Introduction to Programming

Name: Danny Rene Jensen
D1
Supervisor: Jan Baumbach
Teaching assistant: Jakob Rodenberg
DM536

12. januar 2015

Indhold

1 Specification

1.1 Terminal run

This program should be able to solve a sudoku, when the program is run through the terminal, with a another file, in which the unsolved sudoku is.

1.2 GUI

This program should be a GUI program with a sudoku board and some buttons. The first button should be able to load an unsolved sudoku onto the board. The next button should solve the sudoku and fill in the blanks on the board. The last button should be able to save the sudoku to a file which the user specifies.

2 Design

2.1 Field

First of, the Field.java should be coded. This was just taking every single peace and try with a model, that already was specified. First of, it needs to check if the square, it is currently looking at, is empty ie if there is a number other then 0. After that, to get it to check the row was easy, it just needs to run through all i and compare it to the value the program is trying to place, if it is there then it returns false. For it to check the collumn it is almost the same code, just with j instead of i. The hardest part in this was actually to get the math right for the checkBox code. It needs to check the small 3x3 box, in which it is trying to put a number in.

```
1  private boolean checkBox(int val, int i, int j) {
2         for (int n=(i/3*3); n < (i/3*3)+3; n++){
3                 for (int k=(j/3*3); k k< (j/3*3)+3; k++){
4                         if (this.model[n][k] == val){
5                                 return false;
6                         }
7                 }
8         }
9         return true;
10 }
```

It is using whole number division on i and j to find out in which box of 3x3 it is. After that it multiplies that by 3 to go to the first cell in that 3x3 box, then it just adds 3 to go through the box. The last thing was clear. The solver needs to be able to clear the cell, if it made a mistake. So it just needs to set the value at the cell to 0.

2.2 Terminal run

```
public static void solve(Field f, int i, int j) throws SolvedException {
       if (i >= Field.SIZE) {
              throw new SolvedException();
       }
       else{
              if(j >= Field.SIZE) {
                     solve(f,i+1,0);
              }
              else{
                     if(f.isEmpty(i,j) == true ) {
                            for (int val=1; val<10; val++){
                                   if(f.tryValue(val,i,j) == true){
                                   solve(f,i,j+1);
                                   f.clear(i,j);
                                   }
                            }

                     }
                     else{
                            solve(f,i,j+1);
                     }
              }
       }
 }
```

The Sudoku.java is the solver for both this program and the GUI.
The solve function is made recursivly. First of, it checks that if i is larger then the board the program is done. If i is smaller it still needs to solve some cells, or at least check them. So then, it checks if j is larger, if it is, it takes a new row, by setting i+1 and j to 0. When both i and j are smaller then the board size, it then checks the cell at i, j by using the Field. If it gets a true from the tryValue, it then goes on to the next j and uses the solve function on that. Now, if it makes a mistake and it needs to go back a few steps, that is why, right under the recursive call the clear function from Field is. This clears the number, so the isEmpty won't tell the program, that it can not write here.
After the solve is done, it throws a SolvedException for the main to see and then all the main has to do is print out the board from Field.

2.3 GUI

The window should contain 4 buttons, a load button, a solve button, a save button and a clear button. The clear button is added for the sole reason to not load multiple files at a time and get an error.
The window should also hold the sudoku board, which in my case is build of 9 panels with each 9 textfilds inside, all of this is inside a pane.

```
for(int x=0; x<=2; x++){
       for(int z=0; z<=2; z++){
              for(int i=0; i<=2; i++ ){
                     for(int j=0; j<=2; j++){
                            p[x][z].add(tf[i+x*3][j+z*3]);
                     }
              }
              gridPane.add(p[x][z]);
       }
}
```

The gridPane which holds my panels, has a grid layout with a spacing of 5 pixels:

```
gridPane.setLayout(new GridLayout(3,3,5,5));
```

All the buttons is inside a panel, which also has a gridlayout.
Then the btnPanel and gridPanel goes inside a container pane. This is done to easier set the layout that I wanted. The containerPanel is then added to the frame, along with a textfield that tells you to clear the board, before you try to load another file.
The frame has a GridBagContainer layout on, so it is able to set the controlPanel over the textfild, and still have space around them.

```
GridBagLayout layout = new GridBagLayout();
Frame.setLayout(layout);
GridBagConstraints gbc = new GridBagConstraints();

gbc.fill =GridBagConstraints.HORIZONTAL;
gbc.gridx = 0;
gbc.gridy = 0;
Frame.add(controlPanel,gbc);

gbc.gridx = 0;
gbc.gridy = 1;
Frame.add(notCleared,gbc);
```

3 Implementation

.

3.1 Terminal run

This program needs to be able to solve and print out a sudoku from a txt file. The program should do so by using the Field class to check if a number can be placed at that square. If it can, then it needs to try the next place. This is done recursive inside the solve code inside Sudoku.java. After it is done solving the sudoku, the solve code is throwing a solvedExeption and then it prints out the board using the toString code in Field, to create spacing.

3.2 GUI

The program makes the window of a specific size, because I don't want the window to be resizable:

```
1 Frame = new JFrame("SudokuGUI");
2 Frame.setSize(500,500);
3 Frame.setResizable(false);
```

The buttons are just normal JButtons with standard length and height.
The textfields also have no specific size on them, they just fill out the panel they are put on, which if the gridPanel gets resized, they will also get another size.
When all the things are created in the window, when the user clicks on the load button, a load screen needs to appear, this happens in line 5 in the code below. When the user has found the file to be loaded, the button only needs to get the whole path and put that into the Field.fromfile. This will change the model[][][] in Field to be a board of the file. This model[][][] is then copied over to the SudokuGUI into a board[][][], This board is then copied over to all my textfields, so it can be shown on screen.
The load button also has another purpose. If someone tries to load a file without clearing the board, it won't load, but instead, tell the user to clear the board first. This is to prevent multible file to be loaded at the same time and then creating an error.

```
1  openBtn.addActionListener(new ActionListener() {
2         public void actionPerformed(ActionEvent arg0) {
3                 JFileChooser openFile = new JFileChooser();
4                 if(x == 0){
5                         openFile.showOpenDialog(null);
6                         field.fromFile(openFile.getSelectedFile().
                              getAbsolutePath());
7                         x = 1;
8                         for (int j=0; j<9; j++){
9                            for(int i=0; i<9; i++){
10                                  if (board[i][j]==0){
11                                        tf[i][j].setBackground(Color.
                                              YELLOW);
12                                  }else{
13                                        tf[i][j].setText("" + board[i
                                              ][j]);
14                                  }
15                           }
16                        }
17                }else{
18                        notCleared.setText("Please clear field before
                              loading a new file.");
19                }
20        }
21 });
```

The clear button just empties all the textfields and of course the board[][][].
The solve button is constructed almost like the termial run programs main, with only minor changes to it. Instead of making solve all over again, it is using the

solve from the termail program. And instad of printing the finished board in the terminal, it is copied into board[][] and then onto all the textfields.

```
solveBtn.addActionListener(new ActionListener() {
	public void actionPerformed(ActionEvent arg0) {
		try {
			sudoku.solve(field, 0, 0);
		} catch (SolvedException e) {}
		for (int i=0; i<9; i++){
			for(int j=0; j<9; j++){
				tf[i][j].setText("" + board[i][j]);
			}
		}
	}
});
```

The save button was probably the hardest to make. This button brings up the save window, when the user enters the path and the file name then it creates the file. After it is done it writes each textfield to it in the for loops.

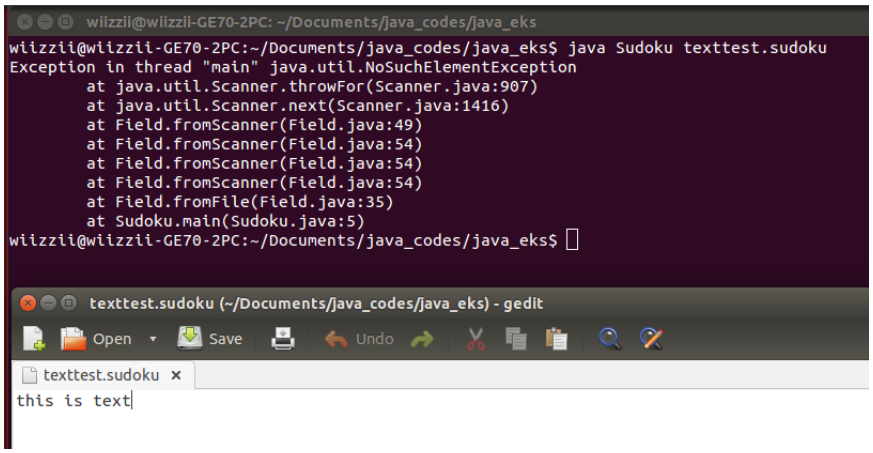
4 Testing

4.1 Terminal run

Let's test the Sudoku.java program with test1.sudoku(test1.sudoku can be found in apendix section 6.3.1):

This worked perfectly. It solved the sudoku and printed it out without any problems.

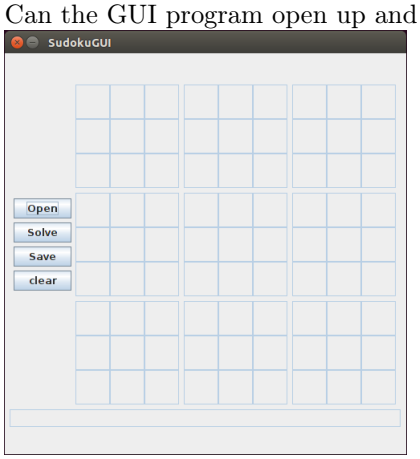
But what happens, if it gets a text file(testtext.sudoku can be seen in the picture at the bottom):



It certainly did not like that, this is because fromScanner in Field.java is trying to insert it into an 2D Array and that only takes intergers.

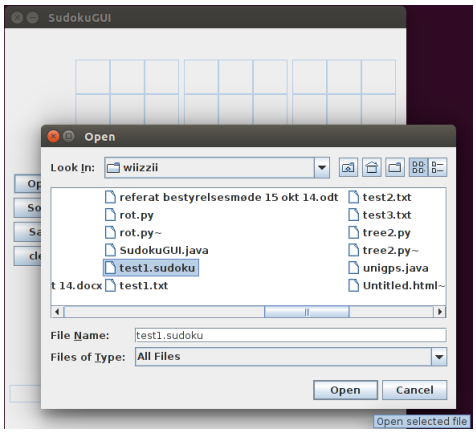
4.2 GUI

Can the GUI program open up and show the window:



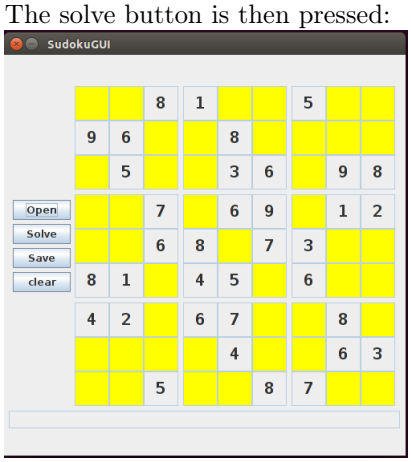
Without any problems, it opened up and showed all the components it is supposed to.

Now, let's try to load a test file(test1.sudoku 6.3.1):



The load screen showed up perfectly and it can be navigated through the computers folders.

The loaded files(test1.sudoku 6.3.1) numbers shows up on the board, and the yellow places is where the numbers are missing.
The solve button is then pressed:

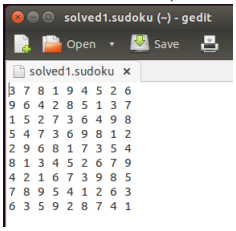


The GUI solves the sudoku without any problems and the user can still see the unsolved sudoku by looking at the squares, that are not yelllow.

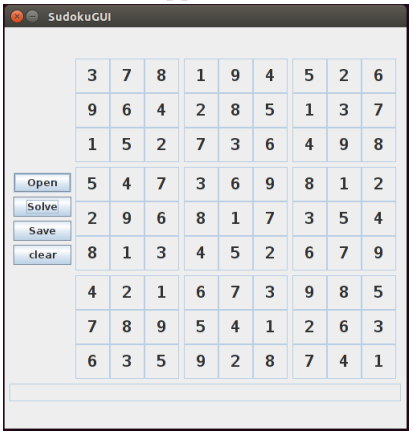
The save button:

The save screen also pops up and the user can also navigate through the pc folders.

The saved file(solved1.sudoku can also be found in appendix section 6.3.3):



Now, what happens if a solved sudoku(solvedtest1) is loaded:

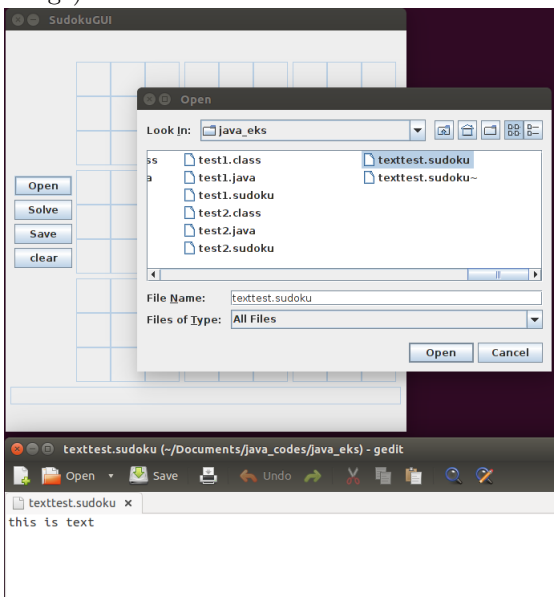


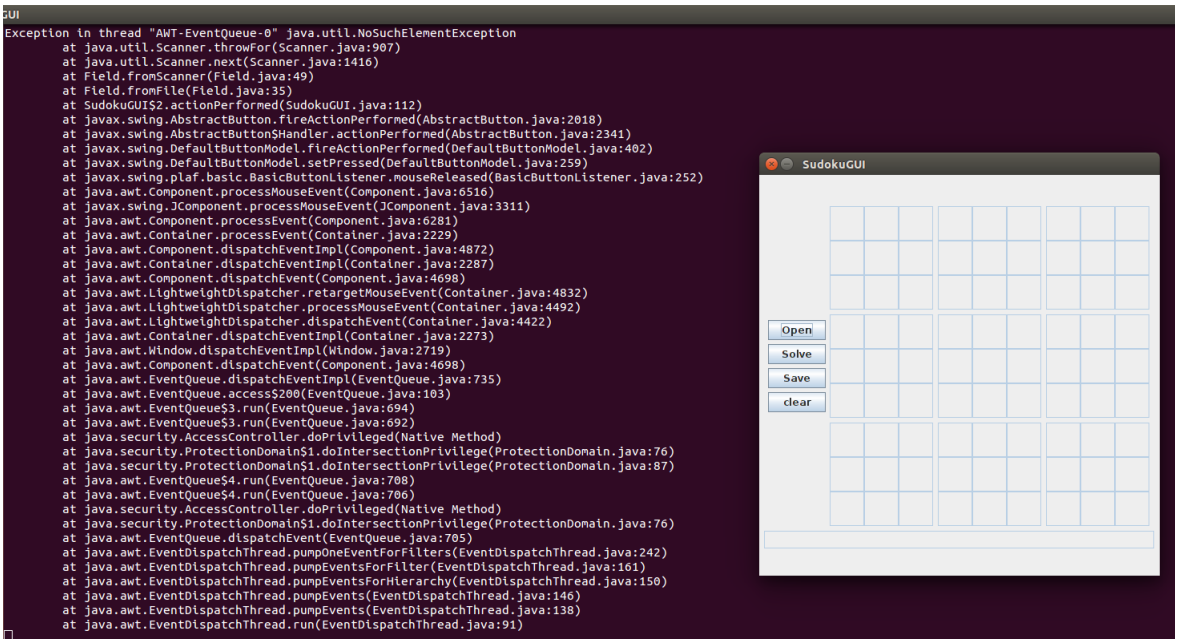
The solved (solved1.sudoku) is loaded in just fine, it didn't show any yellow squares, because there is nothing to solve. The solve button does nothing either.

What happens if the user tries to load a file, when he already has a file loa-

ded:
The build-in safty mekanism kicks in and tells the user to clear the board, before trying to load a new one.

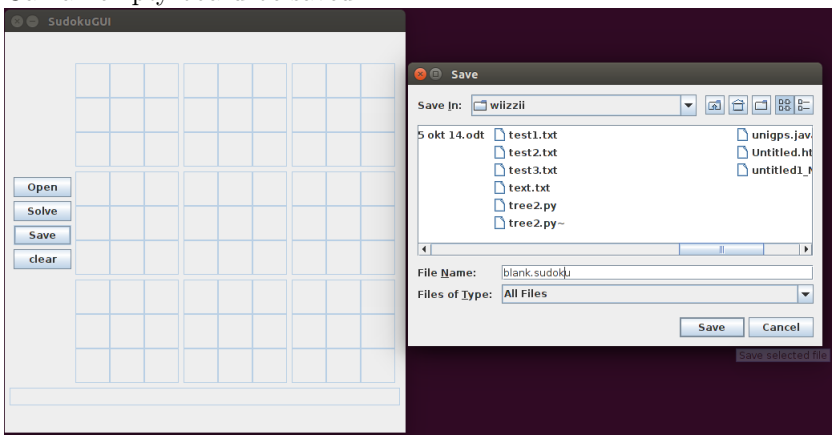
Now, let's try to load a text(testtex.sudoku can be seen at the buttom of the image):



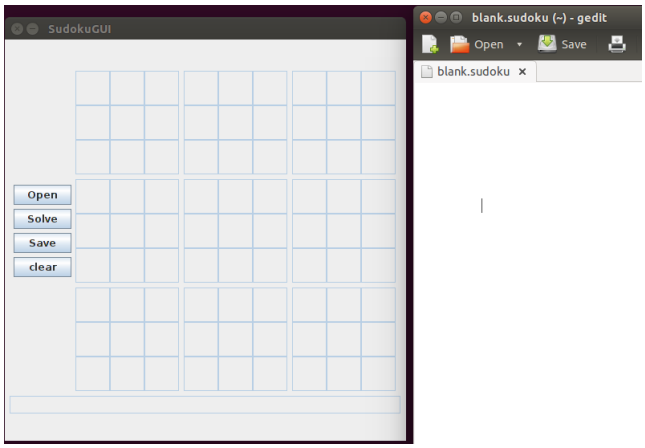


A lot of errors pops up in the terminal, but nothing changes in the GUI window, and a real sudoku file can be loaded and solved without any problems.

Can an empty board be saved:

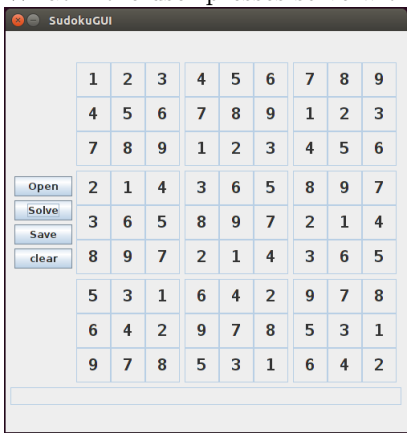


Apparently it goes smoothly, but what is inside the saved file:



The saved file is just some empty spaces on 9 rows seen to the right in the image above.

What if the user presses solve without loading any file:



The GUI actually makes a sudoku by itself! After posting from 1 to 9 in the first row.

5 Conclusion

5.1 Terminal run

All in all the program is working as it should. Maybe it could use something that finds out if it is a .txt file, .png file or a .sudoku file.

5.2 GUI

The GUI layout is just like I wanted it to be, but maybe instead of a clear button, the load button could just clear it, when pressed. And this too could use a wrong file type identifier. The save button could also be coded, so I couldn't save an empty, or unsolved sudoku.

6 Appendix (source code)

6.1 Field

```
1  import java.io.*;
2  import java.util.*;
3
4  /**
5   * Abstract Data Type for Sudoku playing field
6   */
7  public class Field {
8
9    public static final int SIZE = 9;
10
11   private int model[][] ;
12
13   public Field() {
14     // make new array of size SIZExSIZE
15     this.model = new int[SIZE][SIZE];
16     // initialize with empty cells
17     init(SIZE-1, SIZE-1);
18   }
19
20   private void init(int i, int j) {
21     if (i < 0) {
22        // all rows done!
23     } else if (j < 0) {
24       // this row done - go to next!
25       init(i-1, SIZE-1);
26     } else {
27       this.clear(i,j);
28       init(i, j-1);
29     }
30   }
31
32   public void fromFile(String fileName) {
33     try {
34       Scanner sc = new Scanner(new File(fileName));
35       fromScanner(sc, 0, 0);
36     } catch (FileNotFoundException e) {
37       // :-(
38     }
39   }
40
41   private void fromScanner(Scanner sc, int i, int j) {
42     if (i >= SIZE) {
43       // all rows done!
44     } 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  public int[][] getBoard() {
96       return this.model;
97
98  }
99
100 /** checks if the cell at row i and column j is empty,
101  *  i.e., whether it contains 0
102  */
103 public boolean isEmpty(int i, int j) {
```

```
104     // TODO
105                         if (this.model[i][j] == 0){
106                                 return true;
107                         }
108                         else {
109                                 return false;
110                         }
111     }
112
113     /** sets the cell at row i and column j to be empty, i.e.,
114      *  to be 0
115      */
116     public void clear(int i, int j) {
117       // TODO
118                   this.model[i][j]=0;
119     }
120
121     /** checks if val is an acceptable value for the row i */
122     private boolean checkRow(int val, int i) {
123       // TODO
124                         for (int j = 0; j < SIZE; j++){
125                                 if (this.model[i][j] == val){
126                                         return false;
127                                 }
128                         }
129                         return true;
130           }
131
132     /** checks if val is an acceptable value for the column j */
133     private boolean checkCol(int val, int j) {
134       // TODO
135                         for (int i = 0; i < SIZE; i++){
136                                 if (this.model[i][j] == val){
137                                         return false;
138                                 }
139                         }
140                         return true;
141     }
142
143     /** checks if val is an acceptable value for the box around
144      *  the cell at row i and column j
145      */
146     private boolean checkBox(int val, int i, int j) {
147       // TODO
148                         for (int n=(i/3*3); n < (i/3*3)+3; n++){
149                                 for (int k=(j/3*3); k < (j/3*3)+3; k++){
150                                         if (this.model[n][k] == val){
151                                                 return false;
152                                         }
153                                 }
154                         }
155                         return true;
156     }
157
```

```
158 }
```

6.2 Terminal run

```
1  public class Sudoku {
2
3    public static void main(String[] args) {
4      Field field = new Field();
5      field.fromFile(args[0]);
6      try {
7        solve(field, 0, 0);
8      } catch (SolvedException e) {}
9      System.out.println(field);
10   }
11
12   public static void solve(Field f, int i, int j) throws SolvedException
         {
13     if (i >= Field.SIZE) {
14       // we are done!
15                 throw new SolvedException();
16     }
17     else{
18                 if(j >= Field.SIZE) {
19                        solve(f,i+1,0);
20                 }
21                 else{
22                        if(f.isEmpty(i,j) == true ) {
23                               for (int val=1; val<10; val++){
24                                      if(f.tryValue(val,i,j) == true){
25                                      solve(f,i,j+1);
26                                      f.clear(i,j);
27                                      }
28                               }
29
30                        }
31                        else{
32                               solve(f,i,j+1);
33                        }
34                 }
35
36         }
37   }
38
39 }
```

6.3 GUI

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4  import java.io.*;
5
6  public class SudokuGUI extends JFrame{
7         Field field = new Field();
8         Sudoku sudoku = new Sudoku();
9         int[][] board = field.getBoard();
10        int x = 0;
11        private JFrame Frame;
12        private JPanel controlPanel;
13        private JTextField notCleared = new JTextField();
14     private JTextField tf[][]= new JTextField[9][9];
15     private JPanel panel[][]= new JPanel [3][3];
16
17     public SudokuGUI(){
18         for(int i=0; i<9; i++){
19             for(int j=0; j<9; j++){
20                 tf[i][j]=new JTextField(1);
21             }
22         }
23
24         for(int x=0; x<3; x++){
25             for(int z=0; z<3; z++){
26                 panel[x][z]=new JPanel(new GridLayout(3,3));
27             }
28         }
29         PrepareGUI();
30         BtnDisplay();
31     }
32
33     public static void main(String[] args){
34
35                 SudokuGUI win = new SudokuGUI();
36                 win.ShowGroupLayout();
37         }
38
39         private void PrepareGUI(){
40                 Frame = new JFrame("SudokuGUI");
41                 Frame.setSize(500,500);
42                 Frame.setResizable(false);
43
44                 notCleared.setHorizontalAlignment(JTextField.CENTER);
45                 notCleared.setFont(new Font("Monospaced", Font.BOLD, 15));
46                 notCleared.setEditable(false);
47
48                 controlPanel = new JPanel();
49                 controlPanel.setLayout(new FlowLayout());
50
51                 GridBagLayout layout = new GridBagLayout();
```

```
52                Frame.setLayout(layout);
53                GridBagConstraints gbc = new GridBagConstraints();
54
55                gbc.fill = GridBagConstraints.HORIZONTAL;
56                gbc.gridx = 0;
57                gbc.gridy = 0;
58                Frame.add(controlPanel,gbc);
59
60                gbc.gridx = 0;
61                gbc.gridy = 1;
62                Frame.add(notCleared,gbc);
63
64                Frame.addWindowListener(new WindowAdapter() {
65                        public void windowClosing(WindowEvent windowEvent){
66                                System.exit(0);
67                        }
68                });
69
70                Frame.setVisible(true);
71        }
72
73        private void ShowGroupLayout(){
74                Container gridPane = getContentPane();
75                gridPane.setPreferredSize(new Dimension(400, 400));
76                gridPane.setLayout(new GridLayout(3,3,5,5));
77
78                for (int i=0; i<9; i++){
79                        for(int j=0; j<9; j++){
80                                tf[i][j].setHorizontalAlignment(JTextField.
                                    CENTER);
81                                tf[i][j].setFont(new Font("Monospaced", Font
                                    .BOLD, 20));
82                                tf[i][j].setEditable(false);
83                        }
84                }
85
86        for(int x=0; x<=2; x++){
87            for(int z=0; z<=2; z++){
88                for(int i=0; i<=2; i++ ){
89                    for(int j=0; j<=2; j++){
90                        panel[x][z].add(tf[i+x*3][j+z*3]);
91                    }
92                }
93            gridPane.add(panel[x][z]);
94            }
95        }
96                controlPanel.add(gridPane);
97        }
98
99        private void BtnDisplay(){
100               JPanel btnPanel = new JPanel();
101               JButton openBtn = new JButton("Open");
102               JButton solveBtn = new JButton("Solve");
103               JButton saveBtn = new JButton("Save");
```

```
104                 JButton clearBtn = new JButton("clear");
105
106                 openBtn.addActionListener(new ActionListener() {
107                         public void actionPerformed(ActionEvent arg0) {
108                                 JFileChooser openFile = new JFileChooser();
109
110                                 if(x == 0){
111                                         openFile.showOpenDialog(null);
112                                         field.fromFile(openFile.
                                                getSelectedFile().
                                                getAbsolutePath());
113                                         x = 1;
114                                         for (int j=0; j<9; j++){
115                                                 for(int i=0; i<9; i++){
116                                                         if (board[i][j]==0){
117                                                                 tf[i][j].
                                                                        setBackground
                                                                        (Color.
                                                                        YELLOW);
118                                                         }else{
119                                                                 tf[i][j].
                                                                        setText(""
                                                                        + board[i][
                                                                        j]);
120                                                         }
121                                                 }
122                                         }
123                                 }else{
124                                         notCleared.setText("Please clear
                                                field before loading a new file.
                                                ");
125                                 }
126                         }
127                 });
128
129                 solveBtn.addActionListener(new ActionListener() {
130                         public void actionPerformed(ActionEvent arg0) {
131                                 try {
132                                         sudoku.solve(field, 0, 0);
133                                 } catch (SolvedException e) {}
134                                 for (int i=0; i<9; i++){
135                                         for(int j=0; j<9; j++){
136                                                 tf[i][j].setText("" + board[i
                                                        ][j]);
137                                         }
138                                 }
139                         }
140                 });
141
142                 saveBtn.addActionListener(new ActionListener() {
143             public void actionPerformed(ActionEvent arg0) {
144                 JFileChooser saveFile = new JFileChooser();
145                 saveFile.showSaveDialog(null);
146                 try{
```

```
147                                     File file = new File(saveFile.
                                            getSelectedFile().
                                            getAbsolutePath());
148                                     if (!file.exists()){
149                                            file.createNewFile();
150                                     }
151                                     FileWriter fwrite = new FileWriter(
                                            file.getAbsoluteFile());
152                                     BufferedWriter bwrite = new
                                            BufferedWriter(fwrite);
153                                     for (int i=0; i<9; i++){
154                                            for(int j=0; j<9; j++){
155                                                   bwrite.write(tf[i][j].
                                                          getText() + " ");
156                                            }
157                                            bwrite.newLine();
158                                     }
159                                     bwrite.close();
160
161                              } catch (IOException e){
162                                     e.printStackTrace();
163                              }
164                     }
165              });
166
167                     clearBtn.addActionListener(new ActionListener() {
168                            public void actionPerformed(ActionEvent arg0) {
169                                   x = 0;
170                                   notCleared.setText(null);
171                            for (int j=0; j<9; j++){
172                                          for(int i=0; i<9; i++){
173                                                 board[i][j] = 0;
174                                                 tf[i][j].setText(null);
175                                                 tf[i][j].setBackground(null);
176                                          }
177                                   }
178                            }
179                     });
180
181                     btnPanel.setLayout(new GridLayout(4,1,5,5));
182                     controlPanel.add(btnPanel);
183                     btnPanel.add(openBtn);
184                     btnPanel.add(solveBtn);
185                     btnPanel.add(saveBtn);
186                     btnPanel.add(clearBtn);
187              }
188       }
```

6.4 Misc

6.4.1 test1.sudoku

```
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
```

6.4.2 test1.sudoku

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

6.4.3 solved1.sudoku

```
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
```