Introduction to Programming - Project Part 1

Supervisor: Peter Schneider-Kamp

Instructor: Martin stergaard Villumsen

DM 536

D1

Amar Djebbara
Birthday:
26-07-1991
January 12, 2016

Contents

1 Specification **2**

2 Design **2**

3 Implementation **2**

4 Testing **6**

5 Conclusion **7**

A SudokuGUI.java **8**

B Sudoku.java **12**

C Field.java **14**

1 Specification

Sudoku is a number puzzle game, witch can be a nuisance when they swallow your time to solve. To be able to solve such troublesome problem, a possible solution would be a brute force algorithm that could try all possible integers in every cell. To do that we have to use a mathematical method called recursion. This is a very slow method which uses a set of rules to find the numbers while backtracking every cell.

The second task is to build a GUI using implementations from the JAVA library, by building the GUI class for the Sudoku solver called "SudokuGUI". In this class the JFileChooser is implemented to be able to get the file, which contains the Sudoku cells we have to solve. By adding 3 buttons, gives the user ability to pick a file, solving the Sudoku and outputting the result field.

2 Design

The main plan was to take every class, step by step. This will give a clear idea of what the program is suppose to do. After understanding what was asked , it was relatively easy to plan the program structure, at least that was the idea.

3 Implementation

The plan was to follow the tasks in the project description. But after a careful reading and a few conversation with the instructor, the code was ready to be planned. The first tasks was to create a field class that could contain all the cells.
To do that we have to create a public class as such:

```
1  public class Field {
2
3    public static final int SIZE = 9;
4
5    public static int model[][];
6    public static int inputmodel[][];
7
8    public Field() {
9      // make new array of size SIZExSIZE
10     Field.model = new int[SIZE][SIZE];
11     Field.inputmodel = new int[SIZE][SIZE];
12     // initialize with empty cells
13     init(SIZE-1, SIZE-1);
14   }
```

As you can see, there is two field because later on in the code it will be a storing an Array witch is used in the SudokuGUI class.

The next important step in this build is that we need some instructions the program can follow so it could solve the sudoku by backtraicking every cell unto all cells are done. As the soduko rules states, we have to check every row, column and box of 3x3. Class had to be made as such:

```
1  public boolean checkRow(int val, int i) {
2        for( int j = 0; j < 9; j++ )
3               if( Field.model[i][j] == val  )
4                  return false ;
5
6            return true ;
7    }
8
9    /** checks if val is an acceptable value for the column j */
10   public boolean checkCol(int val, int j) {
11         for( int i = 0; i < 9; i++ )
12              if( Field.model[i][j] == val )
13                 return false ;
14
15         return  true;
16   }
17
18   /** checks if val is an acceptable value for the box around
19    *  the cell at row i and column j
20    */
21   public boolean checkBox(int val, int i, int j) {
22
23     i = (i / 3) * 3 ;
24     j = (j / 3) * 3 ;
25
26       for( int r = 0; r < 3; r++ )
27          for( int c = 0; c < 3; c++ )
28              if( Field.model[i+r][j+c] == val )
29                  return false ;
30
31       return true ;
32   }
```

This line of code is how the program opens the FDL files. It also sets a variable name called "s", which is used later in the program

Now that the class is working and reading the files. The next step is to make the rules, and apply them, so that the turtle can draw then in their respective depths. this is achieved as such:

```
def row(x):
    l = []
    for i in range(len(x)):
        if x[i] in s.rule.keys():
            l.extend(s.rule[x[i]])
        else:
            l.extend(x[i])
    return l

def frac(x, depth):
    if depth == 0:
        return x
    else:
        return row(frac(x, depth - 1))

print frac(s.start, s.depth)
```

For the GUI, a whole new java file is made, because of all the special tool of java libraries, witch gives us the ability to interact with the program and order it to do our hard work of backtracking every cell.
The first thing we had to implement is the layout of the field. This is achieved as such:

```
JPanel f1 = new JPanel();


        f1.setLayout( new GridLayout(9,9));

        add(f1, BorderLayout.CENTER);

        // buttons for the lulz :D

        area1 = new Button[9][9];
        // Create an empty view
        for( int i = 0; i < 9; i++ )
           for( int j = 0; j < 9; j++ )
           {
              area1[i][j]  = new Button() ;
              f1.add(area1[i][j]) ;

           }


        f1.setVisible(true);
```

The next step is to implement 3 buttons, "Choose file", "START" and "Save file". Using JfileChooser it was possible to pick a ".txt" file that contained the sudoku field.

```
1  JPanel f1 = new JPanel();
2
3
4          f1.setLayout( new GridLayout(9,9));
5
6          add(f1, BorderLayout.CENTER);
7
8          // buttons for the lulz :D
9
10         area1 = new Button[9][9];
11         // Create an empty view
12         for( int i = 0; i < 9; i++ )
13            for( int j = 0; j < 9; j++ )
14            {
15               area1[i][j]  = new Button() ;
16               f1.add(area1[i][j]) ;
17
18            }
19
20
21         f1.setVisible(true);
```

4 Testing

The big first big challenge, was to use the recursion to solve every cell, after some hard work and a lot of coffee it was possible to solve every sudoku i threw at it. this the finale solve class:

```
1  public static void solve(Field f, int i, int j) throws
       SolvedException, InterruptedException {
2
3
4      if (i >= Field.SIZE) {
5          solved2 = f.toString();
6          solvedf = Field.model;
7          System.out.println(f);
8          throw new SolvedException();
9          }
10      if(Field.model[i][j] != 0) {
11          nextCell(f, i, j);
12          SolvedFieldView();
13          }
14      else {
15         for( int val = 1; val < 10; val++) {
16            if(f.tryValue(val, i, j)) {
17             //System.out.println(f);
18             nextCell(f, i, j);
19             SolvedFieldView();
20
21             Thread.sleep( 50 ) ;
22             }
23         }
24
25         f.clear(i,j);
26         SolvedFieldView();
27      }
28   } // End of solve method
```

5 Conclusion

After several hours of coding, it finally came together. The program works with every single .txt file, respective to its content. And the GUI work's fine. There are room for MANY improvements witch surely will be taking care of, such as using BorderLayout to get the 3x3 boxes highlighted and so on. The most important thing is that it can solve a sudoku and display it.

Appendix

A SudokuGUI.java

```
1  import java.awt.*;
2  import java.awt.event.ActionEvent;
3  import java.awt.event.ActionListener;
4  import java.awt.geom.Area;
5  import java.io.File;
6  import java.io.FileWriter;
7  import java.io.IOException;
8  import java.io.PrintWriter;
9
10 import javax.swing.*;
11 import javax.swing.border.Border;
12 import javax.swing.border.LineBorder;
13
14
15
16 public class SudokuGUI extends JFrame implements ActionListener {
17     /**
18      *
19      */
20     private static final long serialVersionUID = 1L;
21     static JButton btn1 = new JButton("Choose File");
22     static JButton btn2 = new JButton("START");
23     JButton btn3 = new JButton("Save output");
24     JTextField filename = new JTextField();
25     static JTextField dir = new JTextField();
26     public static Button area1[][];
27
28
29
30     int returnval;
31
32     public SudokuGUI() {
33
34
35         super("Sudoku Solver v0.9");
36
37         JPanel f1 = new JPanel();
38
39
40         f1.setLayout( new GridLayout(9,9));
41
42         add(f1, BorderLayout.CENTER);
43
44         // buttons for the lulz :D
45
46         area1 = new Button[9][9];
47         // Create an empty view
48         for( int i = 0; i < 9; i++ )
49            for( int j = 0; j < 9; j++ )
50            {
51               area1[i][j]  = new Button() ;
52               f1.add(area1[i][j]) ;
```

```
53
54              }
55
56
57          f1.setVisible(true);
58
59          // buttons
60          final JPanel ButtonPanel = new JPanel(new FlowLayout(
                FlowLayout.CENTER));
61          ButtonPanel.setLayout(new BorderLayout());
62          ButtonPanel.add(btn1);
63          ButtonPanel.add(btn2);
64          ButtonPanel.add(btn3);
65
66
67          Border LineBorder  = new LineBorder(Color.lightGray);
68          ButtonPanel.setBorder(LineBorder);
69
70          BoxLayout horizontal = new BoxLayout(ButtonPanel, BoxLayout
                .X_AXIS);
71          ButtonPanel.setLayout(horizontal);
72
73          FlowLayout flow = new FlowLayout();
74          ButtonPanel.setLayout(flow);
75
76          add(ButtonPanel, BorderLayout.SOUTH);
77          setVisible(true);
78
79          btn1.addActionListener(new openfile());
80          btn2.addActionListener(new startSolve());
81          btn3.addActionListener(new saveFile());
82      }
83     public static class openfile implements ActionListener{
84
85      public static String path;
86      public static String infield = new String();
87
88      public void actionPerformed(ActionEvent e) {
89
90          if (e.getSource() == btn1) {
91               JFileChooser c = new JFileChooser();
92               int returnVal = c.showOpenDialog(null);
93               if(returnVal == JFileChooser.APPROVE_OPTION){
94               File file = c.getSelectedFile();
95               dir.setText(c.getSelectedFile().getAbsolutePath().
                     toString());
96               path = file.toString();
97               Field field = new Field();
98               String s = openfile.path;
99               field.fromFile(s);
100              System.out.println(field);
101              infield = field.toString();
102              NewFieldView();
103
104
105             }
106
```

```
107         }
108      }
109   }
110     public static class startSolve implements ActionListener{
111
112      public static String donefield = new String();
113
114      public void actionPerformed(ActionEvent e) {
115          if (e.getSource() == btn2) {
116              Field field = new Field();
117              field.fromFile(openfile.path);
118              try {
119                  Sudoku.solve(field, 0, 0);
120
121              } catch (SolvedException e1) {
122                  System.out.println("Done sol");
123                  SolvedFieldView();
124              } catch (InterruptedException e1) {
125                  // TODO Auto-generated catch block
126                  e1.printStackTrace();
127              }
128
129
130              }
131          }
132     }
133
134      public class saveFile implements ActionListener{
135        public void actionPerformed(ActionEvent e) {
136          if (e.getSource() == btn3) {
137              PrintWriter out = null;
138              try {
139                  out = new PrintWriter(new FileWriter("SolvedSudoku.
                        txt"));
140              } catch (IOException e1) {
141                  // TODO Auto-generated catch block
142                  e1.printStackTrace();
143              }
144              out.print(Sudoku.solved2);
145              out.close();
146              }
147          }
148      }
149
150      @Override
151      public void actionPerformed(ActionEvent arg0) {
152          // TODO Auto-generated method stub
153
154      }
155
156
157
158
159
160      public static void NewFieldView() {
161          for( int i = 0; i < 9; i++ )
162               for( int j = 0; j < 9; j++ )
```

```
163                 if( Field.inputmodel[i][j] != 0 )
164                    area1[i][j].setLabel( String.valueOf(Field.model
                           [i][j]) ) ;
165                 else
166                     area1[i][j].setLabel( "" ) ;
167         }
168     public static void SolvedFieldView() {
169         for( int i = 0; i < 9; i++){
170              for( int j = 0; j < 9; j++){
171                  if( Field.model[i][j] != 0 ){
172                    area1[i][j].setLabel( String.valueOf(Field.model
                           [i][j]) ) ;
173                  }
174                  else {
175                         area1[i][j].setLabel( "" ) ;
176                  }
177              }
178          }
179         }
180
181
182 }// end of SUdokuGUI
```

B Sudoku.java

```
1
2
3
4  public class Sudoku extends SudokuGUI{
5
6  /**
7       *
8       */
9    private static final long serialVersionUID = 1L;
10
11   public static String solved2;
12   public static int[][] solvedf;
13
14   public static void main(String[] args) throws SolvedException{
15     SudokuGUI sudokugui = new SudokuGUI();
16     sudokugui.setSize(550, 500);
17     sudokugui.setDefaultCloseOperation(EXIT_ON_CLOSE);
18     sudokugui.setLocationRelativeTo(null);
19
20     Field field = new Field();
21     field.fromFile(openfile.path);
22     //field.model[i][j] = field.inputmodel[i][j];
23     //NewFieldView();
24
25
26     /*Field field = new Field();
27     System.out.println();
28     //String s = openfile.path;
29     String s = "/home/byteme/Desktop/test1.txt";
30     field.fromFile(s);
31     System.out.println(field);
32     solve(field, 0, 0);
33     System.out.println(field);
34     System.out.println("       ##SOLVED##");*/
35
36     }
37
38   public static void solve(Field f, int i, int j) throws
         SolvedException, InterruptedException {
39
40
41     if (i >= Field.SIZE) {
42         solved2 = f.toString();
43         solvedf = Field.model;
44         System.out.println(f);
45         throw new SolvedException();
46         }
47      if(Field.model[i][j] != 0) {
48          nextCell(f, i, j);
49          SolvedFieldView();
50          }
51      else {
52         for( int val = 1; val < 10; val++) {
53            if(f.tryValue(val, i, j)) {
54             //System.out.println(f);
```

```
55              nextCell(f, i, j);
56              SolvedFieldView();
57
58              Thread.sleep( 50 ) ;
59              }
60          }
61
62          f.clear(i,j);
63          SolvedFieldView();
64       }
65    } // End of solve method
66
67    // Calls solve for the next cell
68    public static void nextCell(Field f, int i, int j ) throws
          SolvedException, InterruptedException {
69        if (j < 8)
70              solve(f, i, j+1);
71           else
72              solve(f, i+1, 0);
73    }
74
75
76
77
78
79
80
81
82  }//end of file
```

C Field.java

```
1
2  package Field;
3
4  import java.io.*;
5  import java.util.*;
6
7
8  /**
9   * Abstract Data Type for Sudoku playing field
10  */
11 public class Field {
12
13   public static final int SIZE = 9;
14
15   public int model[][] ;
16
17   public Field() {
18     // make new array of size SIZExSIZE
19     this.model = new int[SIZE][SIZE];
20     // initialize with empty cells
21     init(SIZE-1, SIZE-1);
22   }
23
24   private void init(int i, int j) {
25     if (i < 0) {
26        // all rows done!
27     } else if (j < 0) {
28       // this row done - go to next!
29       init(i-1, SIZE-1);
30     } else {
31       this.clear(i,j);
32       init(i, j-1);
33     }
34   }
35
36   public void fromFile(String fileName) {
37     try {
38       Scanner sc = new Scanner(new File(fileName));
39       fromScanner(sc, 0, 0);
40     } catch (FileNotFoundException e) {
41       // :-(
42     }
43   }
44
45   private void fromScanner(Scanner sc, int i, int j) {
46     if (i >= SIZE) {
47       // all rows done!
48     } else if (j >= SIZE) {
49       // this row done - go to next!
50       fromScanner(sc, i+1, 0);
51     } else {
52       try {
53         int val = Integer.parseInt(sc.next());
54         this.model[i][j] = val;
55       } catch (NumberFormatException e) {
```

```
56          // skip this cell
57        }
58        fromScanner(sc, i, j+1);
59      }
60    }
61
62    public String toString() {
63      StringBuffer res = new StringBuffer();
64      for (int i = 0; i < SIZE; i++) {
65        if (i % 3 == 0) {
66          res.append("+-------+-------+-------+\n");
67        }
68        for (int j = 0; j < SIZE; j++) {
69          if (j % 3 == 0) {
70            res.append("| ");
71          }
72          int val = this.model[i][j];
73          res.append(val > 0 ? val+" " : "  ");
74        }
75        res.append("|\n");
76      }
77      res.append("+-------+-------+-------+");
78      return res.toString();
79    }
80
81    /** returns false if the value val cannot be placed at
82     *  row i and column j. returns true and sets the cell
83     *  to val otherwise.
84     */
85    public boolean tryValue(int val, int i, int j) {
86      if (!checkRow(val, i)) {
87        return false;
88      }
89      if (!checkCol(val, j)) {
90        return false;
91      }
92      if (!checkBox(val, i, j)) {
93        return false;
94      }
95      this.model[i][j] = val;
96      return true;
97    }
98
99    /** checks if the cell at row i and column j is empty,
100    *  i.e., whether it contains 0
101    */
102   public boolean isEmpty(int i, int j) {
103    if (this.model[i][j] > 0)
104       return false;
105       return true;
106   }
107   /** sets the cell at row i and column j to be empty, i.e.,
108    *  to be 0
109    */
110   public void clear(int i, int j) {
111     this.model[i][j] = 0;
112   }
```

```
113
114     /** checks if val is an acceptable value for the row i */
115     private boolean checkRow(int val, int i) {
116         for( int j = 0; j < 9; j++ )
117                if( this.model[i][j] == val )
118                   return false ;
119
120            return true ;
121     }
122
123     /** checks if val is an acceptable value for the column j */
124     private boolean checkCol(int val, int j) {
125           for( int i = 0; i < 9; i++ )
126                if( this.model[i][j] == val )
127                   return false ;
128
129           return  true;
130     }
131
132     /** checks if val is an acceptable value for the box around
133      *  the cell at row i and column j
134      */
135     private boolean checkBox(int val, int i, int j) {
136
137       i = (i / 3) * 3 ;
138       j = (j / 3) * 3 ;
139
140          for( int r = 0; r < 3; r++ )
141             for( int c = 0; c < 3; c++ )
142             if( this.model[i+r][j+c] == val )
143                return false ;
144
145          return true ;
146     }
147
148  }
```