```
public class Sudoku {
	/*
	 * Sudoku.java		Richard Addicott (ria4@aber.ac.uk)
	 * - This class is simply what the code will initiate first when
	 * the program is run at runtime - all it does is create a
	 * SudokuDisplay in the main method of the program.
	 */
	public static void main(String[] args) {
		@SuppressWarnings("unused")
		SudokuDisplay d = new SudokuDisplay();
		System.out.println("Please click SOLVE - MULTIPLE TIMES - to initiate
solving algorithm");
	} // ends main method

} // ends Sudoku class
```

```
public class SudokuModel {
	/*
	 * SudokuModel.java	Richard Addicott (ria4@aber.ac.uk)
	 * - This class provides the rest of the classes with two get and
	 * set methods that are heavily used to store the values of the
	 * Sudoku cells throughout the program. No real algorithms are
	 * present in this class, it is just a reference point for the
	 * rest of the code to call when needed.
	 */
	public char grid[][]; // forms the basis of the grid array
	public int ROWS = 9; // constant variable of 9x9 grid
	public int COLUMNS = 9;

	public SudokuModel(int row, int col) {
		grid = new char[ROWS + 1][COLUMNS + 1];
	} // ends SudokuModel method

	public void setValue(int row, int col, char r) {
		grid[row][col] = r;
	} // ends setValue method

	public char returnValue(int row, int col) {
		char x;
		x = grid[row][col];
		return x;
	} // ends returnValue method

} // ends SudokuModel class
```

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

import javax.swing.*;

@SuppressWarnings("serial")
public class SudokuCanvas extends JPanel {
	/*
	 * SudokuCanvas.java		Richard Addicott (ria4@aber.ac.uk)
	 * - This class contains the code of many of the complex
	 * algorithms that are used to load in, and manipulate the Sudoku
	 * data in order to solve the data. It also provides graphical
	 * tools to produce an output onto the GUI for the user to see.
	 */

	// Variable declaration
	SudokuModel board = new SudokuModel(9,9); // Needed for interaction with Model
class
	int ROWS = 9;
	int COLUMNS = 9;
	char gridArray[][] = new char[9][9]; // Holds coordinate data
	char gridValue[] = new char[81]; // Holds all numerical values
	ArrayList<String> usedNumbers = new ArrayList<String>(); // Holds numbers
found in corresponding cells
	ArrayList<String> possibleNumbers = new ArrayList<String>(); // Holds numbers
that could be solved

	public SudokuCanvas() {
		board = new SudokuModel(ROWS, COLUMNS);
	} // ends SudokuCanvas constructor

	public void paintComponent(Graphics g) {
		// method to drawGrid and provide graphical support
		super.paintComponent(g);
		drawGrid(g);
	} // ends paintComponent method

	public void drawGrid(Graphics g) {
		String value;
		g.setColor(Color.black);
		// Provides graphical outputs using elements
		int width = getSize().width;
		int height = getSize().height;
		// Variables needed to create basic grid structure
		int firstValuei = 20;
		int firstValuej = 25;
		// Loops that perform graphical output to create the grid
		for (int i = 0; i < ROWS; i++) {
			for (int j = 0; j < COLUMNS; j++) {
```

```
                        // Finds if any numerical values are needed to be printed
to the GUI
                        value = String.valueOf(board.returnValue(j,i));
                        g.drawRect((i *40),(j * 40), 40, 40);
                        // Draws the graphic within appropriate cell
                        g.drawString(value, (firstValuei + (i * 40)), (firstValuej
+ (j * 40)));
                  }
            }

            // Additional graphical tools to create 4 thicker lines to divide
subgrids
            Graphics2D g2 = (Graphics2D) g;
            g2.setStroke(new BasicStroke(3));
            g2.drawLine((width / 3), 0, (width / 3), height);
            g2.drawLine((width / 3) * 2, 0, (width / 3) * 2, height);
            g2.drawLine(0, (height / 3), width, (height / 3));
            g2.drawLine(0, (height / 3) * 2, width, (height / 3) * 2);

      } // ends drawGrid method

      public void loadSudoku(String filename) throws IOException {
            BufferedReader br; // Needed for file handling
            // Opens BufferedReader using the filename input from the JTextField
            br = new BufferedReader(new FileReader("src\\" + filename));
            char x; // Declares variable x
            // Uses clearSudoku method to ensure no incorrect values still in data
structures
            clearSudoku();
            // Iterates through the file, reading each line
            for (int i = 0; i < ROWS; i++) {
                  String sCurrentLine = br.readLine();
                  for (int j = 0; j < COLUMNS; j++) {
                        // Parses off the character at index (j) - returning a
value or ' '
                        x = sCurrentLine.charAt(j);
                        // Assigns to gridArray data structure
                        gridArray[i][j] = x;
                        // Sets the value to the graphical board
                        board.setValue(i, j, x);
                        // Sets to '0' for command line output
                        if (x == ' ') {
                              x = '0';
                        }
                        // Command line output
                        System.out.print(x);
                  }
                  System.out.println("");
            }
            // Closes BufferedReader connection
            br.close();
```

```
		// Repaints the graphical output
		repaint();
	} // ends loadSudoku method

	public void clearSudoku() {
		// Iterates through each cell (9,9 grid) - setting everything to ' '
(clearing it)
		for (int i = 0; i < ROWS; i++) {
			for (int j = 0; j < COLUMNS; j++) {
				gridArray[i][j] = ' ';
				board.setValue(i, j, ' ');
			}
		}
		// Repainting the graphical output
		repaint();
	} // ends clearSudoku method

	public void findPossibleCandidates() {
		// Loops through 81 cells, producing the checks if needed
		for (int i = 0; i < ROWS; i++) {
			for (int j = 0; j < COLUMNS; j++) {
				// Checks if cell is empty
				if (gridArray[i][j] == ' ') {
					// Calls check method to produce arrayList of
usedNumbers
					checkRow(i,j);
					checkColumn(i,j);
					checkSquare(i,j);
					solveCell(i,j);
				} else {
					// Cell is not empty, can be left
				}
			}
		}
	} // ends findPossibleCandidates method

	public void checkRow(int row, int col) {
		// Clears the arrayLists so that they can be used each time
		// in the loop without any invalid data in the data structures
		usedNumbers.clear();
		possibleNumbers.clear();
		// Iterates through the columns (9)
		for (int i = 0; i < COLUMNS; i++) {
			// Checks if the coordinate is blank
			if (gridArray[row][i] != ' ') {
				char c;
				// Assigns c to data structure
				c = gridArray[row][i];
				String s = Character.toString(c);
				//check is usedNumbers already contains an instance of the
number s
```

```
                        if (usedNumbers.contains(s)) {
                                // Value is already present in arrayList - do not
add
                        } else {
                                // Adds to data structure
                                usedNumbers.add(s);
                        }
                }
        }
    } // ends checkRow method

    public void checkColumn(int row, int col) {
        // Iterates through the number of rows (9)
        for (int i = 0; i < ROWS; i++) {
            // Checks if the cell is blank
            if (gridArray[i][col] != ' ') {
                char c;
                // Assigns c to relevant data structure value
                c = gridArray[i][col];
                String s = Character.toString(c);
                //check is usedNumbers already contains an instance of the
number s
                if (usedNumbers.contains(s)) {
                        // Value is already present in arrayList - do not
add
                } else {
                        // Add to data structure
                        usedNumbers.add(s);
                }
            }
        }
    } // ends checkColumn method

    public void checkSquare(int row, int col) {
        char c; // Variable needed for use in this method
        // Relevant for cells 0,0 0,1 0,2 1,0 1,1 1,2 2,0 2,1 2,2
        if (row <= 2 && col <= 2) {
            for (int i = 0; i < 3; i++) { //row
                for (int j = 0; j < 3; j++) { //col
                    if (gridArray[i][j] != ' ') {
                        c = gridArray[i][j];
                        String s = Character.toString(c);
                        //check is usedNumbers already contains an
instance of the number s
                        if (usedNumbers.contains(s)) {
                                // Value is already present in
arrayList - do not add
                        } else {
                                // Add to data structure
                                usedNumbers.add(s);
                        }
```

```
                                }
                          }
                    }
             }
             // Relevant for cells 0,3 0,4 0,5 1,3 1,4 1,5 2,3 2,4 2,5
             if (row <=2 && col <= 5 && col > 2) {
                    for (int i = 0; i < 3; i++) { //row
                          for (int j = 3; j < 6; j++) { //col
                                if (gridArray[i][j] != ' ') {
                                      c = gridArray[i][j];
                                      String s = Character.toString(c);
                                      //check is usedNumbers already contains an
instance of the number s
                                      if (usedNumbers.contains(s)) {
                                            // Value is already present in
arrayList - do not add
                                      } else {
                                            usedNumbers.add(s);
                                      }
                                }
                          }
                    }
             }
             // Relevant for cells 0,6 0,7 0,8 1,6 1,7 1,8 2,6 2,7 2,8
             if (row <=2 && col <= 8 && col > 5) {
                    for (int i = 0; i < 3; i++) { //row
                          for (int j = 6; j < 9; j++) { //col
                                if (gridArray[i][j] != ' ') {
                                      c = gridArray[i][j];
                                      String s = Character.toString(c);
                                      //check is usedNumbers already contains an
instance of the number s
                                      if (usedNumbers.contains(s)) {
                                            // Value is already present in
arrayList - do not add
                                      } else {
                                            usedNumbers.add(s);
                                      }
                                }
                          }
                    }
             }
             // Relevant for cells 3,0 3,1 3,2 4,0 4,1 4,2 5,0 5,1 5,2
             if (row <= 5 && col <= 2 && row > 2) {
                    for (int i = 3; i < 6; i++) { //row
                          for (int j = 0; j < 3; j++) { //col
                                if (gridArray[i][j] != ' ') {
                                      c = gridArray[i][j];
                                      String s = Character.toString(c);
                                      //check is usedNumbers already contains an
instance of the number s
```

```
                                        if (usedNumbers.contains(s)) {
                                              // Value is already present in
arrayList - do not add
                                        } else {
                                              usedNumbers.add(s);
                                        }
                                  }
                            }
                      }
                }
                // Relevant for cells 3,3 3,4 3,5 4,3 4,4 4,5 5,3 5,4 5,5
                if (row > 2 && row <= 5 && col <= 5 && col > 2) {
                      for (int i = 3; i < 6; i++) { //row
                            for (int j = 3; j < 6; j++) { //col
                                  if (gridArray[i][j] != ' ') {
                                        c = gridArray[i][j];
                                        String s = Character.toString(c);
                                        //check is usedNumbers already contains an
instance of the number s
                                        if (usedNumbers.contains(s)) {
                                              // Value is already present in
arrayList - do not add
                                        } else {
                                              usedNumbers.add(s);
                                        }
                                  }
                            }
                      }
                }
                // Relevant for cells 3,6 3,7 3,8 4,6 4,7 4,8 5,6 5,7 5,8
                if (row <= 5 && col <= 5 && col > 5) {
                      for (int i = 3; i < 6; i++) { //row
                            for (int j = 6; j < 9; j++) { //col
                                  if (gridArray[i][j] != ' ') {
                                        c = gridArray[i][j];
                                        String s = Character.toString(c);
                                        //check is usedNumbers already contains an
instance of the number s
                                        if (usedNumbers.contains(s)) {
                                              // Value is already present in
arrayList - do not add
                                        } else {
                                              usedNumbers.add(s);
                                        }
                                  }
                            }
                      }
                }
                // Relevant for cells 6,0 6,1 6,2 7,0 7,1 7,2 8,0 8,1 8,2
                if (row <= 8 && col <= 2 && row > 5) {
                      for (int i = 6; i < 9; i++) { //row
```

```
                        for (int j = 0; j < 3; j++) { //col
                              if (gridArray[i][j] != ' ') {
                                    c = gridArray[i][j];
                                    String s = Character.toString(c);
                                    //check is usedNumbers already contains an
instance of the number s
                                    if (usedNumbers.contains(s)) {
                                          // Value is already present in
arrayList - do not add
                                    } else {
                                          usedNumbers.add(s);
                                    }
                              }
                        }
                  }
            }
            // Relevant for cells 6,3 6,4 6,5 7,3 7,4 7,5 8,3 8,4 8,5
            if (row > 5 && row <= 8 && col <= 5 && col > 2) {
                  for (int i = 6; i < 9; i++) { //row
                        for (int j = 3; j < 6; j++) { //col
                              if (gridArray[i][j] != ' ') {
                                    c = gridArray[i][j];
                                    String s = Character.toString(c);
                                    //check is usedNumbers already contains an
instance of the number s
                                    if (usedNumbers.contains(s)) {
                                          // Value is already present in
arrayList - do not add
                                    } else {
                                          usedNumbers.add(s);
                                    }
                              }
                        }
                  }
            }
            // Relevant for cells 6,6 6,7 6,8 7,6 7,7 7,8 8,6 8,7 8,8
            if (row <= 8 && col <= 5 && col > 5) {
                  for (int i = 6; i < 9; i++) { //row
                        for (int j = 6; j < 9; j++) { //col
                              if (gridArray[i][j] != ' ') {
                                    c = gridArray[i][j];
                                    String s = Character.toString(c);
                                    //check is usedNumbers already contains an
instance of the number s
                                    if (usedNumbers.contains(s)) {
                                          // Value is already present in
arrayList - do not add
                                    } else {
                                          // Add to data structure
                                          usedNumbers.add(s);
                                    }
```

```
                                }
                        }
                }
        }
    } // ends checkSquare method

    public void solveCell(int row, int col) {
        // Loops through the numbers 1 to 9 (possible Sudoku grid values)
        for (int i = 1; i < 10; i++) {
            String s = Integer.toString(i);
            // Gets rid of " " from the value, so it passes .contains() check
            s = s.replaceAll("^\"|\"$", "");
            if (usedNumbers.contains(s)) {
                // Value is found, do not add
            } else {
                // Value is added to arrayList (could be number to add in)
                possibleNumbers.add(s);
            }
        }
        // Checks if the arrayList.size() is 1, then this means that there is
only
        // 1 value for the cell and it is correct and can output to grid
        if (possibleNumbers.size() == 1) {
            String str = possibleNumbers.get(0);
            char c = str.charAt(0);
            // Parses off first (and only) value in the arrayList
            board.setValue(row, col, c);
            // Adds to setValue for graphical output
            gridArray[row][col] = c;
            // Adds to gridArray for future usage and data storage
        } else {
            // Do nothing
        }
        // Prints the solved grid
        repaint();
    } // ends solveCell method


} // ends SudokuCanvas class
```

```
import java.awt.*;

import javax.swing.*;

import java.awt.event.*;
import java.io.FileNotFoundException;
import java.io.IOException;

@SuppressWarnings("serial")
public class SudokuDisplay extends JFrame implements ActionListener {
	/*
	 * SudokuDisplay.java            Richard Addicott (ria4@aber.ac.uk)
	 * - This class contains the majority of the code that represents
	 * and creates the graphical user interface of the program. It uses
	 * JFrame features and ActionListeners to implement buttons, panels,
	 * graphics and text fields that the user can interact with and
	 * effectively use the program.
	 */

	// Declaration of variables
	public JTextField load = new JTextField();
	public JPanel buttonPanel, gridPanel;
	public JButton loadButton, solveButton, clearButton;
	SudokuCanvas canvas;

	public SudokuDisplay() {
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		setTitle("Display - ria4"); // Title of GUI window

		buttonPanel = new JPanel();
		load = new JTextField();
		loadButton = new JButton("Load");
		clearButton = new JButton("Clear");
		solveButton = new JButton("Solve");
		// Declaration and addition of GUI elements to GUI
		buttonPanel.setLayout(new GridLayout(1,4));
		buttonPanel.add(load);
		buttonPanel.add(loadButton);
		buttonPanel.add(clearButton);
		buttonPanel.add(solveButton);
		// Add actionListeners to GUI elements
		loadButton.addActionListener(this);
		clearButton.addActionListener(this);
		solveButton.addActionListener(this);
		setLayout(new BorderLayout());
		add(buttonPanel, BorderLayout.NORTH);

		canvas = new SudokuCanvas();
		canvas.setBorder(BorderFactory.createLineBorder(Color.black,2));
		add(canvas, BorderLayout.CENTER);
		// Sets the size of the window and makes it visible
```

```
		setSize(380,430);
		setVisible(true);
	} // ends SudokuDisplay constructor


	public void actionPerformed(ActionEvent e) { // User interaction
		// Passes in the event e and chooses appropriate statement to call
		if (e.getSource() == loadButton) {
			try {
				canvas.loadSudoku(load.getText()); // Passes textfield
data into loadSudoku method
			} catch (FileNotFoundException e1) {
				// Standard error handling
				System.out.println("File " + load.getText() + " not found.
Please enter a valid file.");
			} catch (IOException e1) {
				if (load.getText() == null) {
					System.out.println("No text entered. Try again.");
				} else {
					// Valid
				}
			}
		}
		if (e.getSource() == clearButton) {
			canvas.clearSudoku();
		}
		if (e.getSource() == solveButton) {
			canvas.findPossibleCandidates();
		}
	} // ends actionPerformed method


} // ends SudokuDisplay class
```

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



import org.junit.After;
import org.junit.Before;
import org.junit.Test;

import junit.framework.*;


public class JUnitTest {

	SudokuModel model = new SudokuModel(9, 9);

	@Test
	public void testGet() {
		char c = model.returnValue(0, 0);
		assertEquals(c, 0);
	}

	@Test
	public void testSet() {
		char c = 1;
		model.setValue(0, 0, c);
		assertEquals(1 , model.returnValue(0, 0));
	}

}
```