Introduction to Programming - Part II

Jeff Gyldenbrand
Username: jegyl16
D.O.B.: 22-11-1984
Supervisor: Jan Baumbach
DM550

8. september 2017

Indhold

1 Specification

In this assignment the tasks was to: 1. write a command line solver for Sudoku puzzles, and 2: write a graphical user interface for this solver:

The command line solver takes as argument the name of a file containing a Sudoku puzzle, and prints the solution to the screen.

For the second part of the assignment, the task was to write a GUI-class called SudukuGUI which extends the Sudoku class from the first task, and uses the recursive algorithm already written.

When the SudokuGUI starts, an window has to appear, containing a start-button, load-button, save-button, and a table for both visualization of the unsolved puzzle, and the solved one. When the load-button is pressed, the users should be able to browse their computer for the file containing the puzzle. Then it loads the file into a table. When the start-button is clicked, another table shall show the solution. And finally, when save-button is clicked, the users shall be able to save the solution as a text-file on their computers.

2 Design

The design of the program was straightforward to make, as soon as the given templates; Field.java, Sudoku.java and SolvedException.java were understood properly.

In the Sudoku-class, which contains the main-method running task 1, also contains the solve-method that runs the recursive algorithm that solves the sudoku puzzle. This method is communicating with the Field-class, which contains boolean methods that checks if a given value from 1 to 9 is valid in respectively the rows, columns and the 3x3 sized boxes within the 9x9 sized sudoku-field, and returning an answer to the algorithm, which outputs the answer in a grid-design in the console.

Now for the second part of the assignment, we implemented a new class called SudokuGUI which extends the Sudoku-class. This class has its own main-method and all the graphical components needed for the users to load, solve and save the sudoku puzzles. To make the code more transparent, the GUI-part and the file-handling is separated into two different classes; SudoduGUI and FileHandler.

Whenever the users click the load or save-button from the SudokuGUI-class, it calls the respectively method in FileHandler-class, which then returns the loaded file into a table, or saves the solution.

Note that the start-button only can be pressed when the users have loaded a sudoku-file, and only save the solution, when the start-button has been pressed, and a solution is found.

When the start-button is pressed, the Field-class is called to read the sudoku-file, and the solve-method, which is extended from the Sudoku-class finds the solution and returns it to be visualized in a table.

Installation:
All code is written in Eclipse IDE, and therefore it is recommended to test it in this IDE.

1: Create a new Java project, give the project a descriptive name.

2: Go into 'files'-folder and copy the 5 java-files and 3 text-files into the 'src'-folder in the newly created project. This will automatically create a default package to the different classes.

3: Now run Sudoku, which solves the test1.txt. Feel free to change the path to src/test2.txt or src/test3.txt, and try the different sudoku-puzzles.

4: Finally, run SudokuGUI and load test1.txt from the files folder. Click start and see the solution. Now try saving it to the computer by clicking save.

3 Implementation

The recursive algorithm in solve-method checks if the first cell is empty, then call tryValue with val = 1 and check if its a valid value.

if its valid, it places the value in the cell, do a recursive call on the function and continue. if its not, it iterate val with 1, set the cell to empty, and try again, as long as the number is less than 9.

But if the cell has a value other than empty, it iterate i with one, and do it all again in the next row. When i reaches 8 (all rows done), it iterate j with 1 (next column), and sets i = 0 (starts from the top). when 8th column is reached, it prints out the Field f (the solution).

```
1  public static void solve(Field f, int i, int j) throws SolvedException {
2
3   int SIZE = Field.SIZE;
4   int val = 1;
5
6  do {
7          if(f.model[i][j] == 0) {
8
9          do {
10                 if(f.tryValue(val, i, j)) {
11
12                         f.model[i][j] = val;
13                         solve(f, i, j);
14                  }
15                 val++;
16
17         } while(val <= SIZE);
```

```
18
19          f.clear(i, j);
20          return;
21          }
22          else if(i < 8) {
23                  i++;
24          }
25          else if((i == 8) && (j < 8)) {
26                  i = 0;
27                  j++;
28           }
29           else {
30           System.out.println(f);
31           .
32           .
33           .
```

As said, the algoritm calls the tryValue-method with a value from 1 to 9, and this method is checking for boolean values in; IsEmpty, checkRow, checkCol and checkBox.

For example:

```
1  private boolean checkRow(int val, int i) {
2
3          for(int n = 0; n < SIZE; n++) {
4                  if(model[i][n] == val) {
5                          return false;
6                  }
7          }
8        return true;
9   }
```

The above code is taking as parameter the value from the algorithm, and checking the i'th row, then looping it through the row to see if the value is already there. If it is, it returns false otherwise it returns true.

Its the same procedure for the checkCol-method. But slightly different for the isEmpty-method:

```
1  public boolean isEmpty(int i, int j) {
2
3          for(int n = 0; n < SIZE; n++) {
4                  for (int m = 0; m < SIZE; m++) {
5                          if(model[i][j] == 0) {
6                                  return true;
7                          }
8                  }
9          }
10       return false;
11  }
```

To check if a cell in the sudoku-field (model) is empty, we have to loop through

every single cell. This is done by creating two for-loop within each other and see if model is equal to 0. If it is, we return true, otherwise we return false.

The checkBox-method however, is a little more complicated. In order to check 3x3 boxes, we have to define an lower and upper boundary for each 'box' when looping through the cells.

This is done by if-statements: If the given parameter i is less than 3 and larger or equal to 0, then we define two variables: iLow = 0, and iTop = 3. But if i is less than 6 and larger or equal to 3, then we define de variables iLow = 3, and iTop = 6, and so forth. Same procedure for the parameter j. Finally we loop through the sudoku-field (model) with two for-loops within each other. First one saying, n equals iLow, and as long as n is less than iTop, then iterate with 1. For each of these, we set m equals to jLow, and as long as m is less than jTop, iterate m with 1. Now, if the parameter val, with a value from 1 to 9 exist in this 3x3 field, then it returns false, otherwise it returns true.

```
1  private boolean checkBox(int val, int i, int j) {
2
3          int iLow, iTop, jLow, jTop;
4
5                if((i < 3) && (i >= 0)) {
6                       iLow = 0;
7                       iTop = 3;
8                }
9                       else if((i < 6) && (i >= 3)) {
10                             iLow = 3;
11                             iTop = 6;
12                      }
13                             else {
14                                    iLow = 6;
15                                    iTop = 9;
16                             }
17
18               if((j < 3) && (j >= 0)) {
19                      jLow = 0;
20                      jTop = 3;
21               }
22                      else if((j < 6) && (j >= 3)) {
23                             jLow = 3;
24                             jTop = 6;
25                      }
26                             else {
27                                    jLow = 6;
28                                    jTop = 9;
29                             }
30
31               for(int n = iLow; n < iTop; n++) {
32                      for (int m = jLow; m < jTop; m++) {
33                             if(model[n][m] == val) {
34                                    return false;
35                             }
36                      }
```

```
37                  }
38              return true;
39      }
40  }
```

4 Testing

To test if the command line solver works, and output the correct answer, we try out the three given test text-files; test1.txt, test2.txt and test3.txt:

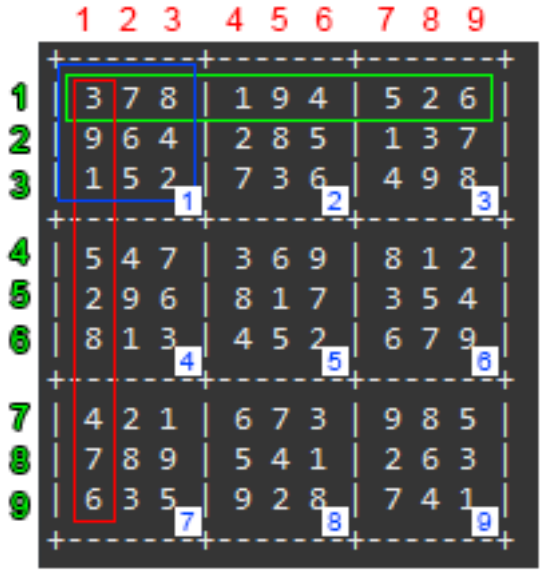
The first testfile looks like this:

```
X X 8 1 X X 5 X X
9 6 X X 8 X X X X
X 5 X X 3 6 X 9 8
X X 7 X 6 9 X 1 2
X X 6 8 X 7 3 X X
8 1 X 4 5 X 6 X X
4 2 X 6 7 X X 8 X
X X X X 4 X X 6 3
X X 5 X X 8 7 X X
```

and gives the output:

```
+-------+-------+-------+
| 3 7 8 | 1 9 4 | 5 2 6 |
| 9 6 4 | 2 8 5 | 1 3 7 |
| 1 5 2 | 7 3 6 | 4 9 8 |
+-------+-------+-------+
| 5 4 7 | 3 6 9 | 8 1 2 |
| 2 9 6 | 8 1 7 | 3 5 4 |
| 8 1 3 | 4 5 2 | 6 7 9 |
+-------+-------+-------+
| 4 2 1 | 6 7 3 | 9 8 5 |
| 7 8 9 | 5 4 1 | 2 6 3 |
| 6 3 5 | 9 2 8 | 7 4 1 |
+-------+-------+-------+
```

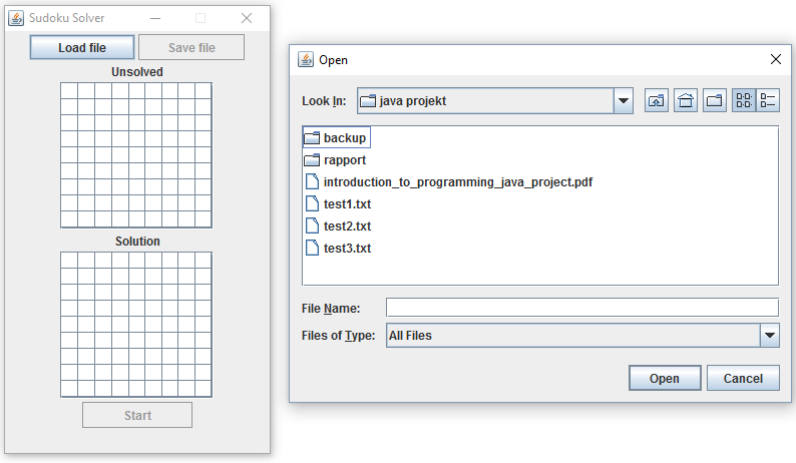
By going through each row, coloumn and 3x3 boxes, checking if they respectively containts the value 1 to 9, we can conclude that the output is correct.



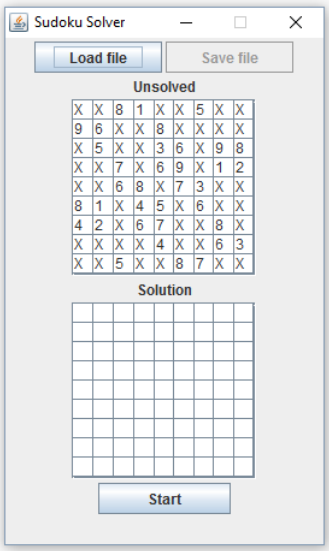
This is done with all three test-files, and all of them giving a valid output.

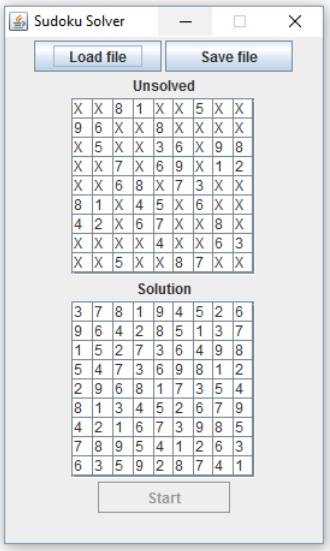
Now we test if the sudoku-part of the program works as well:

The program starts op fine, with all components visible. When load-button is clicked, JFileChooser opens up the dialogwindow, and we can choose the desired textfile.

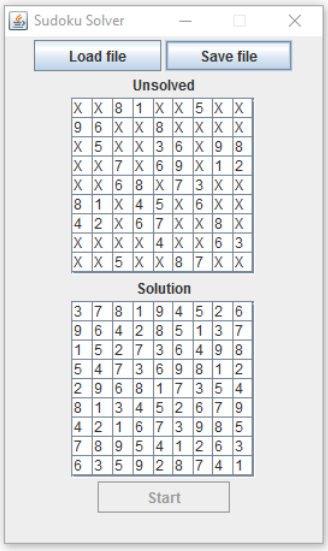


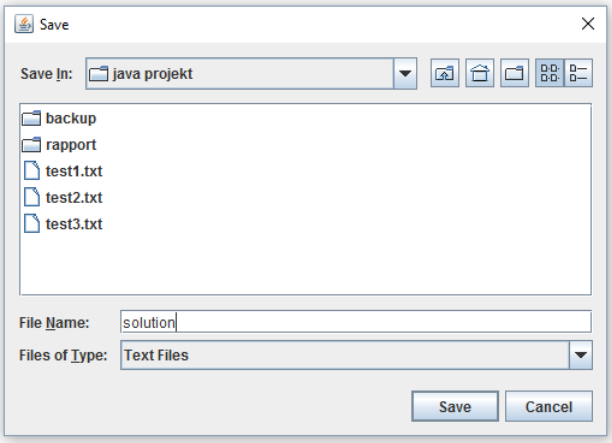
Then the first table is loaded with the data from the textfile, and the start-button is enabled. When start-button is clicked, the solution is displayed in the second table, and we are now able to click save.



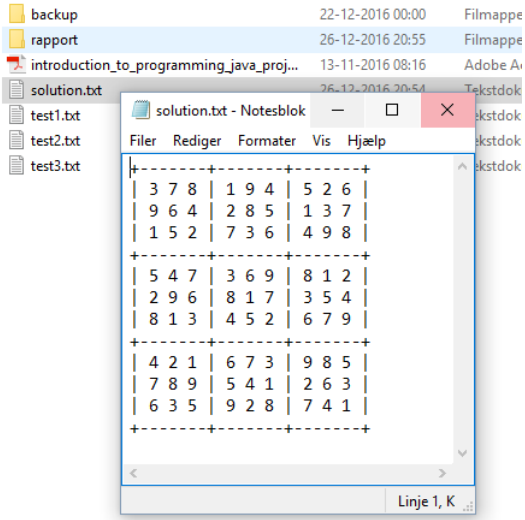


Now we can choose the desired destination for the file to be saved, and we can give the file a name, in this example we use "solution". The program automatically append a ".txt"to the end of the file name, so it is saved as a text-file.





As we see, it succesfully creates a file named solution.txt containing the solution.



5 Conclusion

So we can conclude that both the command line solver and the GUI-part of the assignemt works, and outputs the correct answers. Furthermore we can conclude that we can save the solution to a desired destination with a desired name of our choosing. And when we open this file, it contains the correct output with the grid-design provided in the template class; Field.

So all requirements of the assignment has been met.

We could of course make the program a little better by, make sure that JFile-Chooser provided a warning if a user tried to load a file that does not contain a Sudoku puzzle. Because, if a user loads a different kind of file or a text file with other content than intended, the program will either prompt an error, or load data that cant be used in the solver.

6 Appendix (source code)

The Sudoku class:

```
1  import javax.swing.JPanel;
2
3  @SuppressWarnings("serial")
4  public class Sudoku extends JPanel {
5
6          public static void main(String[] args) {
7            Field field = new Field();
8            field.fromFile("src/test1.txt");
9            try {
10             solve(field, 0, 0);
11           } catch (SolvedException e) {}
12
13         }
14
15         public static void solve(Field f, int i, int j) throws
               SolvedException {
16
17                 int SIZE = Field.SIZE;
18                 int val = 1;
19
20                 // if first cell is empty, then call tryValue with val =
                       1 and check if its a valid value.
21                 // if its valid, place the value in the cell, do a
                       recursive call on the function and continue.
22                 // if its not, iterate val with one, set the cell to
                       empty, and try again, as long as the number is less
                       than 9.
23
24                 // But if the cell has a value other than empty, iterate
                        i with one, and do it all again in the next row
25                 // when i reaches 8 (all rows done), we iterate j with
                       one (next colum), and set i = 0 (starts from the top
                       ).
26                 // when 8th colum is reached, print out the Field f (the
                        solution).
27
28                 do {
29                                 if(f.model[i][j] == 0) {
30
31                                 do {
32
33                                         if(f.tryValue(val, i, j)) {
34                                                 f.model[i][j] = val;
35                                                 solve(f, i, j);
36                                          }
37                                         val++;
38
39                                 } while(val <= SIZE);
40
41                                 f.clear(i, j);
```

```
42                          return;
43
44                          }
45                          else if(i < 8) {
46
47                                  i++;
48                          }
49                          else if((i == 8) && (j < 8)) {
50
51                                  i = 0;
52                                  j++;
53                           }
54                           else {
55
56                                  System.out.println(f);
57
58                                  // converting the Field f-solution
                                        to strings
59                                  String solution = f.toString();
60                                  String solutionWithGrid = f.
                                        toString();
61
62                                  // getting rid of everything but
                                        the numbers
63                                  solution = solution.replace("-", ""
                                        ).replace("+", "").replace("|",
                                         "").replace("\n", "").replace(
                                        " ", "");
64
65                                  // replacing '\n' with '\r\n' so
                                        the FileHandler can output
                                        correctly.
66                                  solutionWithGrid = solutionWithGrid
                                        .replace("\n", "\r\n");
67
68                                  // creating arrays
69                                  String[] solutionArray1D = solution
                                        .split("");
70                                  String solutionArray2D[][] = new
                                        String[9][9];
71
72                                  for(int n = 0; n < 9; n++) {
73                                           for(int m = 0; m < 9; m++)
                                                 {
74                                               solutionArray2D[n][m]
                                                    = solutionArray1D
                                                    [(n*9)+m];
75                                           }
76                                  };
77
78                                  SudokuGUI.getAnswer =
                                        solutionArray2D;
79                                  SudokuGUI.getAnswerWithGrid =
                                        solutionWithGrid;
```

```
                                        break;
                                }

                        } while(true);
        }

}
```

The Field class:

```
import java.io.*;
import java.util.*;

/**
 * Abstract Data Type for Sudoku playing field
 */
public class Field {

  public static final int SIZE = 9;

  public int model[][] ;

  public Field() {
    // make new array of size SIZExSIZE
    this.model = new int[SIZE][SIZE];
    // initialize with empty cells
    init(SIZE-1, SIZE-1);
  }

  private void init(int i, int j) {
    if (i < 0) {
       // all rows done!
    } else if (j < 0) {
      // this row done - go to next!
      init(i-1, SIZE-1);
    } else {
      this.clear(i,j);
      init(i, j-1);
    }
  }

  public void fromFile(String fileName) {
    try {
      Scanner sc = new Scanner(new File(fileName));
      fromScanner(sc, 0, 0);
    } catch (FileNotFoundException e) {
      System.out.println("Invalid path for .txt file");
    }
  }

  private void fromScanner(Scanner sc, int i, int j) {
    if (i >= SIZE) {
      // all rows done!
    } else if (j >= SIZE) {
```

```
45      // this row done - go to next!
46      fromScanner(sc, i+1, 0);
47    } else {
48      try {
49        int val = Integer.parseInt(sc.next());
50        this.model[i][j] = val;
51      } catch (NumberFormatException e) {
52        // skip this cell
53      }
54      fromScanner(sc, i, j+1);
55    }
56  }
57
58  public String toString() {
59    StringBuffer res = new StringBuffer();
60    for (int i = 0; i < SIZE; i++) {
61      if (i % 3 == 0) {
62        res.append("+-------+-------+-------+\n");
63      }
64      for (int j = 0; j < SIZE; j++) {
65        if (j % 3 == 0) {
66          res.append("| ");
67        }
68        int val = this.model[i][j];
69        res.append(val > 0 ? val+" " : " ");
70      }
71      res.append("|\n");
72    }
73    res.append("+-------+-------+-------+");
74    return res.toString();
75  }
76
77  /** returns false if the value val cannot be placed at
78   *  row i and column j. returns true and sets the cell
79   *  to val otherwise.
80   */
81  public boolean tryValue(int val, int i, int j) {
82    if (!checkRow(val, i)) {
83      return false;
84    }
85    if (!checkCol(val, j)) {
86      return false;
87    }
88    if (!checkBox(val, i, j)) {
89      return false;
90    }
91    this.model[i][j] = val;
92    return true;
93  }
94
95  /** checks if the cell at row i and column j is empty,
96   *  i.e., whether it contains 0
97   */
98  public boolean isEmpty(int i, int j) {
```

```
99
100         for(int n = 0; n < SIZE; n++) {
101                 for (int m = 0; m < SIZE; m++) {
102                         if(model[i][j] == 0) {
103                                 return true;
104                         }
105                 }
106         }
107       return false;
108   }
109
110   /** sets the cell at row i and column j to be empty, i.e.,
111    *  to be 0
112    */
113   public void clear(int i, int j) {
114
115                       model[i][j] = 0;
116   }
117
118   /** checks if val is an acceptable value for the row i */
119   private boolean checkRow(int val, int i) {
120
121         for(int n = 0; n < SIZE; n++) {
122                 if(model[i][n] == val) {
123                         return false;
124                 }
125         }
126       return true;
127   }
128
129   /** checks if val is an acceptable value for the column j */
130   private boolean checkCol(int val, int j) {
131
132         for(int n = 0; n < SIZE; n++) {
133                if(model[n][j] == val) {
134                       return false;
135                }
136         }
137       return true;
138   }
139
140   /** checks if val is an acceptable value for the box around
141    *  the cell at row i and column j
142    */
143   private boolean checkBox(int val, int i, int j) {
144
145         int iLow, iTop, jLow, jTop;
146
147               if((i < 3) && (i >= 0)) {
148                       iLow = 0;
149                       iTop = 3;
150               }
151                       else if((i < 6) && (i >= 3)) {
152                               iLow = 3;
```

```
153                             iTop = 6;
154                     }
155                             else {
156                                     iLow = 6;
157                                     iTop = 9;
158                             }
159
160             if((j < 3) && (j >= 0)) {
161                     jLow = 0;
162                     jTop = 3;
163             }
164                     else if((j < 6) && (j >= 3)) {
165                             jLow = 3;
166                             jTop = 6;
167                     }
168                             else {
169                                     jLow = 6;
170                                     jTop = 9;
171                             }
172
173             for(int n = iLow; n < iTop; n++) {
174                     for (int m = jLow; m < jTop; m++) {
175                             if(model[n][m] == val) {
176                                     return false;
177                             }
178                     }
179             }
180         return true;
181   }
182 }
```

The SolvedException class:

```
1  @SuppressWarnings("serial")
2  public class SolvedException extends Exception {
3
4  }
```

The SudokuGUI class:

```
1  import java.awt.Dimension;
2  import java.awt.FlowLayout;
3  import java.awt.event.ActionEvent;
4  import java.awt.event.ActionListener;
5  import java.io.File;
6  import java.io.FileNotFoundException;
7  import javax.swing.JButton;
8  import javax.swing.JFrame;
9  import javax.swing.JLabel;
10 import javax.swing.JPanel;
11 import javax.swing.JScrollPane;
12 import javax.swing.JTable;
13
14 @SuppressWarnings("serial")
```

```
15 public class SudokuGUI extends Sudoku {
16
17         public static String[][] getAnswer;
18         static Object[][] solutionData = new Object [9][9];
19         public static String getAnswerWithGrid;
20
21         public static void Frame(String[] emptyCol, Object[][] loadData,
               Object[][] solutionData, File path) {
22
23                 // frame and panel
24                 JFrame frame = new JFrame("Sudoku Solver");
25                 JPanel panel = new JPanel();
26                 panel.setLayout(new FlowLayout(FlowLayout.CENTER,3,3));
27
28                 // frame settings
29                 frame.setSize(270, 450);
30                 frame.setResizable(false);
31                 frame.add(panel);
32                 frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
33
34                 // buttons
35                 JButton solve = new JButton("    Start       ");
36                 JButton save = new JButton(" Save file   ");
37                 JButton load = new JButton(" Load file   ");
38
39                 // labels
40                 JLabel unsolved = new JLabel("     Unsolved        ");
41                 JLabel solution = new JLabel("      Solution        ");
42
43                 // add load button to panel, and add actionlistener
44                 panel.add(load);
45                 load.addActionListener(new ActionListener() {
46
47                         @SuppressWarnings("static-access")
48                         @Override
49                         public void actionPerformed(ActionEvent e) {
50
51                                 FileHandler load = new FileHandler();
52                                 try {
53                                         load.openFile(null);
54                                         frame.dispose();
55                                 } catch (FileNotFoundException e1) {
56                                         e1.printStackTrace();
57                                 }
58                         }
59                 });
60
61                 // if nothing in first cell of solution-array (then JTable
                       is empty), set save-button enable to false, else true
                      .
62                 if (solutionData[0][0] == null) {
63
64                         panel.add(save);
65                         save.setEnabled(false);
```

```
66              }
67              else {
68                      panel.add(save);
69                      save.addActionListener(new ActionListener() {
70
71                              @Override
72                              public void actionPerformed(ActionEvent e) {
73
74                                      FileHandler save = new FileHandler();
75                                      save.saveFile(getAnswerWithGrid);
76                              }
77                      });
78                      save.setEnabled(true);
79              }
80
81              // add other components
82              panel.add(unsolved);
83              paintInputTable(loadData, emptyCol, panel);
84
85              panel.add(solution);
86              paintOutputTable(solutionData, emptyCol, panel);
87
88              // if nothing in first cell of loadedData-array (then
                    JTable is empty), set solve-button enable to false,
                    else
89              // set solve-button to true, call solver from Field, and
                    set save-button to true.
90              if (loadData[0][0] == null || (loadData[0][0] != null &&
                    solutionData[0][0] != null)) {
91
92                      panel.add(solve);
93                      solve.setEnabled(false);
94              }
95
96              else {
97                      panel.add(solve);
98                      solve.setEnabled(true);
99                      solve.addActionListener(new ActionListener() {
100
101                             @Override
102                             public void actionPerformed(ActionEvent e2)
                                    {
103
104                                     solver(path, loadData);
105                                     frame.dispose();
106                             }
107                     });
108             }
109             frame.setVisible(true);
110     }
111
112     public static void main(String[] args) {
113
114             String[] emptyCol = {"","","","","","","","","",};
```

```
115                 Object[][] loadData = new Object [9][9];
116                 File path = null;
117
118                 // call frame-method with emptyColums to jtable, empty 2D-
                        string to loadData,
119                 // empty 2D-string to solutionsData, and empty path.
120                 Frame(emptyCol, loadData, solutionData, path);
121         }
122
123         public static void paintInputTable(Object[][] loadData, String[]
                emptyCol, JPanel panel) {
124
125                 JTable input = new JTable(loadData, emptyCol);
126                         input.setTableHeader(null);
127                         input.setPreferredScrollableViewportSize(new
                                Dimension(150, 144));
128                         input.setFillsViewportHeight(true);
129
130                 JScrollPane jps = new JScrollPane(input);
131                 panel.add(jps);
132         }
133
134         public static void paintOutputTable(Object[][] solutionData,
                String[] emptyCol, JPanel panel) {
135
136                 JTable output = new JTable(solutionData, emptyCol);
137                         output.setTableHeader(null);
138                         output.setPreferredScrollableViewportSize(new
                                Dimension(150, 144));
139                         output.setFillsViewportHeight(true);
140
141                 JScrollPane jps_2 = new JScrollPane(output);
142                 panel.add(jps_2);
143         }
144
145         public static void solver(File result, Object[][] loadData) {
146
147                         String path = result.toString();
148                         Field field = new Field();
149
150                             field.fromFile(path);
151                             try {
152                               solve(field, 0, 0);
153
154                             String[] emptyCol = {"","","","","","","","","",
                                    ,};
155                             Frame(emptyCol, loadData, getAnswer, null);
156
157                             } catch (SolvedException e) {}
158         }
159 }
```

The FileHandler class:

```
1  import java.awt.FlowLayout;
2  import java.io.BufferedWriter;
3  import java.io.File;
4  import java.io.FileNotFoundException;
5  import java.io.FileWriter;
6  import java.io.IOException;
7  import java.util.Scanner;
8  import javax.swing.JFileChooser;
9  import javax.swing.JPanel;
10 import javax.swing.filechooser.FileNameExtensionFilter;
11
12 public class FileHandler {
13
14        static Object[][] solutionData = new Object [9][9];
15
16        public static void openFile(String title) throws
              FileNotFoundException {
17
18               JPanel panel = new JPanel();
19               panel.setLayout(new FlowLayout(FlowLayout.CENTER,3,3));
20
21               File filePath = null;
22               JFileChooser chooser = new JFileChooser(new File("."));
23
24                   if (title != null)
25                       chooser.setDialogTitle(title);
26                       int retVal = chooser.showOpenDialog(null);
27
28                       if(retVal == JFileChooser.APPROVE_OPTION) {
29                              filePath = chooser.getSelectedFile();
30
31                              Scanner input = new Scanner(filePath);
32                              Object[][] loadData = new Object [9][9];
33
34                                     while(input.hasNext()) {
35
36                                     for (int i = 0; i < 9; i++) {
37                                            for (int j = 0; j < 9; j++) {
38
39                                             loadData[i][j] = input.next
                                                   ();
40                                                  }
41                                                  System.out.println
                                                       ();
42                                                 }
43                                            String[] emptyCol = {"","","""
                                                 ,"","","","","","",};
44                                     SudokuGUI.Frame(emptyCol, loadData,
                                           solutionData, filePath);
45                              }
46                       input.close();
47                              }
48        }
49
```

```
50          public void saveFile(String getAnswerWithGrid) {
51
52          JFileChooser fc = new JFileChooser();
53          FileNameExtensionFilter filter = new FileNameExtensionFilter("
                Text Files", "txt");
54          fc.setFileFilter(filter);
55          int rval = fc.showSaveDialog(fc);
56
57                  if (rval == JFileChooser.APPROVE_OPTION) {
58                          File file = fc.getSelectedFile();
59
60                          try {
61
62
63                                  BufferedWriter out = new BufferedWriter(new
                                        FileWriter(file + ".txt"));
64                                  out.write(getAnswerWithGrid);
65                                  out.flush();
66                                  out.close();
67                          } catch(IOException e) {
68                                  // error
69                            }
70                  }
71          }
72  }
```