drawLine(startX+50,155,startX+50,195);
                  g.drawLine(startX,200,startX+50,200);

                  break;
            case '6':
                  g.drawLine(startX,100,startX+50,100);
                  g.drawLine(startX,105,startX,145);
                  g.drawLine(startX,155,startX,195);
                  g.drawLine(startX,150,startX+50,150);
                  g.drawLine(startX+50,155,startX+50,195);
                  g.drawLine(startX,200,startX+50,200);

                  break;
            case '7':
                  g.drawLine(startX,100,startX+50,100);
                  g.drawLine(startX+50,105,startX+50,145);
                  g.drawLine(startX+50,155,startX+50,195);

                  break;
            case '8':
                  g.drawLine(startX,100,startX+50,100);
                  g.drawLine(startX,105,startX,145);
                  g.drawLine(startX,150,startX+50,150);
                  g.drawLine(startX,155,startX,195);
                  g.drawLine(startX+50,105,startX+50,145);
                  g.drawLine(startX+50,155,startX+50,195);
                  g.drawLine(startX,200,startX+50,200);

                  break;
            case '9':
                  g.drawLine(startX,100,startX+50,100);
                  g.drawLine(startX,105,startX,145);
                  g.drawLine(startX,150,startX+50,150);
                  g.drawLine(startX+50,105,startX+50,145);
                  g.drawLine(startX+50,155,startX+50,195);
                  g.drawLine(startX,200,startX+50,200);

                  break;
            case '+':
                  g.drawLine(startX+5,150,startX+45,150);
                  g.drawLine(startX+25,125,startX+25,175);

                  break;
            case '-':
```

```
                g.drawLine(startX+10,150,startX+35,150);

                break;
            case '=':
                g.drawLine(startX+10,145,startX+35,145);
                g.drawLine(startX+10,155,startX+35,155);

                break;
            default:
                break;
            }
        }
}
```

4. **Transformations.java**

Code of Transformations.java:

```
import java.util.Vector;

public class Transformations {

      public Vector digitNumbers;

      public Transformations() {

            // initialize main vector with 12 fields
            digitNumbers = new Vector();

            this.fillTransformations();

      }

      public void printTransformation()
      {
            for (int i=0; i < this.digitNumbers.size(); i++ )
            {
                  Vector t = (Vector)this.digitNumbers.get(i);
                  for (int k=0; k < t.size() ; k++)
                  {
                        System.out.print(t.get(k));
                  }

                  System.out.println();
            }
      }

      private void fillTransformations()
      {
            Vector digit = new Vector();
            Vector transformations = new Vector();

            // START adding transformations for 0
            transformations.add(8);
            digit.add(transformations.clone());
            transformations.clear();

            transformations.add(6);
            transformations.add(9);
            digit.add(transformations.clone());
            transformations.clear();
```

```
transformations.add("");
digit.add(transformations.clone());
transformations.clear();


digitNumbers.add(digit.clone());
// END adding transformations for 0

digit.clear();

// START adding transformations for 1
transformations.add(7);
digit.add(transformations.clone());
transformations.clear();

transformations.add("");
digit.add(transformations.clone());
transformations.clear();

transformations.add("");
digit.add(transformations.clone());
transformations.clear();

digitNumbers.add(digit.clone());
// END adding transformations for 1

digit.clear();

// START adding transformations for 2
transformations.add("");
digit.add(transformations.clone());
transformations.clear();
transformations.add(3);
digit.add(transformations.clone());
transformations.clear();

transformations.add("");

digit.add(transformations.clone());
transformations.clear();

digitNumbers.add(digit.clone());
// END adding transformations for 2

digit.clear();

// START adding transformations for 3
transformations.add(9);

digit.add(transformations.clone());
transformations.clear();

transformations.add("");

digit.add(transformations.clone());
transformations.clear();

transformations.add("");

digit.add(transformations.clone());
transformations.clear();
```

```
            digitNumbers.add(digit.clone());
            // END adding transformations for 3

            digit.clear();

            // START adding transformations for 4
            transformations.add("");

            digit.add(transformations.clone());
            transformations.clear();

            transformations.add("");

            digit.add(transformations.clone());
            transformations.clear();

            transformations.add("");

            digit.add(transformations.clone());
            transformations.clear();

            digitNumbers.add(digit.clone());
            // END adding transformations for 4

            digit.clear();

            // START adding transformations for 5
            transformations.add(6);
            transformations.add(9);

            digit.add(transformations.clone());
            transformations.clear();

            transformations.add(3);

            digit.add(transformations.clone());
            transformations.clear();

            transformations.add("");

            digit.add(transformations.clone());
            transformations.clear();

            digitNumbers.add(digit.clone());
            // END adding transformations for 5

            digit.clear();

            // START adding transformations for 6
            transformations.add(8);

            digit.add(transformations.clone());
            transformations.clear();

            transformations.add(0);
            transformations.add(9);

            digit.add(transformations.clone());
            transformations.clear();

            transformations.add(5);

            digit.add(transformations.clone());
```

```
        transformations.clear();

        digitNumbers.add(digit.clone());
        // END adding transformations for 6

        digit.clear();

        // START adding transformations for 7
        transformations.add("");

        digit.add(transformations.clone());
        transformations.clear();

        transformations.add("");

        digit.add(transformations.clone());
        transformations.clear();

        transformations.add(5);

        digit.add(transformations.clone());
        transformations.clear();

        digitNumbers.add(digit.clone());
        // END adding transformations for 7

        digit.clear();

        // START adding transformations for 8
        transformations.add("");

        digit.add(transformations.clone());
        transformations.clear();

        transformations.add("");

        digit.add(transformations.clone());
        transformations.clear();

        transformations.add(6);
        transformations.add(0);
        transformations.add(9);

        digit.add(transformations.clone());
        transformations.clear();

        digitNumbers.add(digit.clone());
        // END adding transformations for 8

        digit.clear();

        // START adding transformations for 9
        transformations.add(8);

        digit.add(transformations.clone());
        transformations.clear();

        transformations.add(6);
        transformations.add(0);

        digit.add(transformations.clone());
        transformations.clear();
```

```
            transformations.add(3);
            transformations.add(5);

            digit.add(transformations.clone());
            transformations.clear();

            digitNumbers.add(digit.clone());
            // END adding transformations for 9

            digit.clear();

            // START adding transformations for +
            transformations.add("");

            digit.add(transformations.clone());
            transformations.clear();

            transformations.add("");

            digit.add(transformations.clone());
            transformations.clear();

            transformations.add("-");

            digit.add(transformations.clone());
            transformations.clear();

            digitNumbers.add(digit.clone());
            // END adding transformations for +

            digit.clear();

            // START adding transformations for -
            transformations.add("+");

            digit.add(transformations.clone());
            transformations.clear();

            transformations.add("");

            digit.add(transformations.clone());
            transformations.clear();

            transformations.add("");

            digit.add(transformations.clone());
            transformations.clear();
            digitNumbers.add(digit.clone());

            // END adding transformations for -


      }
}
```