```
//GameBoardManager Class:

package Sudoku;

// This class handles the game board. It includes main.
public class GameManager {

	// This is the main for the Sudoku game.
	public static void main(String[] args) {
		LogInScreen newGame = new LogInScreen();

	}
}


//LogInScreen Class:

package Sudoku;

import java.awt.Color;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.GridBagLayout;
import java.awt.GridLayout;
import java.awt.Image;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JPasswordField;
import javax.swing.JTextField;

// This class handles the log-in proponent.
public class LogInScreen {

	private GameAccount gAccount = null;
	private GameAccountManager account = new GameAccountManager();

	// This is the constructor for the LogInScreen class.
	protected LogInScreen() {
		buildScreen();
	}
```

```
        // This function builds the log in screen.
        private void buildScreen() {
                BufferedImage image = null;
                try {
                        image = ImageIO.read(new File("//Users//
Camille//Documents//COEN275//Assignment1//src//Sudoku//sudoku.png"));
                } catch (IOException ex)
{ System.out.print("IOException Occurred");
                        ex.printStackTrace();
                }

                // Start Game Screen
                JFrame startFrame = new JFrame();
                JPanel startPanelR = new JPanel();
                JPanel startPanelL = new JPanel();
                JPanel startPanel1 = new JPanel();
                JPanel startPanel2 = new JPanel();
                JPanel startPanel3 = new JPanel();
                JPanel startPanel4 =new JPanel();
                JLabel imageLabel = new JLabel();
                imageLabel.setSize(300,300);

                Image dimage =
image.getScaledInstance(imageLabel.getWidth(), imageLabel.getHeight(),
                        Image.SCALE_SMOOTH);

                ImageIcon icon = new ImageIcon(dimage);

                imageLabel.setIcon(icon);

                // Set left start game screen
                startPanelL.setLayout(new GridBagLayout());
                startPanelL.add(imageLabel);

                Font font = new Font("Arial", Font.BOLD,26);
                JLabel label1 = new JLabel("Sign In");
                label1.setFont(font);
                label1.setHorizontalAlignment(JLabel.CENTER);
                label1.setVerticalAlignment(JLabel.CENTER);

                Font font1 = new Font("Arial",Font.PLAIN,14);
                JLabel label2 = new JLabel("Username:");
                label2.setFont(font1);
                JLabel label3 = new JLabel("Password:");
                label3.setFont(font1);

                final int FINAL_WIDTH = 10;
                JTextField text1 = new JTextField(FINAL_WIDTH);
                text1.setText("");
                JPasswordField text2 = new
```

```
JPasswordField(FINAL_WIDTH);

                JButton button1 = new JButton("Login");

                Color orange = Color.getHSBColor(33, 80, 100);
                Color white = Color.GRAY;
                button1.setFont(font1);
                button1.setForeground(white);
                button1.setBackground(orange);
                button1.setOpaque(true);
                button1.setBorderPainted(false);

                startPanelR.setSize(340,660);
                startPanelL.setSize(340,660);

                startPanel1.setSize(85, 100);
                startPanel2.setSize(85, 100);
                startPanel3.setSize(85, 100);
                startPanel4.setSize(85, 100);

                startPanel1.setLayout(new FlowLayout());
                startPanel1.setBounds(0,150,450,50);

                startPanel2.setLayout(new FlowLayout());
                startPanel2.setBounds(0,200,450,50);
                startPanel3.setLayout(new FlowLayout());
                startPanel3.setBounds(0,250,450,250);
                startPanel4.setLayout(new FlowLayout());
                startPanel4.setBounds(0,300,450,50);

                startPanel1.add(label1);
                startPanel2.add(label2);
                startPanel2.add(text1);
                startPanel3.add(label3);
                startPanel3.add(text2);
                startPanel4.add(button1);

                startPanelR.setLayout(null);

                startPanelR.add(startPanel1);
                startPanelR.add(startPanel2);
                startPanelR.add(startPanel3);
                startPanelR.add(startPanel4);

                GridLayout layout = new GridLayout(0,2);
                startFrame.setLayout(layout);
                startFrame.add(startPanelL);
                startFrame.add(startPanelR);

                startFrame.getRootPane().setDefaultButton(button1);
```

```
startFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            startFrame.setSize(900,500);
            startFrame.setLocation(325,600);
            startFrame.setResizable(false);
            startFrame.setVisible(true);

                button1.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent
event) {
                            char[] password =
text2.getText().toCharArray();
                                String userName = text1.getText();
                                gAccount =
account.logIn(userName,password);

                                if(gAccount != null){
                                        startFrame.dispose();
                                        GameBoard game = new
GameBoard(account,gAccount);
                                }
                        }
                });
        }
}
// GameAccountManager Class:

package Sudoku;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Scanner;

import Sudoku.GameAccount;

// This class manages the game accounts.
public class GameAccountManager {

        private ArrayList<GameAccount> list = new
ArrayList<GameAccount>();
        private final String location = "//Users//Camille//Documents//
COEN275//Assignment1//src//Sudoku//PlayerDB.txt";
        private Scanner scanner = new Scanner(System.in);
        private File file;
```

```
        // This is the constructor for the Bank class. It imports a
file, and calls the reader.
        protected GameAccountManager() {
                file = new File(location);
                readDB();
        }

        // This function reads in data from the input file and creates
game accounts.
        private void readDB() {
                try {
                        BufferedReader bufferedReader = new
BufferedReader(new FileReader(file)); //Bufferedreader reads one line.
                        String line = "";

                        while ((line = bufferedReader.readLine()) !=
null) {

                                String[] accountinfo = line.split(",
");
                                GameAccount gameAccount = null;

                                if(accountinfo.length == 6) {     //
string.length() array.length <-- if array is of single variables
                                        gameAccount = new
GameAccount(accountinfo[0],
(accountinfo[1]).toCharArray(),Integer.parseInt(accountinfo[2]),

Integer.parseInt(accountinfo[3]),
Integer.parseInt(accountinfo[4]),Integer.parseInt(accountinfo[5]));
                                }
                                else {
                                        System.out.println
("Error:Incorrect account format.");
                                }

                                if (gameAccount != null) {
                                        list.add(gameAccount);
                                }
                        }
                } catch (FileNotFoundException ex) {
                        System.out.println("FileNotFoundException
Occurred");
                } catch (IOException ex) {
                        System.out.println ("IOException Occurred");
                }
        }

        // This function handles the log-in proponent of the game.
        protected GameAccount logIn(String userName, char[] password)
```

```
{
                GameAccount usersAccount = null;

                for (int i = 0; i < list.size(); i ++) { //
arrayList.size() <-- if array is of objects
                        GameAccount gameAccount = list.get(i);  //<--
arrayList.get(i), array[i]

if(gameAccount.getPlayerName().equals(userName)) {
                                usersAccount = gameAccount;
                                break;
                        }
                }

                if(usersAccount == null) {
                        PopUpManager popUp = new PopUpManager("Error:
Incorrect Username Entered");
                        return usersAccount;
                }

                if (usersAccount.checkPassword(password)) {
                        //gAccount = usersAccount;
                        return usersAccount;

                }

                PopUpManager popUp = new PopUpManager("Error:
Incorrect Password Entered");
                usersAccount = null;
                return usersAccount;
        }

        // This function allows for the game account file to be
rewritten and updated.
        protected void writeDB(GameAccount gAccount) {
                try {
                        FileOutputStream writer = new
FileOutputStream(file,false);

                        for (int i = 0; i < list.size(); i ++) {
                                StringBuilder account = new
StringBuilder();

account.append(list.get(i).getPlayerName());
                                account.append(", ");

account.append(list.get(i).getPassword());
                                account.append(", ");

account.append(list.get(i).getGamesPlayed());
```

```
                                account.append(", ");
account.append(list.get(i).getGamesWon());
                                account.append(", ");
account.append(list.get(i).getGamesLost());
                                account.append(", ");
account.append(list.get(i).getTotalScore());
                                account.append("\n");


writer.write(account.toString().getBytes());
                        }
                        return;
                } catch(FileNotFoundException ex) {
                        System.out.println("FileNotFoundException
occurred.");
                } catch (IOException ex) {
                        System.out.println("IOException occurred.");
                }
        }
}

// GameAccount Class:

package Sudoku;

import java.util.Arrays;

//This class manages the contents of the game account.
public class GameAccount {

        private String playerName;
        private char[] password;
        private int gamesPlayed;
        private int gamesWon;
        private int gamesLost;
        private int totalScore;

        // This is the default constructor for the GameAccount class.
        protected GameAccount() {

        }

        // This is the constructor for the GameAccount class.
        protected GameAccount(String playerName, char[] password, int
gamesPlayed, int gamesWon, int gamesLost, int totalScore) {
                this.playerName = playerName;
                this.password = password;
```

```
                this.gamesPlayed = gamesPlayed;
                this.gamesWon = gamesWon;
                this.gamesLost = gamesLost;
                this.totalScore = totalScore;
        }

        //This function accesses the player name from the game
account.
        protected String getPlayerName() {
                return playerName;
        }

        // This function accesses the password for the game account.
        protected char[] getPassword() {
                return password;
        }

        // This function accesses the number of games played from the
game account.
        protected int getGamesPlayed() {
                return gamesPlayed;
        }

        // This function accesses the number of games won from the
game account.
        protected int getGamesWon() {
                return gamesWon;
        }

        // This function accesses the number of games lost from the
game account.
        protected int getGamesLost() {
                return gamesLost;
        }

        // This function accesses the total score from the game
account.
        protected int getTotalScore() {
                return totalScore;
        }

        // This function checks the login password with the password
for the game account.
        protected boolean checkPassword(char[] pin) {
                if(Arrays.equals(password, pin)) {
                        return true;
                }
                return false;
        }
```

```
	// This function increments the number of games played.
	protected void incrementGamesPlayed() {
		gamesPlayed++;
	}

	// This function increments the number of games won.
	protected void incrementGamesWon() {
		gamesWon++;
	}

	// This function increments the number of games lost.
	protected void incrementGamesLost() {
		gamesLost++;
	}

	// This function recalculates the score.
	protected void setTotalScore(int num) {
		totalScore += num;
	}

	// This function allows the game account to be rewritten in
String form.
	@Override
	public String toString() {
		return playerName + "," + password + "," +
gamesPlayed + "," + gamesWon + "," + gamesLost
				+ "," + totalScore;
	}
}

// GameBoard Class:

package Sudoku;

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


// This class creates the UI for the GameBoard
public class GameBoard {

	private static final Color WHITE = Color.WHITE;
	private static final Color GREEN = Color.GREEN;
	private static final Color ORANGE = Color.ORANGE;
	private GameAccount account = null;
	private Timer timer;
	private int mseconds = 0;
	private int seconds = 0;
	private int minutes = 0;
```

```
        private int score = 0;
        private String mSecondsString = "00";
    private String secondsString = "00";
    private String minutesString = "00";
        private JLabel nameLabel = null;
        private JLabel name = null;
        private JLabel gamesPlayedLabel = null;
        private JLabel gamesPlayed = null;
        private JLabel gamesWonLabel = null;
        private JLabel gamesWon = null;
        private JLabel gamesLostLabel = null;
        private JLabel gamesLost = null;
        private JLabel scoreLabel = null;
        private JLabel scoreVal = null;
        private JLabel timerStr = null;
        private JPanel RPanel = null;
        private SudokuJPanel sudokuPanel = null;
        private GameAccountManager manager = null;
        private Thread thread;

        // This is the constructor for the GameBoard. It calls
createGameBoard().
        protected GameBoard(GameAccountManager
accountManager,GameAccount account) {
                this.account = account;
                manager = accountManager;
                createGameBoard();
        }

        // This function creates the game board.
        private void createGameBoard() {
                // Sudoku Game Screen
                JFrame frame = new JFrame("Sudoku");
                JPanel LPanel = new JPanel();
                LPanel.setBounds(0,0,300,800);
                RPanel = new JPanel();
                RPanel.setBounds(300,0,690,800);
                JPanel LPanelTop = new JPanel();
                LPanelTop.setBounds(0,0,300,300);
                JPanel LPanelCenter = new JPanel();
                LPanelCenter.setBounds(0,300,300,200);
                JPanel LPanelBottom = new JPanel();
                LPanelBottom.setBounds(0,500,300,300);
                sudokuPanel = new SudokuJPanel(account, this);

                // Layouts
                frame.setLayout(null);
                LPanel.setLayout(null);
                LPanelTop.setLayout(null);
                LPanelCenter.setLayout(new
```

```
BoxLayout(LPanelCenter,BoxLayout.Y_AXIS));
                LPanelBottom.setLayout(new
BoxLayout(LPanelBottom,BoxLayout.Y_AXIS));
                RPanel.setLayout(null);

                // Left Top Panel: Scoreboard
                Font font1 = new Font("Arial", Font.BOLD,26);
                JLabel scoreBoard = new JLabel("Scoreboard");
                scoreBoard.setBounds(0,0,300,50);
                scoreBoard.setFont(font1);
                scoreBoard.setHorizontalAlignment(JLabel.CENTER);
            scoreBoard.setVerticalAlignment(JLabel.CENTER);

                nameLabel = new JLabel("Player Name:");
                nameLabel.setBounds(5,50,200,50);
                name = new JLabel(account.getPlayerName());
                name.setBounds(200,50,50,50);
                gamesPlayedLabel = new JLabel("Number of Games
Played:");
                gamesPlayedLabel.setBounds(5,100,200,50);
                gamesPlayed = new
JLabel(Integer.toString(account.getGamesPlayed()));
                gamesPlayed.setBounds(200,100,50,50);
                gamesWonLabel = new JLabel("Number of Games Won:");
                gamesWonLabel.setBounds(5,150,200,50);
                gamesWon = new
JLabel(Integer.toString(account.getGamesWon()));
                gamesWon.setBounds(200,150,50,50);
                gamesLostLabel = new JLabel("Number of Games Lost:");
                gamesLostLabel.setBounds(5,200,200,50);
                gamesLost = new
JLabel(Integer.toString(account.getGamesLost()));
                gamesLost.setBounds(200,200,50,50);
                scoreLabel = new JLabel("Total Score:");
                scoreLabel.setBounds(5,250,200,50);
                scoreVal = new
JLabel(Integer.toString(account.getTotalScore()));
                scoreVal.setBounds(200,250,50,50);

                LPanelTop.add(scoreBoard);
                LPanelTop.add(nameLabel);
                LPanelTop.add(name);
                LPanelTop.add(gamesPlayedLabel);
                LPanelTop.add(gamesPlayed);
                LPanelTop.add(gamesWonLabel);
                LPanelTop.add(gamesWon);
                LPanelTop.add(gamesLostLabel);
                LPanelTop.add(gamesLost);
                LPanelTop.add(scoreLabel);
                LPanelTop.add(scoreVal);
```

```
                // Left Center Panel: Board Color
                JLabel boardColor = new JLabel("Board Color");
                boardColor.setBounds(0,300,300,50);
                boardColor.setHorizontalAlignment(JLabel.CENTER);
            boardColor.setVerticalAlignment(JLabel.CENTER);
                boardColor.setFont(font1);

                String[] options = {"White","Green","Orange"};
                JComboBox<String> colorSelection = new
JComboBox<String>(options);

colorSelection.setMaximumSize( colorSelection.getPreferredSize() );
                ((JLabel)
colorSelection.getRenderer()).setHorizontalAlignment(JLabel.CENTER);
                colorSelection.addActionListener(new ActionListener()
{
                        public void actionPerformed(ActionEvent e) {
                                String color = (String)
colorSelection.getSelectedItem();
                        if(color.equals("White")) {
                                sudokuPanel.changeColor(WHITE);
                        }
                        else if(color.equals("Green")) {
                                sudokuPanel.changeColor(GREEN);
                        }
                        else if(color.equals("Orange")) {
                                sudokuPanel.changeColor(ORANGE);
                        }
                        }
                });

                LPanelCenter.add(boardColor);
                LPanelCenter.add(colorSelection);

                // Left Panel Bottom: Timer
                JLabel timerLabel = new JLabel("Timer");
                timerLabel.setBounds(0,600,300,50);
                timerLabel.setHorizontalAlignment(JLabel.CENTER);
            timerLabel.setVerticalAlignment(JLabel.CENTER);
                timerLabel.setFont(font1);
                timerLabel.setAlignmentX(Component.CENTER_ALIGNMENT);
                timerStr = new JLabel("00:" + "00:" + "00:" + "00");
                Font font2 = new Font("Arial", Font.PLAIN, 20);
                timerStr.setBounds(0,650,300,50);
                timerStr.setHorizontalAlignment(JLabel.CENTER);
            timerStr.setVerticalAlignment(JLabel.CENTER);
            timerStr.setFont(font2);
            timerStr.setAlignmentX(Component.CENTER_ALIGNMENT);
```

```
             JButton playButton = new JButton("Play Now");
                  playButton.setBounds(0,700,300,50);
                  playButton.setHorizontalAlignment(JButton.CENTER);
                  playButton.setVerticalAlignment(JButton.CENTER);
                  playButton.setAlignmentX(Component.CENTER_ALIGNMENT);
                  playButton.addActionListener(new ActionListener() {
             public void actionPerformed(ActionEvent event) {
                  thread = new Thread() {
                     public void run() {
                         try {
                             long elapsedTime = 0;
                             long startTime =
System.currentTimeMillis();


                            while (minutes < 30) {
                                elapsedTime =
System.currentTimeMillis() - startTime;
                                try {
                                  sleep(1);
                                } catch(InterruptedException
exception) {
                                  System.out.println("Interrupted");
                                }
                                mseconds = (int) (elapsedTime % 1000);

                                if (mseconds + 1 == 1000) {
                                    mseconds = 0;
                                    seconds++;
                                }

                                if (seconds == 60) {
                                    mseconds = 0;
                                    seconds = 0;
                                    minutes++;
                                }

                                if (mseconds < 10) {
                                    mSecondsString = "0" + mseconds;
                                } else {
                                    mSecondsString =
String.valueOf(mseconds);
                                }

                                if (seconds < 10) {
                                    secondsString = "0" + seconds;
                                } else {
                                    secondsString =
String.valueOf(seconds);
                                }
```

```
                                if (minutes < 10) {
                                    minutesString = "0" + minutes;
                                }else {
                                    minutesString =
String.valueOf(minutes);
                                }

                                timerStr.setText("00:   " +
minutesString + "  :   " + secondsString + "  :   " + mSecondsString);
                                    }

                            }catch(Exception exception) {
                                exception.printStackTrace();
                            }
                              sudokuPanel.setVisible(false);
                              PopUpManager popUp = new PopUpManager("Time's
up! You lost the game.");
                              account.incrementGamesLost();
                              account.incrementGamesPlayed();
                              manager.writeDB(account);

                        }
                    };
                    thread.start();
                    RPanel.setVisible(true);
                              }
                     });

                     LPanelBottom.add(timerLabel);
                     LPanelBottom.add(timerStr);
                     LPanelBottom.add(playButton);

                     LPanel.add(LPanelTop);
                     LPanel.add(LPanelCenter);
                     LPanel.add(LPanelBottom);

                     // Right Panel: Sudoku Board
                     JLabel sudokuLabel = new JLabel("Sudoku");
                     sudokuLabel.setHorizontalAlignment(JLabel.CENTER);
                     sudokuLabel.setVerticalAlignment(JLabel.CENTER);
                     sudokuLabel.setBounds(0,0,680,30);
                     sudokuLabel.setFont(font1);

                     sudokuPanel.setBounds(20,30,680,680);
                     sudokuPanel.setLayout(new GridLayout(9,9));

                     RPanel.add(sudokuLabel);
                     RPanel.add(sudokuPanel);
```

```
                frame.add(LPanel);
                frame.add(RPanel);
                RPanel.setVisible(false);

                // Provide final details regarding JFrame.
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setSize(975,700);
            frame.setLocation(325,600);
            frame.setResizable(false);
            frame.setVisible(true);

            frame.addWindowListener(new java.awt.event.WindowAdapter()
{
                @Override
                public void windowClosing(java.awt.event.WindowEvent
windowEvent) {
                    manager.writeDB(account);
                }
            });
        }

        // This function returns the minutes counted by the timer.
        protected int getMinutes(){
                return minutes;
        }

        // This function returns the seconds counted by the timer.
        protected int getSeconds(){
                return seconds;
        }

        // This function stops the timer thread.
        @SuppressWarnings("deprecation")
        protected void stopTimer(boolean status) {
                if (status) {
                        thread.stop();
                }
        }

        // This function resets the game board.
        protected void resetGameBoard() {
                stopTimer(true);

                RPanel.setVisible(false);


gamesPlayed.setText(Integer.toString(account.getGamesPlayed()));

gamesWon.setText(Integer.toString(account.getGamesWon()));
```

```
gamesLost.setText(Integer.toString(account.getGamesLost()));

scoreVal.setText(Integer.toString(account.getTotalScore()));
                timerStr.setText("00:" + "00:" + "00:" + "00");
                mSecondsString = "00";
        secondsString = "00";
        minutesString = "00";
        mseconds = 0;
        seconds = 0;
        minutes = 0;

                long elapsedTime = 0;
        long startTime = System.currentTimeMillis();

        sudokuPanel.resetContents();
        }
}

// SudokuJPanel Class:
package Sudoku;

import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.font.FontRenderContext;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import javax.swing.JPanel;

// This class extends the JPanel. It facilitates in all the actions
that occur within the SudokuJPanels.
public class SudokuJPanel extends JPanel {

        private String[][]originalStringContents = new String[9][9];
        private String[][]gameStringContents = new String[9][9];
        private String[][]resetContents = new String[9][9];
        private final String location = "//Users//Camille//Documents//
COEN275//Assignment1//src//Sudoku//contents.txt";
        private File file;
        private int currentlySelectedRow;
        private int currentlySelectedCol;
        private SudokuJPanel object = this;
        private GameAccount account = null;
```

```
        private Color gridColor = Color.white;
        private GameBoard board = null;

        /* This is the constructor for the SudokuJPanel class. It
initializes the input file, and initiates ReadContents.
         * It also holds the mouse listener.
         */
        protected SudokuJPanel(GameAccount account, GameBoard board){
                this.board = board;
                this.account = account;
                file = new File(location);
                readContents();
                addMouseListener(new MouseAdapter() {
                        public void mousePressed(MouseEvent e) {
                                int slotWidth = 70;
                                int slotHeight = 70;

                                currentlySelectedRow = e.getX() /
slotHeight;
                                currentlySelectedCol = e.getY() /
slotWidth;


if(originalStringContents[currentlySelectedRow]
[currentlySelectedCol].equals("0")) {
                                        PopUpManager popUp = new
PopUpManager(object);
                                }
                        }
                });
        }

        /* This function reads the input in from the file and
separates each value into the
         * gameStringContents array.
         */
        private void readContents() {
                final String DELIMITER = ",";
                String[]values=new String[81];

                try {
                        BufferedReader bufferedReader = new
BufferedReader(new FileReader(file)); //Bufferedreader reads one line.
                        String line = "";
                        int j = 0;

                        while((line = bufferedReader.readLine()) !=
null) {

                                values = line.split(DELIMITER);
```

```
                    for (int i = 0; i < values.length; i++) {
                         gameStringContents[i][j]= values[i];
                         originalStringContents[i][j]=values[i];
                         resetContents[i][j] = values[i];
                    }
                    j++;
                         }
                         bufferedReader.close();
                } catch (FileNotFoundException ex) {
                         System.out.println("FileNotFoundException
Occurred");
                } catch (IOException ex) {
                         System.out.println ("IOException Occurred");
                }
        }

        // This function updates the data within the SudokoPanel and
the gameStringContents array.
        protected void updateData(String num) {
                boolean boardStatus = false;

                gameStringContents[currentlySelectedRow]
[currentlySelectedCol] = num;

                this.revalidate();
                this.repaint();

                for (int i = 0; i < 9; i++) {
                         for (int j = 0; j < 9; j++){

if(Integer.parseInt(gameStringContents[i][j]) > 0 &&
Integer.parseInt(gameStringContents[i][j]) < 10) {
                                          continue;
                                 }
                                 else {
                                          boardStatus = false;
                                          return;
                                 }
                         }
                }
                boardStatus = true;

                if (boardStatus) {
                         board.stopTimer(true);
                         int minutes = board.getMinutes();
                         int seconds = board.getSeconds();
                         SolutionChecker solution = new
SolutionChecker(gameStringContents, account, board,minutes,seconds);
                }
```

```
        }

        // This function resets the gameStringContents to its original
contents so that the board can be reset.
        protected void resetContents() {
                gameStringContents = resetContents;
        }

        // This function allows for the Sudoku board color to be
changed.
        protected void changeColor(Color color) {
                this.revalidate();
                gridColor = color;
                this.repaint();
        }

        // This function handles the graphics within the
SudokuJPanels, including the borders and painting of values.
        protected void paintComponent(Graphics g) {
                super.paintComponent(g);
                Graphics2D g2d = (Graphics2D) g;

                // Set the color of the SudokoJPanel and grid.
                g2d.setColor(new Color(255,255,255));

                // Set the stroke width for the vertical grid.
                for(int x = 0;x <= 630;x+=70) {
                        if((x % 210) == 0) {
                                g2d.setColor(gridColor);
                                g2d.setStroke(new BasicStroke(10));
                                g2d.drawLine(x, 0, x, 630);
                        }
                        else {
                                g2d.setColor(gridColor);
                                g2d.setStroke(new BasicStroke(2));
                                g2d.drawLine(x, 0, x, 630);
                        }
                }

                // Set the stroke width for the horizontal grid.
                for(int y = 0;y <= 630;y+=70) {
                        if((y % 210) == 0) {
                                g2d.setColor(gridColor);
                                g2d.setStroke(new BasicStroke(10));
                                g2d.drawLine(0, y, 630, y);
                                }
                        else {
                                g2d.setColor(gridColor);
                                g2d.setStroke(new BasicStroke(2));
                                g2d.drawLine(0, y, 630, y);
```

```
                        }
                }

                // Paint the values into the SudokoJPanel from
originalStringContents.
                Font f = new Font("Arial", Font.PLAIN, 24);
                g2d.setFont(f);
                FontRenderContext fContext =
g2d.getFontRenderContext();
                for (int i = 0; i < 9; i++) {
                        for (int j = 0; j < 9; j++) {

if(Integer.parseInt(originalStringContents[i][j]) > 0 &&
Integer.parseInt(originalStringContents[i][j]) < 10) {
                                                        g2d.setColor(new
Color(96,96,96));

g2d.drawString(originalStringContents[i][j],((i*70)+25),((j*70)+40));
                                }
                        }
                }

                // Paint the values into the SudokoJPanel from
gameStringContents.
                for (int i = 0; i < 9; i++) {
                        for (int j = 0; j < 9; j++) {

if(Integer.parseInt(gameStringContents[i][j]) !=
Integer.parseInt(originalStringContents[i][j])){

if(Integer.parseInt(gameStringContents[i][j]) > 0 &&
Integer.parseInt(gameStringContents[i][j]) < 10) {

g2d.setColor(new Color(255,0,127));

g2d.drawString(gameStringContents[i][j],((i*70)+25),((j*70)+40));
                                                }
                                }
                        }
                }
        }
}

// PopUpManager Class:

package Sudoku;

import java.awt.BorderLayout;
import java.awt.Component;
import java.awt.FlowLayout;
```

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.InputMismatchException;
import javax.swing.*;

// This class holds all the pop-ups for the game.
public class PopUpManager {
	private GameBoard board = null;
	private GameAccount account = null;

	// This is the constructor for the PopUpManager. It calls
createPopUpOne().
	protected PopUpManager(SudokuJPanel panel)
	{
		createPopUpOne(panel);
	}

	/* This is the constructor for the PopUpManager that takes in
a boolean input. It then
	 * calls either createPopUpTwo() or createPopUpThree()
depending on the input status.
	 */
	protected PopUpManager(boolean status,GameAccount account,
GameBoard board)
	{
		this.board = board;
		this.account = account;
		if(status == true) {
			createPopUpTwo(account,board);
		}
		else {
			createPopUpThree(account,board);
		}
	}

	// This is the constructor for the PopUpManager that takes in
a String message and calls createPopUpFour().
	protected PopUpManager(String message)
	{
		createPopUpFour(message);
	}

	/* This function handles pop-up one. It allows for the user to
input a number into the
	 * Sudoku board.
	 */
	private void createPopUpOne(SudokuJPanel panel){
		// Create JFrame, JPanels, JButtons, JLabels, and
JTextField.
		JFrame frame = new JFrame("Input");
```

```
                JPanel panel1 = new JPanel();
                JPanel panel2= new JPanel();
                JPanel panel3 = new JPanel();
                JPanel panel4 = new JPanel();
                JButton button1 = new JButton("OK");
                JButton button2 = new JButton("Cancel");
                JLabel label1 = new JLabel("Enter a number");
                JTextField textField = new JTextField(20);

                // Set layout for the panels as FlowLayout.
                panel1.setLayout(new FlowLayout());
                panel2.setLayout(new FlowLayout());
                panel3.setLayout(new FlowLayout());
                panel4.setLayout(new FlowLayout());

            // Add JLabel, JTextField, and JButtons to JPanels.
            panel2.add(label1);
            panel3.add(textField);
            panel4.add(button1);
            panel4.add(button2);

            // Add an action listener to button1.
            button1.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent
event) {
                                try {
                                        String num =
textField.getText();
                                        if (Integer.parseInt(num) >
0 && Integer.parseInt(num) < 10) {
                                                frame.dispose();

panel.updateData(num);
                                        }
                                        else {

System.out.println("Invalid entry");
                                        }
                                } catch(InputMismatchException e) {

System.out.println("InputMismatchException Occurred");
                                } catch(NumberFormatException e) {

System.out.println("NumberFormatException Occurred");
                                }
                        }
                });

            // Add an action listener to button2.
            button2.addActionListener(new ActionListener() {
```

```
                        public void actionPerformed(ActionEvent
event) {
                                frame.dispose();
                        }
                });

            // Add JPanels to JFrame using BorderLayout.
                frame.add(panel1, BorderLayout.EAST);
                frame.add(panel2, BorderLayout.NORTH);
                frame.add(panel3, BorderLayout.CENTER);
                frame.add(panel4, BorderLayout.SOUTH);

                // Provide final details regarding JFrame.
                frame.getRootPane().setDefaultButton(button1);
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setSize(400, 150);
            frame.setResizable(false);
            frame.setLocation(450,250);
            frame.setVisible(true);
        }

        // This function handles pop-up two. It displays a message if
the Sudoku solution is correct.
        private void createPopUpTwo(GameAccount account, GameBoard
board){
                // Create JFrame, JPanels, JButtons, JLabels, and
JTextField.
                JFrame frame = new JFrame("Message");
                JPanel panel1 = new JPanel();
                JPanel panel2= new JPanel();
                JPanel panel3 = new JPanel();
                JButton button1 = new JButton("OK");
                JButton button2 = new JButton("Play Again");
                JLabel label1 = new JLabel("You won the Sudoku!");

                int minutes = board.getMinutes();
                int minsLeft = 29-minutes;
                int seconds = board.getSeconds();
                int secsLeft = 60-seconds;

                int newScore = (minsLeft * 60) + secsLeft + 500;

                JLabel label2 = new JLabel("Your Score: " +
newScore);

                // Set layout for the panels as FlowLayout.
                panel1.setLayout(new FlowLayout());
                panel2.setLayout(new
BoxLayout(panel2,BoxLayout.Y_AXIS));
                panel3.setLayout(new FlowLayout());
```

```
                label1.setAlignmentX(Component.CENTER_ALIGNMENT);
                label2.setAlignmentX(Component.CENTER_ALIGNMENT);

            // Add JLabel, JTextFie2d, and JButtons to JPanels.
            panel2.add(label1);
            panel2.add(label2);
            panel3.add(button1);
            panel3.add(button2);

            // Add an action listener to button1.
            button1.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent
event) {
                                frame.dispose();
                        }
                });

            // Add an action listener to button2.
            button2.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent
event) {
                                frame.dispose();
                                board.resetGameBoard();
                        }
                });

            // Add JPanels to JFrame using BorderLayout.
                frame.add(panel1, BorderLayout.EAST);
                frame.add(panel2, BorderLayout.CENTER);
                frame.add(panel3, BorderLayout.SOUTH);

                // Provide final details regarding JFrame.
                frame.getRootPane().setDefaultButton(button1);
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setSize(400, 100);
            frame.setResizable(false);
            frame.setLocation(450,250);
            frame.setVisible(true);
        }

        // This function handles pop-up three. It displays a message
if the Sudoku solution is incorrect.
        private void createPopUpThree(GameAccount account, GameBoard
board){
                // Create JFrame, JPanels, JButtons, JLabels, and
JTextField.
                JFrame frame = new JFrame("Message");
                JPanel panel1 = new JPanel();
                JPanel panel2= new JPanel();
```

```
                JPanel panel3 = new JPanel();
                JButton button1 = new JButton("Go Back to Game
Board");
                JButton button2 = new JButton("Start Over");
                JLabel label1 = new JLabel("Incorrect Solution");

                // Set layout for the panels as FlowLayout.
                panel1.setLayout(new FlowLayout());
                panel2.setLayout(new FlowLayout());
                panel3.setLayout(new FlowLayout());

            // Add JLabel, JTextField, and JButtons to JPanels.
            panel2.add(label1);
            panel3.add(button1);
            panel3.add(button2);

            // Add an action listener to button1.
            button1.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent
event) {
                                frame.dispose();
                        }
                });

          // Add an action listener to button2.
            button2.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent
event) {
                                frame.dispose();
                                board.resetGameBoard();
                        }
                });

            // Add JPanels to JFrame using BorderLayout.
                frame.add(panel1, BorderLayout.EAST);
                frame.add(panel2, BorderLayout.CENTER);
                frame.add(panel3, BorderLayout.SOUTH);

                // Provide final details regarding JFrame.
                frame.getRootPane().setDefaultButton(button1);
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setSize(400, 100);
            frame.setResizable(false);
            frame.setLocation(450,250);
            frame.setVisible(true);
        }

        // This function handles pop-up four. It displays any message
passed to it.
        private void createPopUpFour(String message){
```

```
                // Create JFrame, JPanels, JButtons, JLabels, and
JTextField.
                JFrame frame = new JFrame("Message");
                JPanel panel1 = new JPanel();
                JPanel panel2= new JPanel();
                JPanel panel3 = new JPanel();
                JButton button1 = new JButton("OK");
                JLabel label1 = new JLabel(message);

                // Set layout for the panels as FlowLayout.
                panel1.setLayout(new FlowLayout());
                panel2.setLayout(new FlowLayout());
                panel3.setLayout(new FlowLayout());

            // Add JLabel, JTextField, and JButtons to JPanels.
            panel2.add(label1);
            panel3.add(button1);

            // Add an action listener to button1.
            button1.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent
event) {
                                frame.dispose();
                        }
                });

            // Add JPanels to JFrame using BorderLayout.
                frame.add(panel1, BorderLayout.EAST);
                frame.add(panel2, BorderLayout.CENTER);
                frame.add(panel3, BorderLayout.SOUTH);

                // Provide final details regarding JFrame.
                frame.getRootPane().setDefaultButton(button1);
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setSize(400, 100);
            frame.setResizable(false);
            frame.setLocation(450,250);
            frame.setVisible(true);
        }
}

// SolutionChecker Class:

package Sudoku;

import Sudoku.GameAccount;
import Sudoku.GameBoard;
import Sudoku.PopUpManager;

//This class checks whether the sudoku board is correct.
```

```
public class SolutionChecker {

	private GameBoard board = null;
	private GameAccount account = null;
	private int minutes = 0;
	private int seconds = 0;

	// This is the constructor for the SolutionChecker class. It
galls the solution function.
	protected SolutionChecker(String[][] solutionGrid, GameAccount
account, GameBoard board, int minutes, int seconds) {
		this.board = board;
		this.account = account;
		this.minutes = minutes;
		this.seconds = seconds;

		solution(solutionGrid, board);
	}

	/* This function takes in the solution grid and iterates
through the rows, columns, and
	 * 3x3 grids to confirm that there are no repeats of integers.
	 */
	private void solution(String[][] solutionGrid, GameBoard
board){
		boolean status = true;
		account.incrementGamesPlayed();

		// Check rows for duplicates
		for(int i = 0; i < 9; i++) {
			for(int j = 0; j < 8; j++) {
				for(int k = j + 1; k < 9; k++) {

if(Integer.parseInt(solutionGrid[i]
[j])==Integer.parseInt(solutionGrid[i][k])) {
status = false;
}
}
}
		}

		// Check columns for duplicates.
		for(int j = 0; j < 9; j++) {
			for(int i = 0; i < 8; i++) {
				for(int k = i + 1; k < 9; k++) {

if(Integer.parseInt(solutionGrid[i]
[j])==Integer.parseInt(solutionGrid[k][j])) {
status = false;
}
```

```
                                }
                        }
                }

                // Check 3x3 squares for duplicates.
                for(int i = 0; i < 9; i += 3) {
                        for(int j = 0; j < 9; j += 3) {
                                for(int k = 0; k < 8; k++) {
                                        for(int l = k + 1; l < 9; l+
+) {

if(Integer.parseInt(solutionGrid[i + k%3][j + k/
3])==Integer.parseInt(solutionGrid[i + l%3][j + l/3])) {
                                                                status =
false;
                                                        }
                                                }
                                        }
                                }
                        }
                }

                if (status) {
                        account.incrementGamesWon();
                        int minsLeft = 29 - minutes;
                        int secondsLeft = 60 - seconds;
                        account.setTotalScore(500 + (minsLeft * 60) +
secondsLeft);
                }
                else {
                        account.incrementGamesLost();
                        account.setTotalScore(0);
                }

                PopUpManager popUp = new
PopUpManager(status,account,board);
        }
}
```