CS21120

Sudoku Solver

Laura Collins

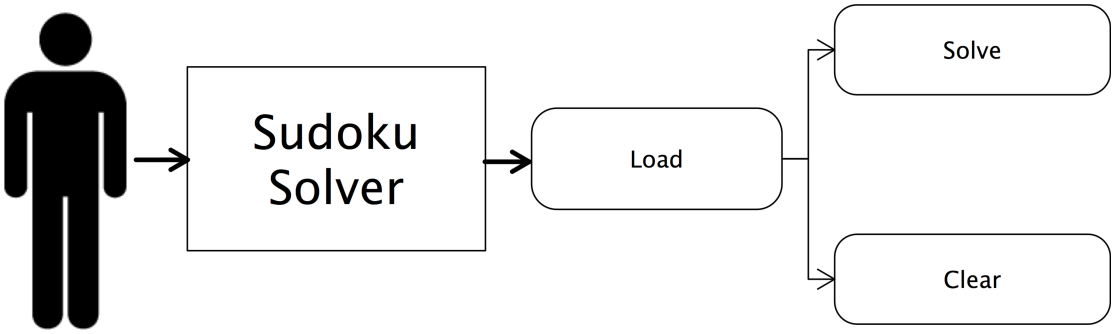
ABSTRACT

Design an algorithm to solve a Sudoku puzzle, design the data structures to enable the algorithms work and produce a well-structured and commented java program with evidence of testing.

Table of Contents

Analysis

Use Case Diagram



The Problem

Create a Sudoku Solving program, so puzzles can be loaded in, and solved quickly and easily. This can be done by a number of different solving algorithms for solving Sudokus, but brute-force must not be used. The Sudoku file must be of '.sud' extension and be a 9x9 board of the numbers 1-9 and spaces for empty values.

How I approached this problem

Due to the time restraints of this project, I believed solving the Sudoku in the Console would be the first priority, with just a simple GUI with buttons to control the functions of 'load' and 'solve'. I applied prior knowledge to loading in text files into a program, and then set about researching solving algorithms for Sudokus.

The grid in my GUI is drawn using Graphics and paintComponent, and is a simple 9x9 grid, which should have been filled with the puzzle, although due to time-constraints, was not completed. Three simple buttons of 'Load', 'Solve' and 'Clear' control the actions on the board.

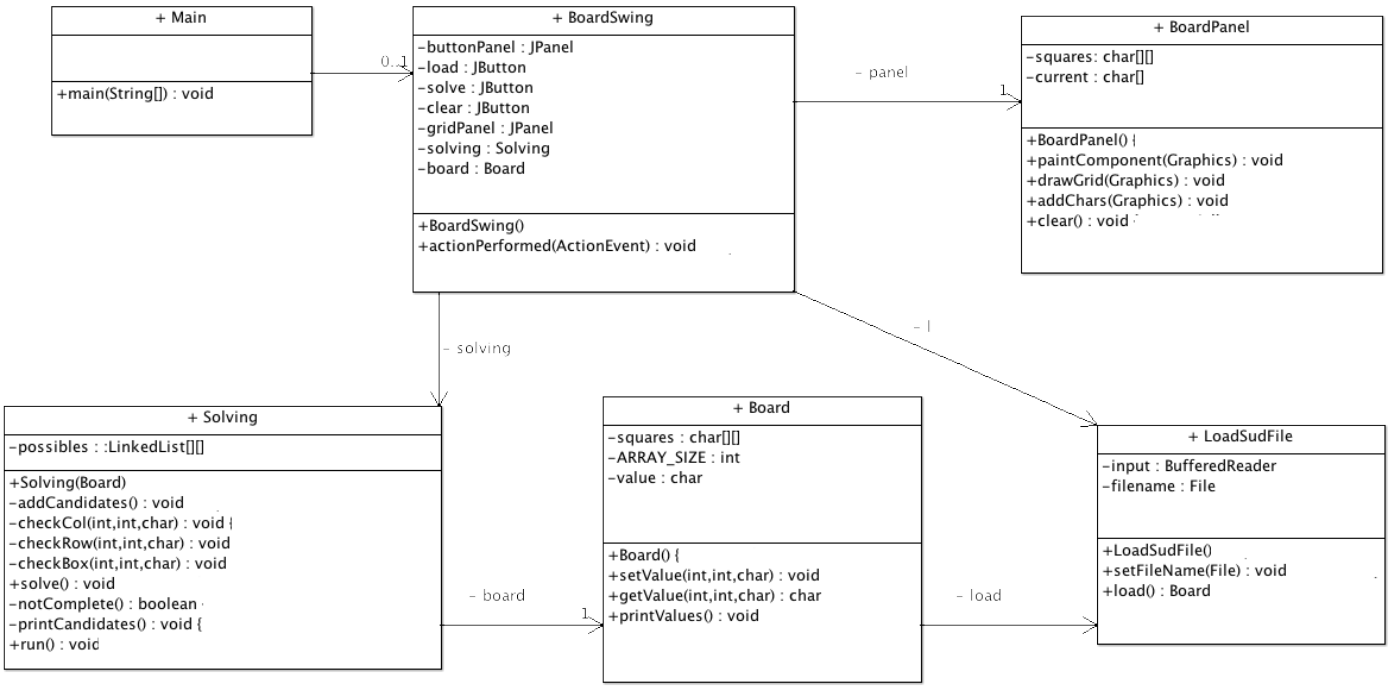
My Sudoku board is made of a 2D Array, containing chars (1-9 and space), so can be read in directly from file, and no conversion into integers is needed. So each square in the board is assigned a value, be it a number or a space, so when the puzzle is loaded in, it's added to the array, and then printed out in the console.

When it comes to solving a puzzle, first it must be noticed which squares are empty, then give them a list of possible candidate values, to be whittled down to 1 hopefully.

When whittled down to one it is inputted into the board and the solving algorithm runs again, checking values until eventually every box has only 1 possible value and the Sudoku is complete.

Design

Class Diagram



My Data Structure and why I chose it

I decided to implement the board as a 2D array, simply because it's a set board of 9x9 squares, and it's simple to iterate through an array of this type using 'for' loops and 'if' statements. Each square has a space in the array, and can be assigned two integer coordinates, an 'x' coordinate for it's row and a 'y' coordinate for it's column, and also can be assigned a char Value.

For example: At coordinate 0,0 in simple349.sud is the Value '2'.

Each square also contains a LinkedList of possible candidates, which is eventually whittled down from the initial 1-9 values, by a series of searching algorithms, removing char values as they're encountered in the row, column or 3x3 box.

I only implemented one common Sudoku-solving algorithm, Naked Singles, although I did research into other algorithms I could've added if I'd had more time, such as Hidden Singles, Naked Pairs, Naked Triples etc.

My program, whilst being Object-Orientated, as Java should be, is as compact and concise as I feel necessary.

Class Descriptions

Main

This runs the entire program by instantiating a new instance of the BoardSwing class, and that's all. It's an extremely simple class using only the Class's constructor to call the Swing and set the window's location on the screen.

BoardSwing

This created the GUI for the program, so the Sudoku board, and three buttons for control of the functions. The three buttons are placed upon a JPanel called 'buttonPanel', and control the loading in of a file, the solving of a puzzle, and clearing the GUI board of any values. Within this class I also set the close method, so the program stops running when the window is closed, especially useful for use on Macs, as closing the window doesn't automatically close a program. There is code in here to help with the loading function, with use of a JFileChooser, so a puzzle can be easily found and selected.

BoardPanel

This draws the Sudoku board by using the Graphics object and paintComponent. This Class should draw the chars from the array squares onto the GUI and then also enable the User to clear the board of all values, but due to time-restraints, this has not been implemented.

Board

This is where the array of chars containing each value for each square is contained and managed. A new 2d Array of chars is created and set to the dimensions of 9x9 and each square is set to a value when it is loaded in, be it 1-9 or ' '(space). Can also return the values of each square, so the Sudoku board can be printed out in the Console.

Solving

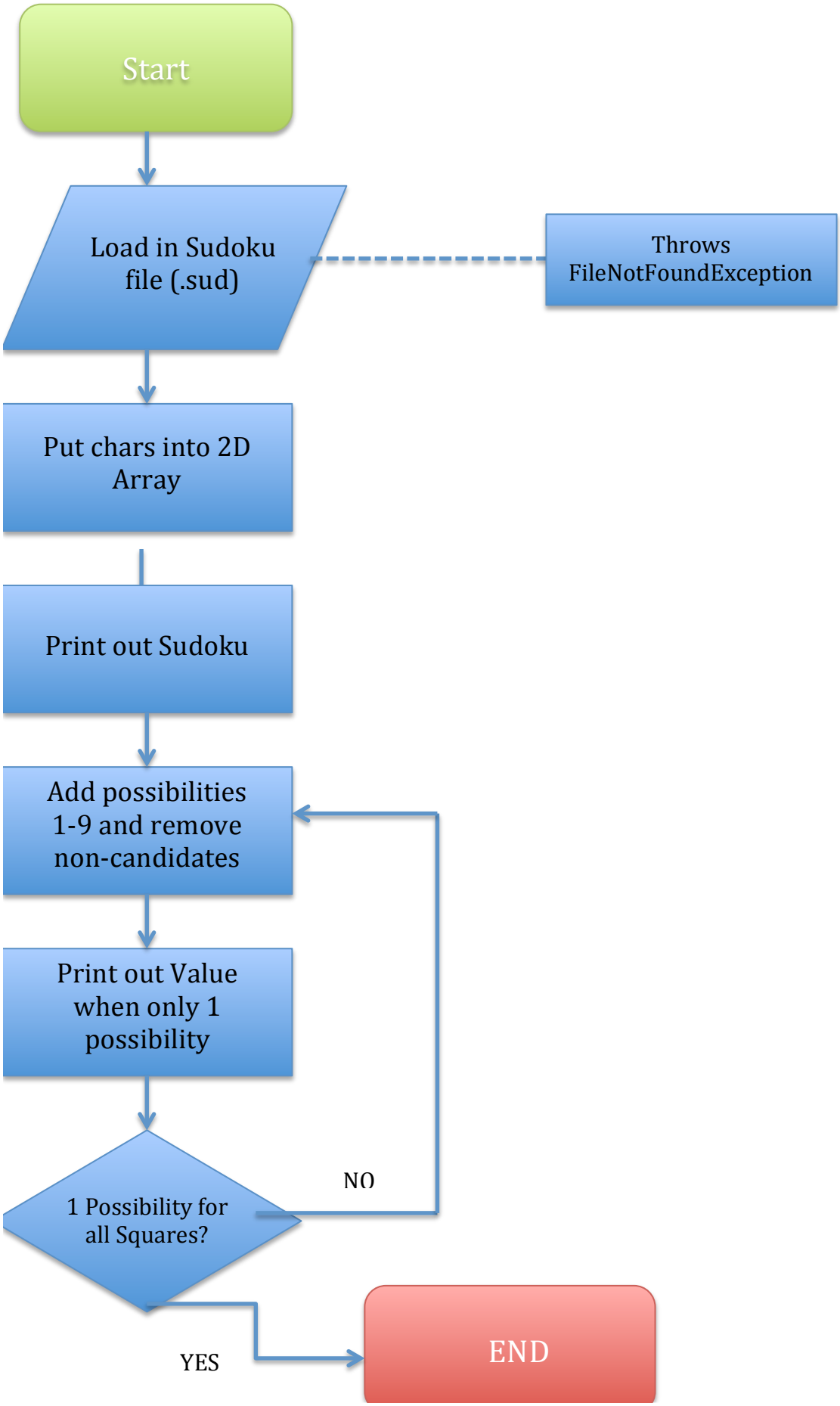
This is by far the most complex class in the program, aimed at solving the Sudoku puzzle. A list of possibilities is created for all the empty squares on the board, and filled with the numbers 1-9, numbers are then eliminated if they are found in the same row, column or 3x3 box as the square. When there is only one possibility left that then becomes the assigned value of the square and is printed on the Sudoku board. As it can't solve all squares in one go, it loops through this elimination method until it solves. If the puzzle cannot be solved, it loops infinitely, so must be stopped to avoid crashing the program.

LoadSudFile

This is a basic Class that simply reads in the file and assigns it to the corresponding square in the board. This is done by reading in each line individually, then splitting them up into separate chars and adding them to the squares array load it in. Each value is then printed out to create the Sudoku board in the Console, with strategically placed '\n' for line breaks.

Algorithms

Sudoku Algorithm Flowchart



Naked Singles Algorithm Pseudo-code

Add Candidates

```
FOR 9 iterations of integer i
        FOR 9 iterations of integer j
                IF square is empty
                        SET new LinkedList
                        ADD char '1'
                        ADD char '2'
                        ADD char '3'
                        ADD char '4'
                        ADD char '5'
                        ADD char '6'
                        ADD char '7'
                        ADD char '8'
                        ADD char '9'
                ENDIF
        ENDFOR
ENDFOR
```

Check Column

```
FOR 9 iterations of integer x
        IF square is not empty
                FOR the size of the possibilities LinkedList number of iterations
                        IF the value of the square equals a value in the Possibilities LinkedList
                                Delete value from Possibilities LinkedList
                        ENDIF
                ENDFOR
        ENDIF
ENDFOR
```

Check Row

```
FOR 9 iterations of integer y
        IF square is not empty
                FOR the size of the possibilities LinkedList number of iterations
                        IF the value of the square equals a value in the Possibilities LinkedList
                                Delete value from Possibilities LinkedList
                        ENDIF
                ENDFOR
        ENDIF
ENDFOR
```

Check 3x3 Box

```
FOR every 3x3 box (Iterate 9 times, using different coordinates)
        FOR 9 iterations of integer x
                FOR 9 iterations of integer y
                        IF the square is not empty
                                FOR the size of the possibilities LinkedList number of iterations
                                        IF the value of the square equals a value in the Possibilities LinkedList
                                                Delete value from Possibilities LinkedList
                                        ENDIF
                                ENDFOR
                        ENDIF
                ENDFOR
        ENDFOR
ENDFOR
```

Check if Puzzle is solved

```
SET integer k equal to 0
FOR 9 iterations of integer i
        FOR 9 iterations of integer j
                IF the square is empty
                        INCREMENT k
                ENDIF
        ENDFOR
ENDFOR
IF integer k is greater than 0
        Return TRUE
ENDIF
Return False
```

Run the Solve Method

```
WHILE above algorithm equals true, so puzzle is not complete
        FOR 9 iterations of integer j
                FOR 9 iterations of integer i
                        IF the square is empty
                                Check Column for chars method
                                Check Row for chars method
                                Check 3x3 Box for chars method
                                IF the possibilities LinkedList size is equal to 1
                                        SET the value in that square to that possibility
                                ENDIF
                        ENDIF
                ENDFOR
        ENDFOR
ENDWHILE
```

Testing

My Approach

I took a practical approach to testing, running the program and trying to break it by entering bad data/pressing buttons when nothing is loaded in.

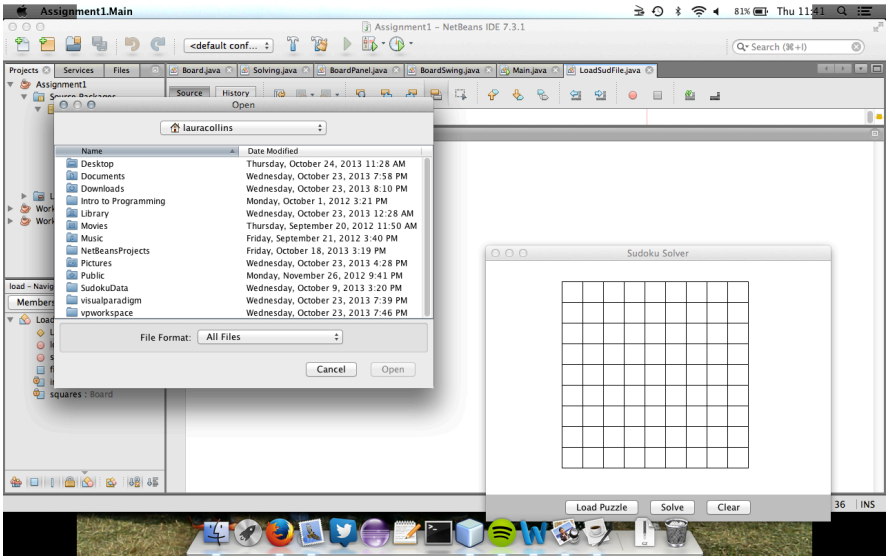
On the next page is a Test Table with 10 different tests, some of which do not have corresponding screenshots as it's not possible to show a window closing etc. in some pictures. I tested loading in a file, cancelling the load function, the solve button, the clear button, threading and the ability to quit easily on a Mac.

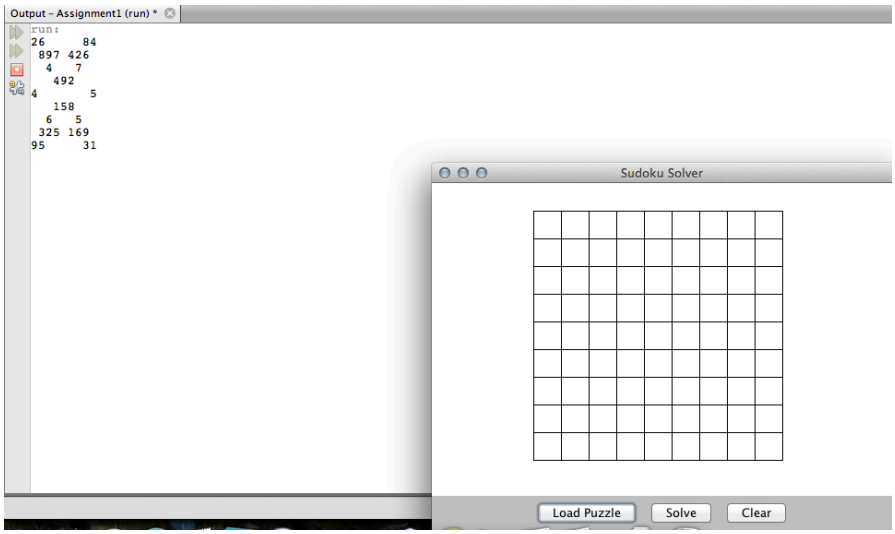
I did several JUnit tests as well, really testing my knowledge in the area, as I'm not the most confident in implementing them.

Test Table

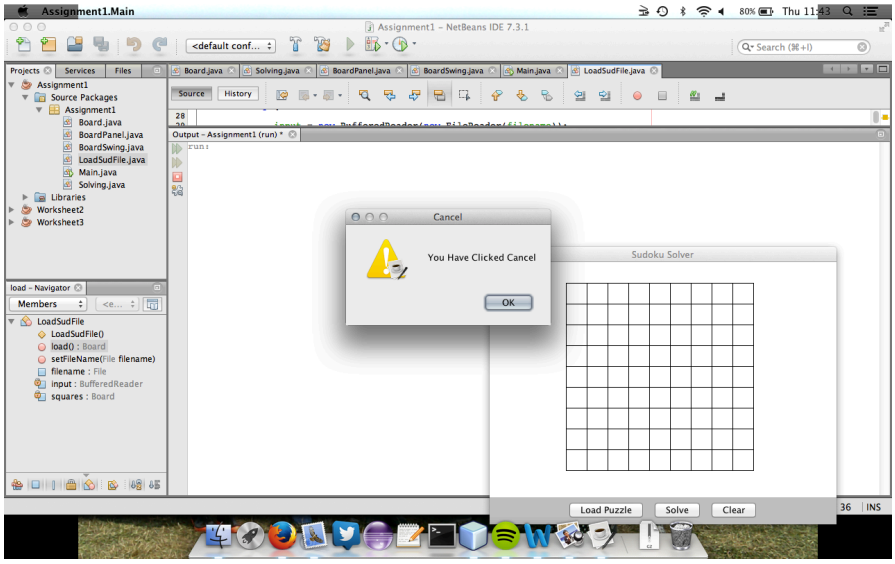
No.	Description	Input	Expected Outcome	Pass/Fail	Comments
1	Loading in an .sud file	Click on 'load' JButton in JPanel	Sudoku is loaded in	Pass	Loads Sudoku into Console
2	Pressing cancel on load window	Click 'Cancel' button in JFileChooser	Window is closed without error	Pass	Warning message appears telling user they've clicked cancel.
3	Loading in a file which is not an '.sud' file	Attempting to load in a jpg image	Warning Message appears	Fail	No Error-Catching implemented
4	Pressing 'solve' when no puzzle is loaded in	Click on 'solve' JButton in JPanel	Warning Message appears	Pass	
5	Pressing 'solve' when a puzzle is loaded in	Click on 'solve' JButton in JPanel	Solves Sudoku	Pass	Solves the two Simple Sudokus
6	Pressing clear when not puzzle is loaded in	Click on 'clear' JButton in JPanel	Shows Warning Message	Pass	
7	Pressing clear when puzzle is loaded in	Click on 'clear' JButton in JPanel	Clears board	Minor Pass	No easy way to clear the console, but prints enough lines for console to appear empty.
8	Pressing clear when puzzle is loading in, then loading in another puzzle and solving	Click on 'clear' JButton in JPanel, then Click 'load' and 'solve' JButtons	Clears board, loads in new puzzle, solves it	Pass	
9	Being able to quit when puzzle is trying to solve	Click 'x' in window	Closes window	Pass - No Screenshot	Thanks to Threads, when the puzzle cannot be solved and gets stuck in an infinite loop, the window can be closed
10	Being able to close window using 'x' on a Mac	Click 'x' in window	Closes window	Pass- No Screenshot	

1.

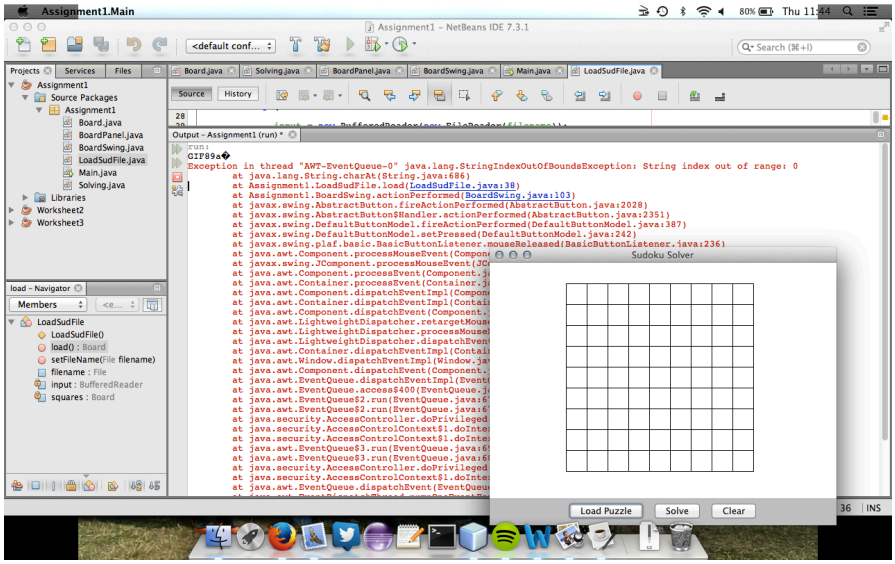

Assignment1.Main
Assignment1 – NetBeans IDE 7.3.1
Open
lauracollins
Name
Date Modified
Desktop
Documents
Downloads
Intro to Programming
Library
Movies
Music
NetBeansProjects
Pictures
Public
SudokuData
visualparadigm
vpworkspace
File Format:
All Files
Cancel
Open
Sudoku Solver
Load Puzzle
Solve
Clear


Output – Assignment1 (run) *
run:
26 84
897 426
4 7
492
4 5
158
6 5
325 169
95 31
Sudoku Solver
Load Puzzle
Solve
Clear

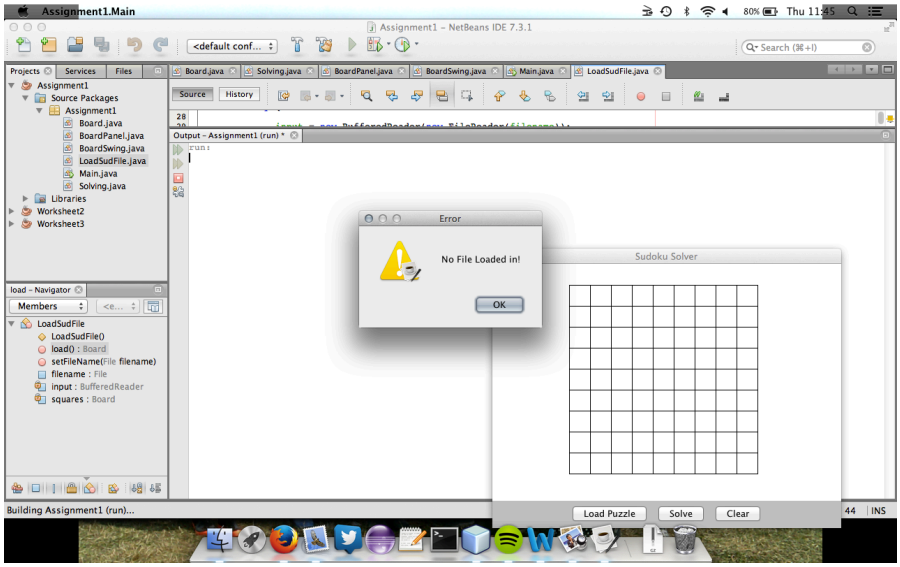
2.


Assignment1.Main
Assignment1 – NetBeans IDE 7.3.1
Cancel
You Have Clicked Cancel
OK
Sudoku Solver
Load Puzzle
Solve
Clear

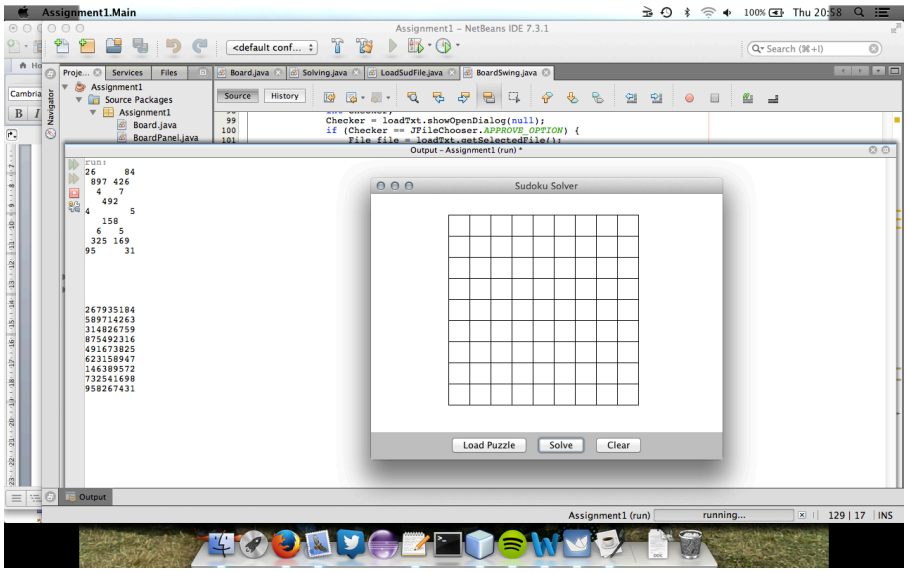
3.


Assignment1.Main
Assignment1 – NetBeans IDE 7.3.1
Sudoku Solver
Load Puzzle
Solve
Clear

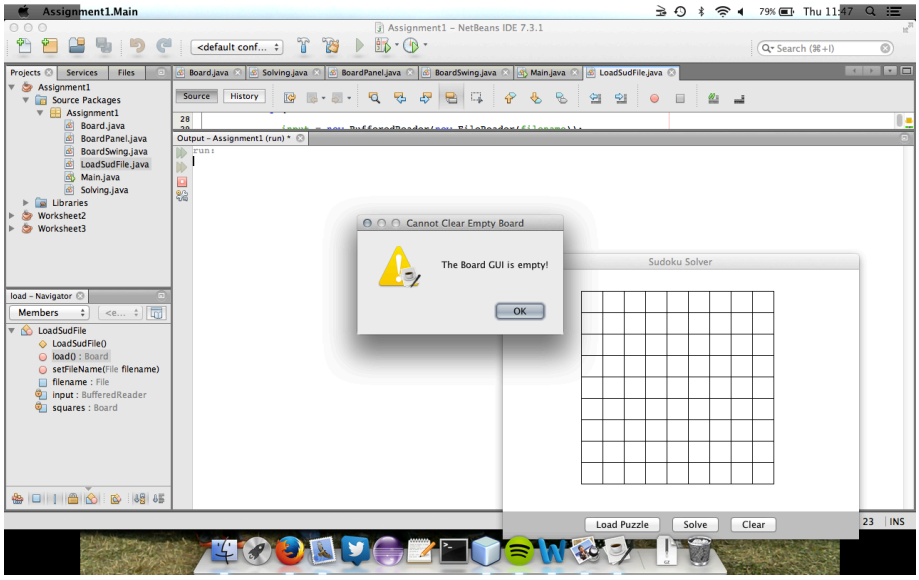
4.


Assignment1.Main
Assignment1 – NetBeans IDE 7.3.1
Error
No File Loaded in!
OK
Sudoku Solver
Load Puzzle
Solve
Clear

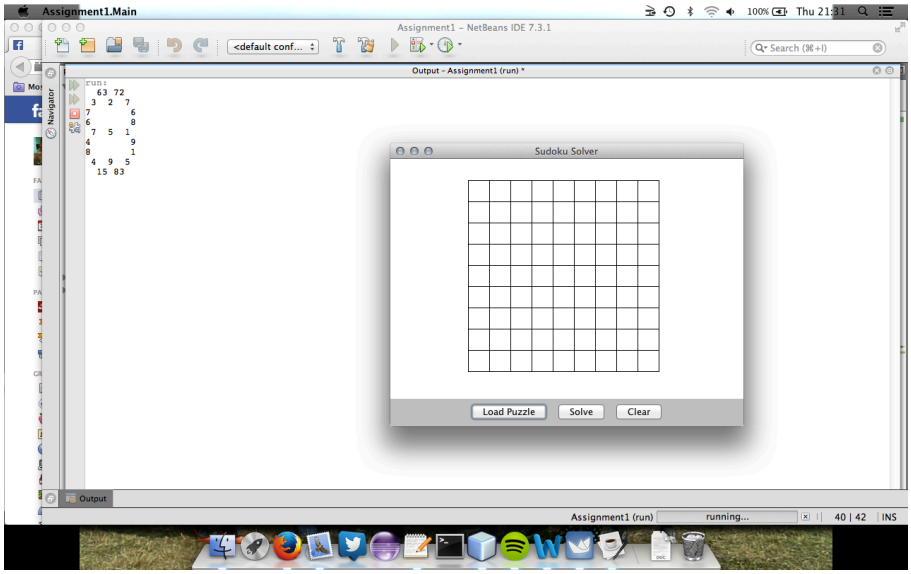
5.

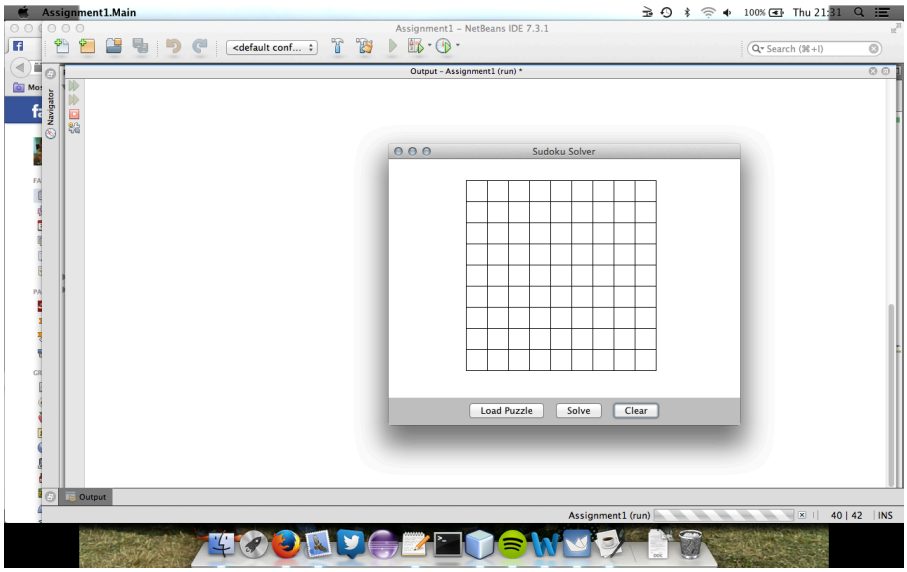

Assignment1.Main
Assignment1 – NetBeans IDE 7.3.1
Sudoku Solver
Load Puzzle
Solve
Clear

6.

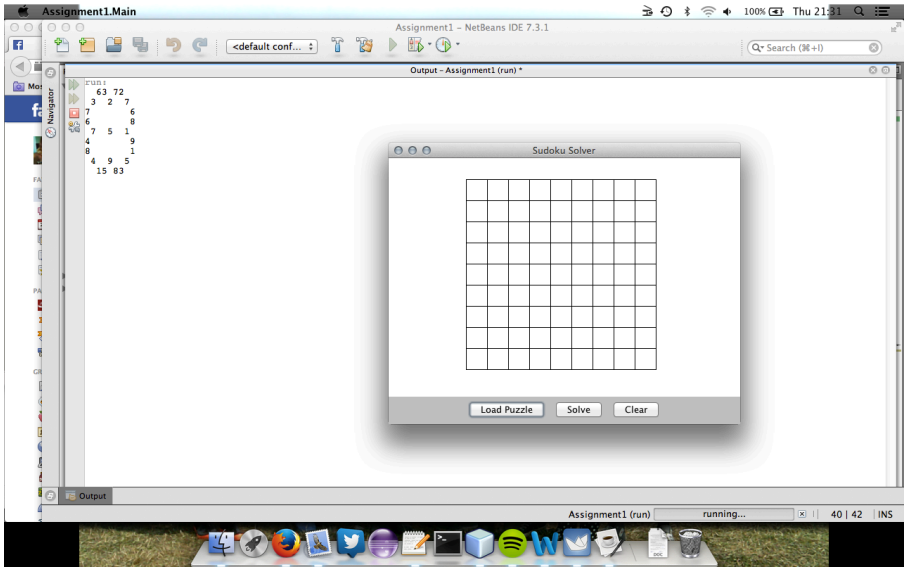

Assignment1.Main
Assignment1 – NetBeans IDE 7.3.1
Cannot Clear Empty Board
The Board GUI is empty!
OK
Sudoku Solver
Load Puzzle
Solve
Clear

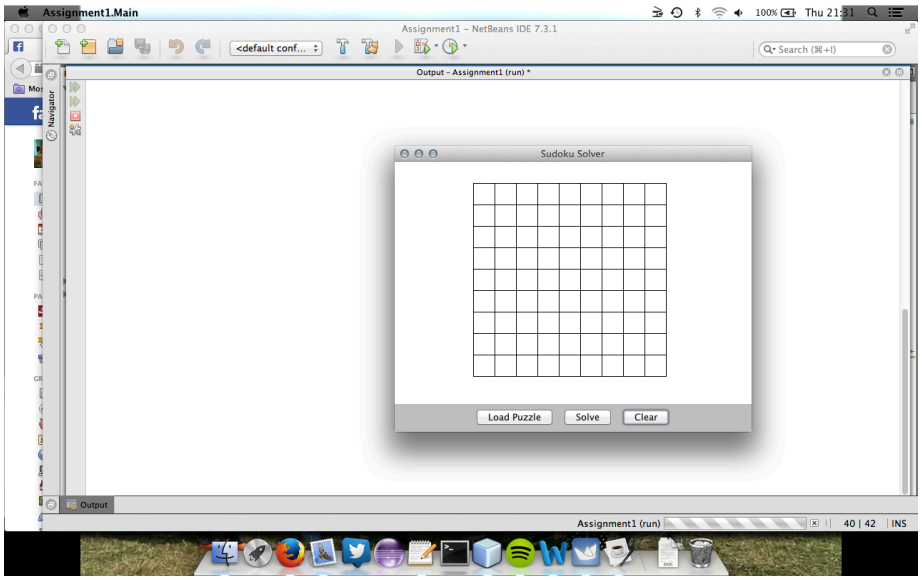
7.

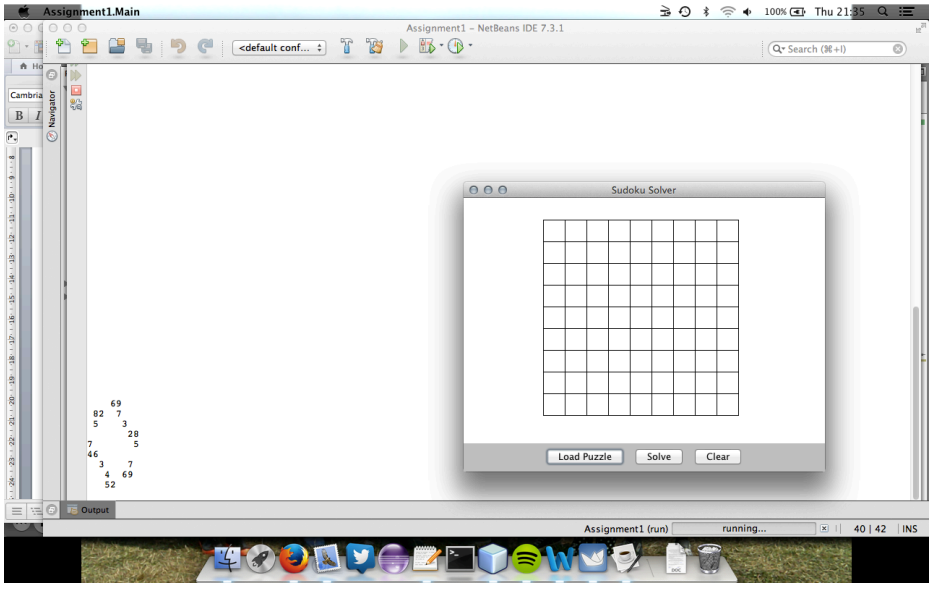

Assignment1.Main
Assignment1 – NetBeans IDE 7.3.1
Sudoku Solver
Load Puzzle
Solve
Clear


Assignment1.Main
Assignment1 – NetBeans IDE 7.3.1
Sudoku Solver
Load Puzzle
Solve
Clear

8.


Assignment1.Main
Assignment1 – NetBeans IDE 7.3.1
Output – Assignment1 (run) *
Sudoku Solver
Load Puzzle
Solve
Clear
Assignment1 (run)
running...


Assignment1.Main
Assignment1 – NetBeans IDE 7.3.1
Output – Assignment1 (run) *
Sudoku Solver
Load Puzzle
Solve
Clear
Assignment1 (run)


Assignment1.Main
Assignment1 – NetBeans IDE 7.3.1
Sudoku Solver
Load Puzzle
Solve
Clear
Assignment1 (run)
running...

Solving 3 Puzzles

Simple349

```
26      84
 897 426
  4   7
   492
4        5
   158
  6   5
 325 169
95      31
```

```
267935184
589714263
314826759
875492316
491673825
623158947
146389572
732541698
958267431
```

Simple 351

```
  49 2  1
 62   73
5
  874 6
6       7
  5 361
        6
 76   81
9  2 14
```

```
834972561
162458739
597613248
318749652
649125387
725836194
451387926
276594813
983261475
```

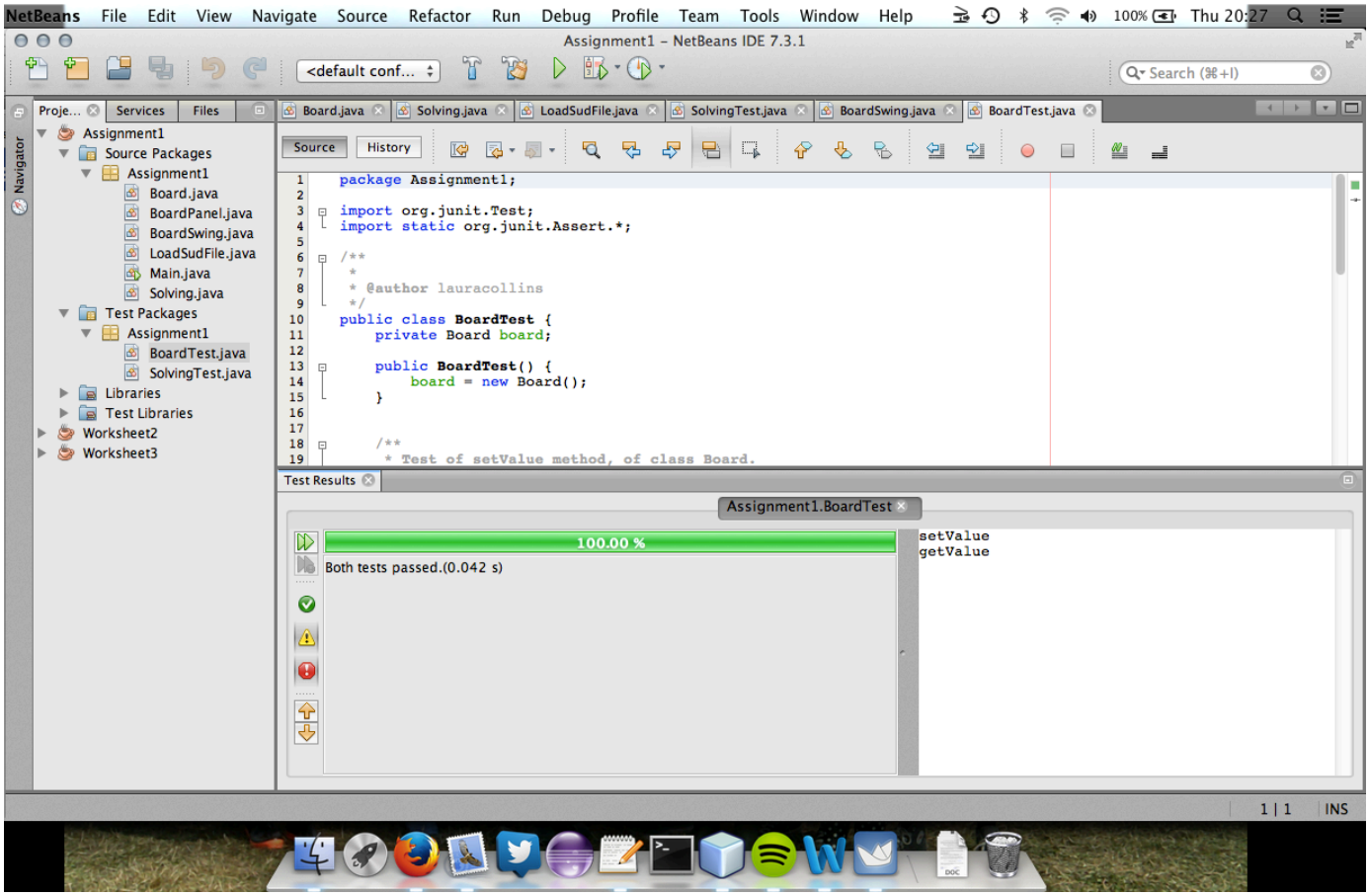
web

```
run:
 6 1 4 5
  83 56
2       1
8  4 7  6
  6   3
7  9 1  4
5       2
  72 69
 4 5 8 7
```

```
 6 1 4 5
  83 56
2  6 9  1
8  437  6
  68523
7  961  4
5  7 3  2
  72 69
642598173
```

JUnit Testing

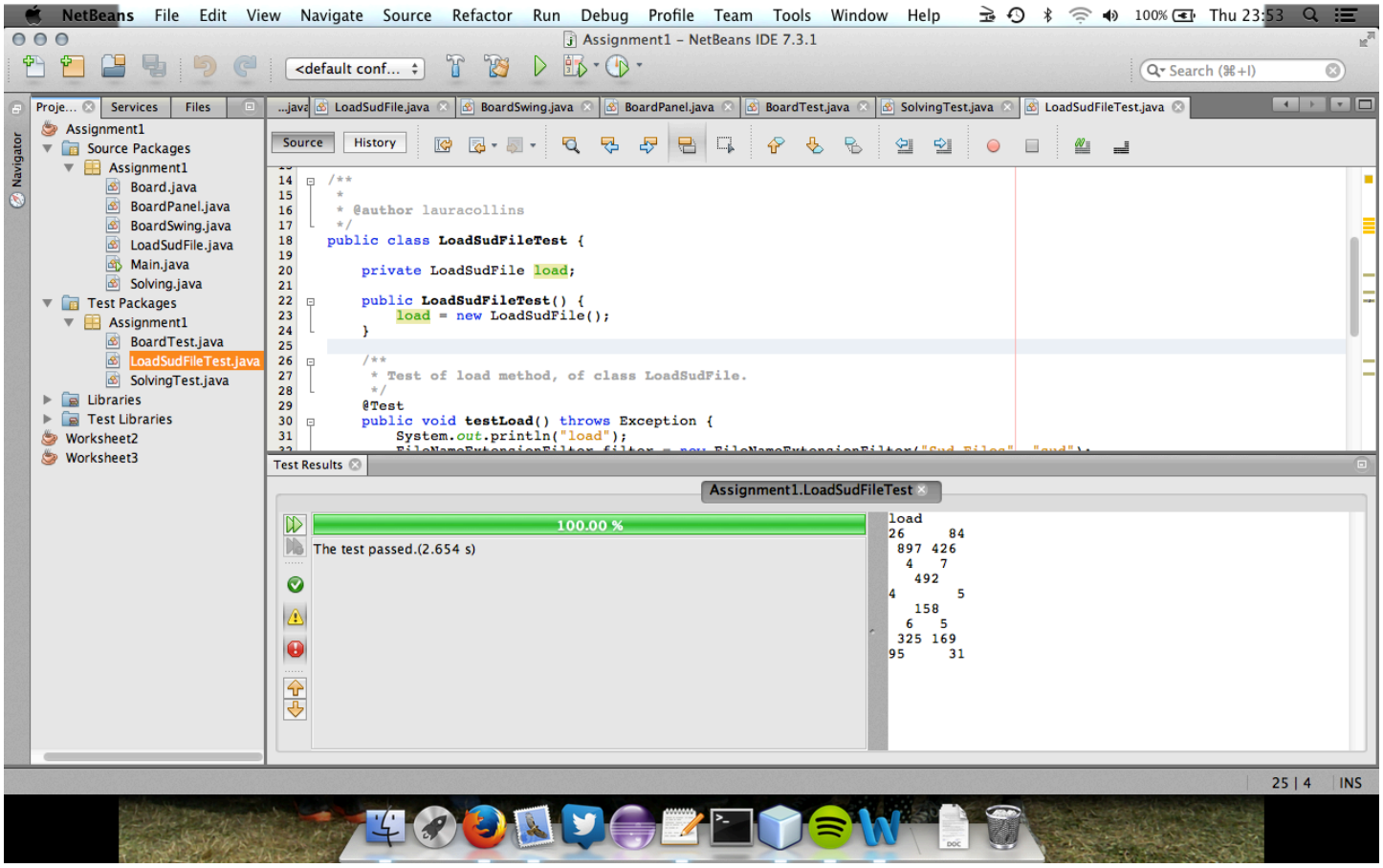
1. Board Class



2 Tests:

1) testSetValue - Tests the method setValue by using it to assign the char value ‘c’ to the square 0,0 and then compared the char ‘c’ to the output of the getValue method. They were equal so the test returns true.
2) testGetValue - Similar to set Value. Uses setValue to set a value at square 0,0, then tests if the output from getValue is the same to the value inputted earlier.

2. LoadSudFile Class



1 Test:

1) Tests the load method using JFileChooser and asserts that the Board Result is not Null, even though it was instantiated as Null.
   The Board can clearly be seen loaded in in the bottom right-hand window.

Conclusion

All-in-all, this assignment has been a challenge, but one that has really highlighted my strengths and weaknesses in programming such a programme. I found Arrays to be my biggest Challenge, and because of the time lost trying to implement my 2D arrays and then my LinkedList, I lost out on implementing my GUI fully.

I also found JUnit once again to be a difficult concept, although I did manage to do some useful tests with them.

As for the outcome of this assignment, I am proud of my efforts, for being able to solve 2 Sudokus, and have clearly identified areas in my knowledge I need to improve upon.

Estimated Time spent on Project: 40+ hours

Source Code

Main

```
package Assignment1;

import java.io.FileNotFoundException;

/**
 * @author lauracollins (lac32)
 * Date Started: 16th October
 *
 * Very Simple Class to run the program.
 */

    /**
     * @throws FileNotFoundException
     */

public class Main
{
    public static void main(String[] args) throws FileNotFoundException
    {
        BoardSwing b = new BoardSwing();
        b.setLocation(700, 350);
    }
}
```

BoardSwing

```
package Assignment1;

import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.logging.*;
import javax.swing.*;
import javax.swing.filechooser.FileNameExtensionFilter;


/**
 * Implements the Sudoku Solver's GUI
 *
 * @author lac32
 * Date Started - 12th Oct. 2013
 *
 */

public class BoardSwing extends JFrame implements ActionListener {
    private JPanel buttonPanel;
    private JButton load;
    private JButton solve;
    private JButton clear;
    private BoardPanel panel;
    private JPanel gridPanel;
    private Solving solving;
    private Board board;
    private Thread calc;
```

```
    /*
     * Main constructor containing most of the Swing implementation.
     * Creates two JPanels for the window, and three buttons.
     * Also calls from the BoardPanel class to draw the basic grid.
     * @throws FileNotFoundException
     */

    public BoardSwing() throws FileNotFoundException {
        setTitle("Sudoku Solver");

        buttonPanel = new JPanel();
        gridPanel = new JPanel(new GridLayout(1,1));

        load = new JButton("Load Puzzle");
        load.addActionListener(this);
        buttonPanel.add(load);

        solve = new JButton("Solve");
        solve.addActionListener(this);
        buttonPanel.add(solve);

        clear = new JButton("Clear");
        clear.addActionListener(this);
        buttonPanel.add(clear);

        buttonPanel.setBackground(Color.LIGHT_GRAY);

        panel = new BoardPanel();

        gridPanel.add(panel);

        add(buttonPanel, BorderLayout.SOUTH);
        add(gridPanel, BorderLayout.CENTER);
        setSize(500, 400);
      this.setVisible(true);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //On Mac, closing via 'x'
does not stop application

    }

    public void mouseClicked(MouseEvent me) {
        throw new UnsupportedOperationException("Not supported yet.");
    }

    public void mousePressed(MouseEvent me) {
        throw new UnsupportedOperationException("Not supported yet.");
    }

    public void mouseReleased(MouseEvent me) {
        throw new UnsupportedOperationException("Not supported yet.");
    }

    public void mouseEntered(MouseEvent me) {
        throw new UnsupportedOperationException("Not supported yet.");
    }

    public void mouseExited(MouseEvent me) {
        throw new UnsupportedOperationException("Not supported yet.");
    }

    /**
     * actionPerfomed is a automatically-generated method when ActionListener is
implemented.
     * Controls what happens when a user clicks a specific button, and calls appropriate
methods.
     *
```

```
     * @param ae
     */

    @Override
    public void actionPerformed(ActionEvent ae) {
        if (ae.getSource() == load) {
        try {
            LoadSudFile l = new LoadSudFile();
            FileNameExtensionFilter filter = new FileNameExtensionFilter("Sud Files",
"sud");
            JFileChooser loadTxt = new JFileChooser();
            int Checker;
            Checker = loadTxt.showOpenDialog(null);
            if (Checker == JFileChooser.APPROVE_OPTION) {
                File file = loadTxt.getSelectedFile();
                loadTxt.setFileFilter(filter);
                l.setFileName(file);
                board = l.load();

            }
            else {
                JOptionPane.showMessageDialog(null, "You Have Clicked Cancel", "Cancel",
JOptionPane.WARNING_MESSAGE); //Stops it throwing Null Pointer when load is cancelled
            }

                        }
        catch (FileNotFoundException e) {
                JOptionPane.showMessageDialog(null, "File Not Found", "Error",
JOptionPane.WARNING_MESSAGE); //Warning Message if there'board an error loading in a file
            }
        catch (IOException ex) {
            Logger.getLogger(BoardSwing.class.getName()).log(Level.SEVERE, null, ex);
//Automatically generated
        }
        }

        if (ae.getSource() == solve) {

            if (board != null){

                solving = new Solving(board);
                calc = new Thread(solving);
                calc.start();
            }
            else {
                JOptionPane.showMessageDialog(null, "No File Loaded in!", "Error",
JOptionPane.WARNING_MESSAGE);
            }
        }

        if (ae.getSource() == clear) {
            if (board != null){
            panel.clear();
            }
            else {
                JOptionPane.showMessageDialog(null, "The Board GUI is empty!", "Cannot
Clear Empty Board", JOptionPane.WARNING_MESSAGE); //Stops it throwing Null Pointer when
clear is clicked as GUI is empty

            }
        }
    }
 }
```

BoardPanel

```
package Assignment1;

import java.awt.*;
import javax.swing.*;


/**
 * Basic Class drawing the grid in the GUI, and controlling the 'Clear' function.
 *
 * @author Laura Collins lac32
 * Date Started: 12th October
 */
public class BoardPanel extends JPanel {

    /*
     * BoardPanel constructor, simply sets the background colour.
     */
    public BoardPanel() {
        setBackground(Color.white);
    }

    /*
     * Calls paintComponent, saw Grid can be drawn.
     *
     * @param g
     */
    @Override
    public void paintComponent(Graphics g) {
              super.paintComponent(g);
              drawGrid(g);
    }

    /*
     * Draws the grid using simple For loops and the Graphics component drawRect.
     *
     * @param g
     */
   public void drawGrid(Graphics g) {
              int i, j, begini = 110, beginj = 30;

                    for (i = 0; i < 9; i++) {
                      for (j = 0; j < 9; j++) {
                            g.setColor(Color.black);
                            g.drawRect(begini + i * 30, beginj + j * 30, 30, 30);
                      }
                    }
                }
   /*
    * Originally going to be used for clearing the GUI board of all values.
    * Now being used to make Console appear to have been cleared.
    * A clear method for the Console has yet to be properly defined.
    */

    public void clear() {
        for (int i=0; i<9; i++){
            for (int j=0; j<9; j++) {
                System.out.println(" ");
            }
        }
    }

}
```

Board

```
package Assignment1;

/**
 * Is a board of Squares, setting values to each square, returning these values.
 * Also has a Print method for print out the Sudoku board once it is completed.
 *
 * @author Laura Collins (lac32)
 * Date started - 14th Oct. 2013
 */

public class Board {

    private char squares[][];
    private static final int ARRAY_SIZE = 9;
    private char value;

    /*
     * Instatiate the 2D Arrays 'squares' and set it's size to be the final array size
of 9x9
     */

    public Board() {
           squares = new char[ARRAY_SIZE][ARRAY_SIZE];
    }

    /*
     * Sets Value of a Square in the Board
     * @param i Row value
     * @param j Column value
     * @param Value the char value
     */

    public void setValue(int i, int j, char value) {
        this.value = value;
        squares[i][j] = value;
    }

    /*
     * Returns the value of a Square in the board
     * @param i Row value
     * @param j Column value
     * @return char Value of the square
     */

    public char getValue(int i, int j) {
        return squares[i][j];
    }

    /*
     * Prints out the Board.
     * For use in the final solving algorithm.
     */
    public void printValues() {
        for (int i = 0; i<9; i++) {
            for (int j = 0; j<9; j++) {
                System.out.print(getValue(i,j));
            }
            System.out.print('\n');
         }
    }

}
```

Solving

```
package Assignment1;

import java.util.LinkedList;

/**
 *
 * Contains Solving algorithms for finding the Sudoku's solution.
 * Uses the Naked Singles solving algorithm, so finds squares that have only one possibility and inputting it.
 * Repeats this until all squares are filled.
 * This solving algorithm can solve 2 simple Sudokus.
 *
 * @author Laura Collins (lac32)
 * Date Started: 16th October 2013
 *
 */

public class Solving implements Runnable {

    private Board board;
    private LinkedList <Character> possibles[][] = new LinkedList[9][9];

    /*
     * Links board from the Board class, into this Class
     * @param board Board array of squares
     */
    public Solving(Board squares) {
        this.board = squares;
    }

    /*
     * Checks if char is empty, create a new linkedlist of possibilities, and then adds possible values
     * that don't come up in the corresponding row, column or box.
     */

    private void addCandidates() {
       // char current='c';
        for (int i=0; i<9; i++){
            for (int j=0; j<9; j++) {
               if (board.getValue(i,j) == ' ') {
                   possibles[i][j] = new LinkedList();
                   possibles[i][j].add('1');
                   possibles[i][j].add('2');
                   possibles[i][j].add('3');
                   possibles[i][j].add('4');
                   possibles[i][j].add('5');
                   possibles[i][j].add('6');
                   possibles[i][j].add('7');
                   possibles[i][j].add('8');
                   possibles[i][j].add('9');
                 }
            }
        }
        System.out.print("\n");

    }
```

```
    /*
     * Iterates through each row in a column, finding chars and removing them from the
Possibility LinkedList.
     * Then iterates through all 9 columns.
     * @param i
     * @param j
     */

    private void checkCol(int i, int j){

        for (int x = 0; x < 9; x++) {
            if (board.getValue(x, j) != ' '){
            for (int k = 0; k < possibles[i][j].size(); k++) {
                if (board.getValue(x, j) == possibles[i][j].get(k)) {
                    possibles[i][j].remove(k);
                }
                }
            }

        }
    }

   /*
    * Iterates through each column in a row, finding chars and removing them from the
Possibility LinkedList.
    * Then iterates through all 9 rows.
    * @param i
    * @param j
    */

    private void checkRow(int i, int j) {
        for (int y = 0; y < 9; y++) {
            if (board.getValue(i, y) != ' '){
                for (int k = 0; k < possibles[i][j].size(); k++) {
                     if (board.getValue(i, y) == possibles[i][j].get(k)) {
                         possibles[i][j].remove(k);
                     }
                }
            }
        }
    }

    /*
     * Checks 3x3 Box, one at a time, starting with the upper-leftmost box, and
iterating right.
     * Once it gets to the last 3x3 box on that row, then checks the next row down etc.
     * Changing the i and j changes between which coordinates to check.
     * @param i
     * @param j
     */

    private void checkBox(int i, int j) {
        if (i<3 && j<3) {
            for (int x = 0; x<3; x++) {
                for (int y = 0; y<3; y++) {
                    if (board.getValue(x, y) != ' ') {
                        for (int k =0; k < possibles[i][j].size(); k++) {
                            if (board.getValue(x, y) == possibles[i][j].get(k)) {
                                possibles[i][j].remove(k);
                            }
                        }
                    }
                }
            }
        }
```

```
if ((i>2 && i<6) && j<3) {
    for (int x = 3; x<6; x++) {
        for (int y = 0; y<3; y++) {
            if (board.getValue(x, y) != ' ') {
                for (int k =0; k < possibles[i][j].size(); k++) {
                    if (board.getValue(x, y) == possibles[i][j].get(k)) {
                        possibles[i][j].remove(k);
                    }
                }
            }
        }
    }
}

if ((i>5 && i<9) && j<3) {
    for (int x = 6; x<9; x++) {
        for (int y = 0; y<3; y++) {
            if (board.getValue(x, y) != ' ') {
                for (int k =0; k < possibles[i][j].size(); k++) {
                    if (board.getValue(x, y) == possibles[i][j].get(k)) {
                        possibles[i][j].remove(k);
                    }
                }
            }
        }
    }
}

if (i<3 && (j>2 && j<6)) {
    for (int x = 0; x<3; x++) {
        for (int y = 3; y<6; y++) {
            if (board.getValue(x, y) != ' ') {
                for (int k =0; k < possibles[i][j].size(); k++) {
                    if (board.getValue(x, y) == possibles[i][j].get(k)) {
                        possibles[i][j].remove(k);
                    }
                }
            }
        }
    }
}

if ((i>2 && i<6) && (j>2 && j<6)) {
    for (int x = 3; x<6; x++) {
        for (int y = 3; y<6; y++) {
            if (board.getValue(x, y) != ' ') {
                for (int k =0; k < possibles[i][j].size(); k++) {
                    if (board.getValue(x, y) == possibles[i][j].get(k)) {
                        possibles[i][j].remove(k);
                    }
                }
            }
        }
    }
}

if ((i>5 && i<9) && (j>2 && j<6)) {
    for (int x = 6; x<9; x++) {
        for (int y = 3; y<6; y++) {
            if (board.getValue(x, y) != ' ') {
                for (int k =0; k < possibles[i][j].size(); k++) {
                    if (board.getValue(x, y) == possibles[i][j].get(k)) {
                        possibles[i][j].remove(k);
                    }
                }
            }
        }
    }
```

```
                }
            }
        }

        if (i<3 && (j>5 && j<9)) {
            for (int x = 0; x<3; x++) {
                for (int y = 6; y<9; y++) {
                    if (board.getValue(x, y) != ' ') {
                        for (int k =0; k < possibles[i][j].size(); k++) {
                            if (board.getValue(x, y) == possibles[i][j].get(k)) {
                                possibles[i][j].remove(k);
                            }
                        }
                    }
                }
            }
        }

        if ((i>2 && i<6) && (j>5 && j<9)) {
            for (int x = 3; x<6; x++) {
                for (int y = 6; y<9; y++) {
                    if (board.getValue(x, y) != ' ') {
                        for (int k =0; k < possibles[i][j].size(); k++) {
                            if (board.getValue(x, y) == possibles[i][j].get(k)) {
                                possibles[i][j].remove(k);
                            }
                        }
                    }
                }
            }
        }

        if ((i>5 && i<9) && (j>5 && j<9)) {
            for (int x = 6; x<9; x++) {
                for (int y = 6; y<9; y++) {
                    if (board.getValue(x, y) != ' ') {
                        for (int k =0; k < possibles[i][j].size(); k++) {
                            if (board.getValue(x, y) == possibles[i][j].get(k)) {
                                possibles[i][j].remove(k);
                            }
                        }
                    }
                }
            }
        }
    }


    /*
     * Brings all methods together
     */


    public void solve() {
        addCandidates();

        for (int j=0; j<9; j++) {
            for (int i=0; i<9; i++) {
                if (board.getValue(i, j) == ' ') {
                    checkCol(i,j);
                    checkRow(i,j);
                    checkBox(i,j);
                    if (possibles[i][j].size()==1) {
                         board.setValue(i, j, possibles[i][j].getFirst());
                    }
                }
```

```
            }
            //  System.out.print("\n");
        }
        // printCandidates(); //Uncommented, this prints out each box's possibilities
whilst the solving algorithm runs.
        //System.out.print('\n');
        }


    //}

    /*
     * Checks to see if there are any empty squares left.
     * Not the best way to implement a loop so it keeps solving, as for ones it cannot
solve it gets stuck in an infinite loop.
     * To combat this, create thread, so when method is called, the window can still be
closed.
     */

    private boolean notComplete() {
        int k = 0;

        for (int i=0; i<9; i++) {
            for (int j=0; j<9; j++) {
                if(board.getValue(i, j) == ' ') {
                    k++;
            }
        }
    }
        if (k>0) {
            return true;
        }
        return false;
    }

    /*
     * Currently commented out, this is the method for returning which squares are
blank, and what possibilities they contain.
     */

    private void printCandidates() {
        for (int i=0; i<9; i++) {
            for (int j=0; j<9; j++) {
                if (board.getValue(i, j) == ' ') {
                    System.out.print(i + " " + j + ": " + possibles[i][j]);
                }
                 System.out.print('\n');
            }

        }
    }

    /*
     * Needed to implement runnable, so a thread can be created to help with infinite
loops.
     * @Override
     */

    @Override
    public void run() {
        while (notComplete()){
            solve();
        }
        board.printValues();
    }
}
```

LoadSudFile

```
package Assignment1;

import java.io.*;

/**
 * This loads in the .sud file and adds it to the Board array.
 * Each square in the array is assigned a char value, read in from the file.
 * Used BufferedReader to read in the file.
 * Threw several exceptions to catch any unwanted errors.
 *
 * @author lac32
 * Date Started: 12th Oct. 2013
 *
 */
public class LoadSudFile {

    private BufferedReader input;
    public File filename;
    private Board squares = new Board();

    public LoadSudFile() {

    }

    /*
     * So when using the JFileChooser, the appropriate filename is assigned.
     */

    public void setFileName(File filename) {
        this.filename = filename;
    }

    /*
     * Load in .sud file.
     * Improvements could be made here, or in the Swing class so it can only load in
.sud files.
     * Also improvements where the board could be printed in the GUI.
     */

    public Board load() throws FileNotFoundException {

        try {

            input = new BufferedReader(new FileReader(filename));

            for (int x=0; x<9; x++) {

                  String currentLine = input.readLine();

                  for (int y=0; y<9; y++) {
                      squares.setValue(x,y,currentLine.charAt(y));
                      System.out.print(squares.getValue(x, y));
                  }

                System.out.print("\n");
            }
        }

        catch (IOException e){
```

```
            e.printStackTrace();
        }

        finally {
            try {
                if (input != null) {
                     input.close();
                }
            }
            catch (IOException ie) {
                ie.printStackTrace();
            }
        }
        return squares;
    }
}
```

BoardTest

```
package Assignment1;

import org.junit.Test;
import static org.junit.Assert.*;

/**
 * Test Class testing the Board class.
 *
 * @author lauracollins
 * Date Started: 24th October 2013
 */
public class BoardTest {
    private Board board;

    public BoardTest() {
        board = new Board();
    }


    /**
     * Test of setValue method, of class Board.
     */
    @Test
    public void testSetValue() {
        System.out.println("setValue");
        int i = 0;
        int j = 0;
        char value = 'c';

        board.setValue(i, j, value);
        assertEquals("Should get char 'c' at specific coordinates", 'c',
board.getValue(i, j));
    }

    /**
     * Test of getValue method, of class Board.
     */
    @Test
    public void testGetValue() {
        System.out.println("getValue");
        int i = 0;
        int j = 0;
        char expResult = 'c';
        board.setValue(i, j, expResult);
```

```
        char result = board.getValue(i, j);
        assertEquals(expResult, result);

    }

}
```

LoadSudFileTest

```
package Assignment1;

import java.io.File;
import javax.swing.JFileChooser;
import javax.swing.filechooser.FileNameExtensionFilter;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
 * @author lauracollins
 */
public class LoadSudFileTest {

    private LoadSudFile load;

    public LoadSudFileTest() {
        load = new LoadSudFile();
    }

    /**
     * Test of load method, of class LoadSudFile.
     */
    @Test
    public void testLoad() throws Exception {
        System.out.println("load");
        FileNameExtensionFilter filter = new FileNameExtensionFilter("Sud Files",
"sud");
            JFileChooser loadTxt = new JFileChooser();
            int Checker;
            Checker = loadTxt.showOpenDialog(null);
            if (Checker == JFileChooser.APPROVE_OPTION) {
                File file = loadTxt.getSelectedFile();
                loadTxt.setFileFilter(filter);
                load.setFileName(file);
       // Board expResult = null;
        Board result = load.load();

        assertNotNull(result);
            }
    }
}
```