```
1    /**
2    IB Computer Science Higher Level Program Dossier
3    EIS-J Ecology Club Recycling Activity Monitoring System
4    Author: Harris Rasheed
5    IDE: JCreator 4.50.010 - Xinox Software, Windows 7 Operating System
6    Date: Saturday 13th March 2010
7    Candidate Number: 000666-037, May 2010 Session
8    Emirates International School - Jumeirah
9    **/
10
11   import java.awt.*;
12   import java.awt.event.*;
13   import javax.swing.*;
14   import java.io.*;
15   import java.util.*;
16   import java.text.*;
17   //-------------------------------------------------------------------------------------------------------------------------
18   //Main Program; This is the password screen that opens up on start-up of the application.
19   public class Program extends JFrame implements ActionListener
20   {
21   //-----------------------------------------------------------------
22   //Main method of program initiates the password screen
23       public static void main (String [] arg)
24       {
25           Program EP = new Program("");
26           EP.FR.setVisible(true);
27       }
28   //-----------------------------------------------------------------
29       JFrame FR = new JFrame("Ecology Club - Recycling Activity Monitoring System");
30       Container Obj1 = getContentPane();
31       GridBagLayout GBL = new GridBagLayout();
32       GridBagConstraints GBC = new GridBagConstraints();
33
34       JMenuBar MB = new JMenuBar();
35       JMenu file = new JMenu("File");
36       JMenuItem CloseFile = new JMenuItem("Close");
37
38       Font f = new Font("Comic Sans MS", Font.BOLD, 22);
39       Color c = new Color(6,69,1);
40
41       ImageIcon logo = new ImageIcon("ClubLogo.jpg");
42       JLabel lbl1 = new JLabel("",logo,SwingConstants.TRAILING);
43       JLabel lbl2 = new JLabel("The EIS-J Ecology Club");
44       JLabel lblPass = new JLabel("Password:");
45       JButton btnSubmit = new JButton("Submit");
46       JButton btnForgotPass = new JButton("Forgot Password?");
47
48       JPasswordField txtPass = new JPasswordField(20);
49       int PassCounter = 0;        //Counter for failed login attempts
50       int sQtionCounter = 0;      //Counter for failed secret question attempts
51
52   //-----------------------------------------------------------------
53   //Constructor for the Password Screen that places components on the Frame
54       public Program(String str)
55       {
56           super(str);
57
58           getContentPane().setLayout(GBL);
59
```

```
60              FR.setJMenuBar(MB);
61              FR.add(getContentPane());
62              MB.add(file);
63              file.add(CloseFile);
64
65              GBC.fill = GridBagConstraints.BOTH;
66              GBC.anchor = GridBagConstraints.CENTER;
67              GBC.gridwidth = 3;
68              GBC.gridheight = 1;
69              GBC.gridy = 0;
70              GBC.gridx = 0;
71              GBC.insets = new Insets(10,10,10,10);
72              lbl2.setFont(f);
73              lbl2.setForeground(Color.white);
74              lbl2.setHorizontalAlignment(JLabel.CENTER);
75              GBL.setConstraints(lbl2,GBC);
76              getContentPane().add(lbl2);
77
78              GBC.gridy = 1;
79              GBC.gridheight = 3;
80              GBL.setConstraints(lbl1,GBC);
81              getContentPane().add(lbl1);
82
83              GBC.gridx = 2;
84              GBC.gridy = 6;
85              GBC.gridwidth = 1;
86              GBC.gridheight = 2;
87              GBL.setConstraints(btnSubmit,GBC);
88              getContentPane().add(btnSubmit);
89
90              GBC.gridx = 4;
91              GBC.gridy = 5;
92              GBC.gridwidth = 1;
93              GBL.setConstraints(btnForgotPass,GBC);
94              getContentPane().add(btnForgotPass);
95
96              txtPass.setEchoChar('*');
97              lblPass.setLabelFor(txtPass);
98
99              GBC.gridx = 2;
100             GBC.gridwidth = 3;
101             GBC.gridheight = 1;
102             GBC.gridy = 4;
103             GBL.setConstraints(lblPass,GBC);
104             lblPass.setForeground(Color.white);
105             getContentPane().add(lblPass);
106
107             GBC.gridy = 5;
108             GBC.gridwidth = 1;
109             GBL.setConstraints(txtPass,GBC);
110             getContentPane().add(txtPass);
111
112             getContentPane().setBackground(c);
113             FR.setExtendedState(Frame.MAXIMIZED_BOTH);
114
115             btnSubmit.addActionListener(this);
116             btnForgotPass.addActionListener(this);
117             CloseFile.addActionListener(this);
118             validate();
```

```
119
120         addWindowListener(new WindowAdapter()
121         {
122             public void windowClosing(WindowEvent we)
123             {
124                 setVisible(false);
125                 System.exit(0);
126             }
127         });
128     }
129 //---------------------------------------------------------------
130 //This method reads the random access file thats store the system
131 //security details. The password stored in the file is returned to
132 //the method and this is used in the actionPerformed method.
133     private String currentPassword()
134     {
135         File PasswordStore = new File("SystemSecurity.dat");    //Creates object of SystemSecurity.dat file that stor
    the program's password
136         String pass = "";               //Initialises variable to store the password in the RAF
137
138         if(!PasswordStore.exists())     //Checks if the system security file exists in the current directory
139         {
140             Toolkit.getDefaultToolkit().beep();             //Error beep sound
141             JOptionPane.showMessageDialog(this, "Error. The file that stores the password does not exist.","Error
    Message", JOptionPane.ERROR_MESSAGE);
142             System.exit(1);     //Shuts down the program as a malfunction because login is not possible without the
    password file
143             return null;
144         }
145         else
146         {
147             try
148             {
149                 RandomAccessFile RAF = new RandomAccessFile(PasswordStore, "r");    //Creates object to read RAF
150                 RAF.seek(0);                                //Sets pointer to start of file
151                 for(int i = 0; i < 20; i++)
152                 {
153                     byte letter = RAF.readByte();       //Reads each character from the first 20 characters of the fi
154                     pass = pass + (char) letter;        //Adds each character from the password field of the file
155                 }
156                 pass = pass.trim();                     //Removes spaces from the string
157                 RAF.close();                            //Closes Random Access File
158             }
159             catch(Exception e)
160             {
161                 Toolkit.getDefaultToolkit().beep();     //Error beep sound
162                 JOptionPane.showMessageDialog(this, "An error has occured. Please contact Harris Rasheed to deal with
    this issue\nError Code: " + e,"Error Message", JOptionPane.ERROR_MESSAGE);    //Output any errors caught
163             }
164         }
165         return pass;            //Returns the password stored in the file
166     }
167 //---------------------------------------------------------------
168 //This method reads the random access file thats store the system
169 //security details. The secret question's answer stored in the file
170 //is returned to the method and this is used in the actionPerformed method.
171     private String[] currentSQtionAnswer()
172     {
173             File PasswordStore = new File("SystemSecurity.dat");    //Creates an object of SystemSecurity.dat
```

```
174
175             try
176             {
177                 String[] sQtion = new String[2];
178                 RandomAccessFile RAF = new RandomAccessFile(PasswordStore, "r");        //Creates object to read Rand
    Access File
179
180                 RAF.seek(20);           //Goes to the 20th position of the file where the secret question is stored
181                 sQtion[0] = "";
182
183                 for(int i = 0; i < 60; i++)     //Loop reads secret question from RAF
184                 {
185                     byte c = RAF.readByte();
186                     sQtion[0] += (char) c;
187                 }
188
189                 RAF.seek(80);           //Goes to the 80th position where the secret answer is stored
190                 sQtion[1] = "";
191
192                 for(int i = 0; i < 20; i++)     //Loop reads answer to secret question from RAF
193                 {
194                     byte c = RAF.readByte();
195                     sQtion[1] += (char) c;
196                 }
197
198                 sQtion[0] = sQtion[0].trim();   //Remove whitespace after and before the secret question
199                 sQtion[1] = sQtion[1].trim();   //Remove whitespace after and before the secret answer
200                 RAF.close();
201                 return sQtion;          //Returns RAF secret question and answer
202             }
203             catch(Exception e)
204             {
205                 Toolkit.getDefaultToolkit().beep();
206                 JOptionPane.showMessageDialog(this, "An unexpected error occured. Please contact Harris Rasheed for m
    information.\nError Code: " + e,"Error Message", JOptionPane.ERROR_MESSAGE);   //Output any errors caught
207             }
208         return null;
209     }
210 //-----------------------------------------------------------------
211 //This method is used to execute the appropriate method when the user performs an action event
212     public void actionPerformed(ActionEvent ae)
213     {
214         if(ae.getSource()==btnSubmit)
215         {
216             if(txtPass.getText().equals(""))
217             {
218                 JOptionPane.showMessageDialog(this, "Error! Please input a password.","Error Message",
    JOptionPane.ERROR_MESSAGE);
219             }
220             else if((txtPass.getText()).trim().equals(currentPassword()))   //Condition that input password is correc
221             {
222                 FR.setVisible(false);               //Hides current window
223                 menuPage MP = new menuPage("");     //Creates object of Menu class and executes constructor
224                 MP.FR.setVisible(true);             //Makes Menu Page's Frame visible
225             }
226             else
227             {
228                 PassCounter++;                  //Adds one to the counter for the failed attempt
229
```

```
230                 if(PassCounter==5)              //Checks if the counter has reached five failed attempts
231                 {
232                     JOptionPane.showMessageDialog(this, "You have exceeded the number of login attempts available.\nT
    program will now shut down.","Error Message", JOptionPane.ERROR_MESSAGE);
233                     Toolkit.getDefaultToolkit().beep();
234                     System.exit(0);             //Exits program because of excess login attempts
235                 }
236                 else
237                 {
238                     JOptionPane.showMessageDialog(this, "The password you have input is incorrect. Please retype the
    correct password.","Error Message", JOptionPane.ERROR_MESSAGE);
239                     Toolkit.getDefaultToolkit().beep();
240                     txtPass.setText("");        //Clears the password field
241                     txtPass.requestFocus();     //Makes the cursor focus on the password field so that the password c
    be retyped
242                 }
243             }
244         }
245
246         else if(ae.getSource()==btnForgotPass)
247         {
248             String gAnswer = JOptionPane.showInputDialog(null, currentSQtionAnswer()[0],"Forgot Password", 3);  //Use
    given secret answer
249
250             if(gAnswer.equals(null)||gAnswer.equals(""))            //Tests if the given answer is blank or null
251             {
252                 JOptionPane.showMessageDialog(this, "Error. Please input an answer!","Error Message",
    JOptionPane.ERROR_MESSAGE);
253                 return;
254             }
255
256             if(gAnswer.equalsIgnoreCase(currentSQtionAnswer()[1]))      //Test if the input answer is correct
257             {
258                 FR.setVisible(false);
259                 menuPage MP = new menuPage("");
260                 MP.FR.setVisible(true);
261             }
262             else
263             {
264                 Toolkit.getDefaultToolkit().beep();
265                 sQtionCounter++;                //Appends one to the secret question counter because of failed login
266
267                 if(sQtionCounter == 3)          //Tests when the 3 failed secret question attempts have been made
268                 {
269                     JOptionPane.showMessageDialog(this, "You have exceeded the number of secret question attempts
    available.\nThis program will now shut down.","Error Message", JOptionPane.ERROR_MESSAGE);
270                     System.exit(0);             //Exits program because of excess login attempts
271                 }
272                 else
273                 {
274                     JOptionPane.showMessageDialog(null, "Error! The answer you have input is incorrect.", "Error",
    JOptionPane.ERROR_MESSAGE);
275                 }
276             }
277         }
278
279         else if(ae.getSource()==CloseFile)
280         {
281             System.exit(0);     //Exits program because the close button from the menubar is pressed
```

```
282             }
283         }
284     }
285     //-------------------------------------------------------------------------------------------------------------------
286     //Menu Screen; This is the main menu of the program where all features of the program can be accessed through.
287     class menuPage extends JFrame implements ActionListener
288     {
289         JFrame FR = new JFrame("Recycling Activity Monitoring System - Main Menu");
290         Container Obj1 = getContentPane();
291         GridBagLayout GBL = new GridBagLayout();
292         GridBagConstraints GBC = new GridBagConstraints();
293 
294         JMenuBar MB = new JMenuBar();
295         JMenu file = new JMenu("File");
296         JMenu view = new JMenu("View");
297         JMenu help = new JMenu("Help");
298         JMenu bgColour = new JMenu("Background Colour");
299         JMenuItem logOut = new JMenuItem("Log out");
300         JMenuItem Exit = new JMenuItem("Exit");
301         JMenuItem About = new JMenuItem("About");
302         ButtonGroup rbg = new ButtonGroup();
303         JRadioButtonMenuItem bgYellow = new JRadioButtonMenuItem("Yellow", false);
304         JRadioButtonMenuItem bgOrange = new JRadioButtonMenuItem("Orange", false);
305         JRadioButtonMenuItem bgRed = new JRadioButtonMenuItem("Red", false);
306         JRadioButtonMenuItem bgPink = new JRadioButtonMenuItem("Pink", false);
307         JRadioButtonMenuItem bgLightGreen = new JRadioButtonMenuItem("Light Green", false);
308         JRadioButtonMenuItem bgDarkGreen = new JRadioButtonMenuItem("Dark Green", true);
309         JRadioButtonMenuItem bgBlue = new JRadioButtonMenuItem("Dark Blue", false);
310         JRadioButtonMenuItem bgCyan = new JRadioButtonMenuItem("Cyan", false);
311         JRadioButtonMenuItem bgMagenta = new JRadioButtonMenuItem("Magenta", false);
312         JRadioButtonMenuItem bgWhite = new JRadioButtonMenuItem("White", false);
313         JRadioButtonMenuItem bgLightGray = new JRadioButtonMenuItem("Light Gray", false);
314         JRadioButtonMenuItem bgDarkGray = new JRadioButtonMenuItem("Dark Gray", false);
315         JRadioButtonMenuItem bgBlack = new JRadioButtonMenuItem("Black", false);
316 
317         JButton btnThursRecycQuota = new JButton("Thursday Recycling Quota");       //Create Menu Screen Buttons
318         JButton btnRecyAttendanceReport = new JButton("Recycler Attendance Report");
319         JButton btnRecyActivityReport = new JButton("Recycling Activity Report");
320         JButton btnRecyRegist = new JButton("Recycler Registration");
321         JButton btnRecyMonCrit = new JButton("Recycler of the Month Candidate Criteria");
322         JButton btnTeacherClass = new JButton("Teachers & Classrooms Plan");
323         JButton btnFormClass = new JButton("Form Class Locations");
324         JButton btnSecuritySett = new JButton("Security Settings");
325         JButton btnLogOut = new JButton("Log out");
326         JLabel lblTitle = new JLabel("Menu");
327 
328         Color c = new Color(6,69,1);
329         Font f = new Font("Comic Sans MS", Font.BOLD, 28);
330 
331     //-----------------------------------------------------------------
332     //Constructor for the Menu Screen that places components on the Frame
333         public menuPage(String str)
334         {
335             super(str);
336             getContentPane().setLayout(GBL);
337 
338             FR.setJMenuBar(MB);
339             FR.add(getContentPane());
340 
```

```
341             GBC.fill = GridBagConstraints.BOTH;
342             GBC.anchor = GridBagConstraints.CENTER;
343             GBC.gridwidth = 2;
344             GBC.gridheight = 2;
345             GBC.gridy = 1;
346             GBC.gridx = 1;
347             GBC.ipady = 20;
348             GBC.insets = new Insets(10,10,10,10);
349
350             lblTitle.setFont(f);
351             lblTitle.setHorizontalAlignment(JLabel.CENTER);
352             lblTitle.setForeground(Color.white);
353             GBL.setConstraints(lblTitle,GBC);
354             getContentPane().add(lblTitle);
355
356             GBC.gridy = 3;
357             GBL.setConstraints(btnThursRecycQuota,GBC);
358             getContentPane().add(btnThursRecycQuota);
359
360             GBC.gridy = 5;
361             GBL.setConstraints(btnRecyAttendanceReport,GBC);
362             getContentPane().add(btnRecyAttendanceReport);
363
364             GBC.gridy = 7;
365             GBL.setConstraints(btnRecyActivityReport,GBC);
366             getContentPane().add(btnRecyActivityReport);
367
368             GBC.gridy = 9;
369             GBL.setConstraints(btnRecyRegist,GBC);
370             getContentPane().add(btnRecyRegist);
371
372             GBC.gridy = 11;
373             GBL.setConstraints(btnRecyMonCrit,GBC);
374             getContentPane().add(btnRecyMonCrit);
375
376             GBC.gridy = 13;
377             GBL.setConstraints(btnTeacherClass,GBC);
378             getContentPane().add(btnTeacherClass);
379
380             GBC.gridy = 15;
381             GBL.setConstraints(btnFormClass,GBC);
382             getContentPane().add(btnFormClass);
383
384             GBC.gridy = 17;
385             GBC.gridheight = GridBagConstraints.RELATIVE;
386             GBL.setConstraints(btnSecuritySett,GBC);
387             getContentPane().add(btnSecuritySett);
388
389             GBC.gridy = 25;
390             GBC.gridx = 2;
391             GBC.weighty = 1;
392             GBC.gridheight = GridBagConstraints.REMAINDER;
393             GBL.setConstraints(btnLogOut,GBC);
394             getContentPane().add(btnLogOut);
395
396             MB.add(file);
397             MB.add(view);
398             MB.add(help);
399             file.add(logOut);
```

```
400             file.add(Exit);
401             view.add(bgColour);
402             help.add(About);
403 
404             view.add(bgColour);
405             bgColour.add(bgYellow);     //Add Background Colour Radio Buttons
406             rbg.add(bgYellow);          //Add Radio Buttons to Button Group
407             bgColour.add(bgOrange);
408             rbg.add(bgOrange);
409             bgColour.add(bgRed);
410             rbg.add(bgRed);
411             bgColour.add(bgPink);
412             rbg.add(bgPink);
413             bgColour.add(bgLightGreen);
414             rbg.add(bgLightGreen);
415             bgColour.add(bgDarkGreen);
416             rbg.add(bgDarkGreen);
417             bgColour.add(bgCyan);
418             rbg.add(bgCyan);
419             bgColour.add(bgBlue);
420             rbg.add(bgBlue);
421             bgColour.add(bgMagenta);
422             rbg.add(bgMagenta);
423             bgColour.add(bgWhite);
424             rbg.add(bgWhite);
425             bgColour.add(bgLightGray);
426             rbg.add(bgLightGray);
427             bgColour.add(bgDarkGray);
428             rbg.add(bgDarkGray);
429             bgColour.add(bgBlack);
430             rbg.add(bgBlack);
431 
432             file.setMnemonic('f');      //Add Keyboard Shortcut Keys
433             view.setMnemonic('v');
434             Exit.setMnemonic('x');
435             logOut.setMnemonic('o');
436             bgColour.setMnemonic('b');
437             bgYellow.setMnemonic('y');
438             bgOrange.setMnemonic('o');
439             bgRed.setMnemonic('r');
440             bgPink.setMnemonic('p');
441             bgLightGreen.setMnemonic('l');
442             bgDarkGreen.setMnemonic('g');
443             bgBlue.setMnemonic('u');
444             bgCyan.setMnemonic('c');
445             bgMagenta.setMnemonic('m');
446             bgWhite.setMnemonic('w');
447             bgLightGray.setMnemonic('a');
448             bgDarkGray.setMnemonic('d');
449             bgBlack.setMnemonic('b');
450 
451             getContentPane().setBackground(c);
452             FR.setExtendedState(Frame.MAXIMIZED_BOTH);
453 
454             bgYellow.addActionListener(this);
455             bgOrange.addActionListener(this);
456             bgRed.addActionListener(this);
457             bgPink.addActionListener(this);
458             bgLightGreen.addActionListener(this);
```

```
459           bgDarkGreen.addActionListener(this);
460           bgBlue.addActionListener(this);
461           bgCyan.addActionListener(this);
462           bgMagenta.addActionListener(this);
463           bgWhite.addActionListener(this);
464           bgLightGray.addActionListener(this);
465           bgDarkGray.addActionListener(this);
466           bgBlack.addActionListener(this);
467
468           btnThursRecycQuota.addActionListener(this);
469           btnRecyAttendanceReport.addActionListener(this);
470           btnRecyActivityReport.addActionListener(this);
471           btnRecyRegist.addActionListener(this);
472           btnRecyMonCrit.addActionListener(this);
473           btnTeacherClass.addActionListener(this);
474           btnFormClass.addActionListener(this);
475           btnSecuritySett.addActionListener(this);
476           btnLogOut.addActionListener(this);
477
478           About.addActionListener(this);
479           logOut.addActionListener(this);
480           Exit.addActionListener(this);
481           validate();
482
483           addWindowListener(new WindowAdapter()           //Activate Window 'X' Button
484           {
485               public void windowClosing(WindowEvent we)
486               {
487                   setVisible(false);
488                   System.exit(0);
489               }
490           });
491       }
492   //-------------------------------------------------------------------
493   //This method is used to execute the appropriate method when the user performs an action event
494       public void actionPerformed(ActionEvent ae)
495       {
496           if(bgYellow.isSelected()==true)     //ActionListener checks if the user has chosen a yellow background colour
497           {
498               getContentPane().setBackground(Color.yellow);       //The background colour changes to yellow
499           }
500
501           else if(bgOrange.isSelected()==true)
502           {
503               getContentPane().setBackground(Color.orange);
504           }
505
506           else if(bgRed.isSelected()==true)
507           {
508               getContentPane().setBackground(Color.red);
509           }
510
511           else if(bgPink.isSelected()==true)
512           {
513               getContentPane().setBackground(Color.pink);
514           }
515
516           else if(bgLightGreen.isSelected()==true)
517           {
```

```
518             getContentPane().setBackground(Color.green);
519         }
520
521         else if(bgDarkGreen.isSelected()==true)
522         {
523             getContentPane().setBackground(c);
524         }
525
526         else if(bgWhite.isSelected()==true)
527         {
528             getContentPane().setBackground(Color.white);
529         }
530
531         else if(bgBlue.isSelected()==true)
532         {
533             getContentPane().setBackground(Color.blue);
534         }
535
536         else if(bgCyan.isSelected()==true)
537         {
538             getContentPane().setBackground(Color.cyan);
539         }
540
541         else if(bgMagenta.isSelected()==true)
542         {
543             getContentPane().setBackground(Color.magenta);
544         }
545
546         else if(bgBlack.isSelected()==true)
547         {
548             getContentPane().setBackground(Color.black);
549         }
550
551         else if(bgDarkGray.isSelected()==true)
552         {
553             getContentPane().setBackground(Color.darkGray);
554         }
555
556         else if(bgLightGray.isSelected()==true)
557         {
558             getContentPane().setBackground(Color.lightGray);
559         }
560
561         if(ae.getSource()==btnThursRecycQuota)
562         {
563             FR.setVisible(false);
564             thursdayRecyclingQuota TRQ = new thursdayRecyclingQuota("");
565             TRQ.FR.setVisible(true);
566         }
567
568         if(ae.getSource()==btnRecyAttendanceReport)
569         {
570             FR.setVisible(false);
571             recyclerAttendanceReport RAttR = new recyclerAttendanceReport("");
572             RAttR.FR.setVisible(true);
573         }
574
575         if(ae.getSource()==btnRecyActivityReport)
576         {
```

```
577             FR.setVisible(false);
578             recyclingActivityReport RActR = new recyclingActivityReport("");
579             RActR.FR.setVisible(true);
580         }
581
582         if(ae.getSource()==btnRecyRegist)
583         {
584             FR.setVisible(false);
585             recyclerRegistration RR = new recyclerRegistration("");
586             RR.FR.setVisible(true);
587         }
588
589         if(ae.getSource()==btnRecyMonCrit)
590         {
591             FR.setVisible(false);
592             RoMCriterion RoMC = new RoMCriterion("");
593             RoMC.FR.setVisible(true);
594         }
595
596         if(ae.getSource()==btnTeacherClass)
597         {
598             FR.setVisible(false);
599             teacherClassPlan TCP = new teacherClassPlan("");
600             TCP.FR.setVisible(true);
601         }
602
603         if(ae.getSource()==btnFormClass)
604         {
605             FR.setVisible(false);
606             formClassroomLocation FCL = new formClassroomLocation("");
607             FCL.FR.setVisible(true);
608         }
609
610         if(ae.getSource()==btnSecuritySett)
611         {
612             FR.setVisible(false);
613             securitySett SS = new securitySett("");
614             SS.FR.setVisible(true);
615         }
616
617         if(ae.getSource()==About)
618         {
619             JOptionPane.showMessageDialog(this, "Recycling Activity Monitoring System Version 1.0\nDeveloper: Harris
    Rasheed\nDate Developed: 13th March 2010", "About", JOptionPane.INFORMATION_MESSAGE);
620         }
621
622         if(ae.getSource()==logOut||ae.getSource()==btnLogOut)
623         {
624             FR.setVisible(false);
625             Program EP = new Program("");
626             EP.FR.setVisible(true);
627         }
628         if(ae.getSource()==Exit)
629         {
630             System.exit(0);
631         }
632     }
633 }
634 //-----------------------------------------------------------------------------------------------------------
```

```
635   //Thursday Recycling Quota Menu Screen; This is the Thursday Recycling Quota Menu screen where the Morning Skip Monit
636   //input screen can be accessed or the Lunch Collection Rounds input screen.
637   class thursdayRecyclingQuota extends JFrame implements ActionListener
638   {
639       JFrame FR = new JFrame("Recycling Activity Monitoring System - Thursday Recycling Quota");
640       Container Obj1 = getContentPane();
641       GridBagLayout GBL = new GridBagLayout();
642       GridBagConstraints GBC = new GridBagConstraints();
643       JMenuBar MB = new JMenuBar();
644       JMenu file = new JMenu("File");
645       JMenuItem logOut = new JMenuItem("Log out");
646       JMenuItem Exit = new JMenuItem("Exit");
647       JButton MorningSkipbtn = new JButton("Morning Skip Monitor");
648       JButton LunchCollbtn = new JButton("Lunch Collection Rounds");
649       JButton btnBack = new JButton("Back");
650       JLabel lblThursRecycQuota = new JLabel("Thursday Recycling Quota");
651       Color c = new Color(6,69,1);
652       Font f = new Font("Comic Sans MS", Font.BOLD, 22);
653
654   //-----------------------------------------------------------------
655   //Constructor for the Thursday Recycling Quota Screen that places components on the Frame
656       public thursdayRecyclingQuota(String str)
657       {
658           super(str);
659
660           FR.setJMenuBar(MB);
661           MB.add(file);
662           file.add(logOut);
663           file.add(Exit);
664
665           getContentPane().setLayout(GBL);
666           FR.add(Obj1);
667
668           GBC.fill = GridBagConstraints.BOTH;
669           GBC.anchor = GridBagConstraints.PAGE_START;
670           GBC.gridwidth = 2;
671           GBC.gridheight = 2;
672           GBC.gridy = 1;
673           GBC.gridx = 1;
674           GBC.insets = new Insets(10,10,10,10);
675           GBC.fill = GridBagConstraints.VERTICAL;
676           GBL.setConstraints(lblThursRecycQuota,GBC);
677           lblThursRecycQuota.setFont(f);
678           lblThursRecycQuota.setHorizontalAlignment(JLabel.CENTER);
679           lblThursRecycQuota.setForeground(Color.white);
680           getContentPane().add(lblThursRecycQuota);
681
682           GBC.anchor = GridBagConstraints.CENTER;
683           GBC.gridy = 3;
684           GBC.ipady = 20;
685           GBC.ipadx = 100;
686           GBL.setConstraints(MorningSkipbtn,GBC);
687           getContentPane().add(MorningSkipbtn);
688
689           GBC.gridy = 5;
690           GBL.setConstraints(LunchCollbtn,GBC);
691           getContentPane().add(LunchCollbtn);
692
693           GBC.gridy = 20;
```

```
694             GBC.anchor = GridBagConstraints.PAGE_END;
695             GBC.insets = new Insets(250,10,10,10);
696             GBL.setConstraints(btnBack,GBC);
697             getContentPane().add(btnBack);
698 
699             getContentPane().setBackground(c);
700             FR.setExtendedState(Frame.MAXIMIZED_BOTH);
701 
702             MorningSkipbtn.addActionListener(this);
703             LunchCollbtn.addActionListener(this);
704             logOut.addActionListener(this);
705             Exit.addActionListener(this);
706             btnBack.addActionListener(this);
707             validate();
708         }
709     //-----------------------------------------------------------------
710     //This method is used to execute the appropriate method when the user performs an action event
711         public void actionPerformed(ActionEvent ae)
712         {
713             if(ae.getSource()==MorningSkipbtn)
714             {
715                 FR.setVisible(false);
716                 morningSkipMonitor MSM = new morningSkipMonitor("");
717                 MSM.FR.setVisible(true);
718             }
719             if(ae.getSource()==LunchCollbtn)
720             {
721                 FR.setVisible(false);
722                 lunchCollRounds LCR = new lunchCollRounds("");
723                 LCR.FR.setVisible(true);
724             }
725             if(ae.getSource()==btnBack)
726             {
727                 FR.setVisible(false);
728                 menuPage MP = new menuPage("");
729                 MP.FR.setVisible(true);
730             }
731             if(ae.getSource()==logOut)
732             {
733                 FR.setVisible(false);
734                 Program EP = new Program("");
735                 EP.FR.setVisible(true);
736             }
737             if(ae.getSource()==Exit)
738             {
739                 System.exit(0);
740             }
741         }
742     }
743     //-------------------------------------------------------------------------------------------------------------
744     //Morning Skip Monitor Screen; This is the screen where the user can input data
745     //collected by the Recycling Skip Supervisor on Thursday Mornings
746     class morningSkipMonitor extends JFrame implements ActionListener
747     {
748         JFrame FR = new JFrame("Recycling Activity Monitoring System - Morning Skip Monitor");
749         Container Obj1 = getContentPane();
750         GridBagLayout GBL = new GridBagLayout();
751         GridBagConstraints GBC = new GridBagConstraints();
752         JMenuBar MB = new JMenuBar();
```

```
753       JMenu file = new JMenu("File");
754       JMenuItem logOut = new JMenuItem("Log out");
755       JMenuItem exit = new JMenuItem("Exit");
756       JButton save = new JButton("Save");
757       JButton cancel = new JButton("Cancel");
758       JLabel lblMorningSkip = new JLabel("Morning Skip Monitor");
759       JLabel lblDate = new JLabel("Date: ");
760       JTextField txtDate = new JTextField(10);
761       Color c = new Color(6,69,1);
762       Font f = new Font("Comic Sans MS", Font.BOLD, 26);
763       Object records[][] = new Object[38][2];
764       String[] colNames = {"Form Class", "Points"};
765       JTable table = new JTable(records(38), colNames);
766       JScrollPane scroll = new JScrollPane(table);
767       public static final String DATE_FORMAT_NOW = "yyyy-MM-dd HH:mm:ss";
768
769   //-----------------------------------------------------------------
770   //Constructor for the Morning Skip Monitor Screen that places components on the Frame
771       public morningSkipMonitor(String str)
772       {
773           super(str);
774
775           FR.setJMenuBar(MB);
776           MB.add(file);
777           file.add(logOut);
778           file.add(exit);
779
780           getContentPane().setLayout(GBL);
781           FR.add(Obj1);
782
783           GBC.fill = GridBagConstraints.BOTH;
784           GBC.anchor = GridBagConstraints.CENTER;
785           GBC.gridwidth = 2;
786           GBC.gridy = 1;
787           GBC.gridx = 1;
788           GBC.insets = new Insets(10,10,10,10);
789           GBL.setConstraints(lblMorningSkip,GBC);
790           lblMorningSkip.setFont(f);
791           lblMorningSkip.setHorizontalAlignment(JLabel.CENTER);
792           lblMorningSkip.setForeground(Color.white);
793           getContentPane().add(lblMorningSkip);
794
795           GBC.gridy = 2;
796           GBC.gridwidth = 1;
797           GBL.setConstraints(lblDate,GBC);
798           lblDate.setForeground(Color.white);
799           lblDate.setLabelFor(txtDate);
800           getContentPane().add(lblDate);
801
802           GBC.gridx = 2;
803           GBL.setConstraints(txtDate,GBC);
804           getContentPane().add(txtDate);
805           txtDate.setText(systemDateSet());
806
807           GBC.gridy = 4;
808           GBC.gridx = 1;
809           GBC.gridwidth = 2;
810           GBL.setConstraints(scroll,GBC);
811           getContentPane().add(scroll);
```

```
812
813             GBC.gridy = 10;
814             GBC.ipady = 20;
815             GBC.ipadx = 100;
816             GBL.setConstraints(save,GBC);
817             getContentPane().add(save);
818
819             GBC.gridy = 12;
820             GBL.setConstraints(cancel,GBC);
821             getContentPane().add(cancel);
822
823             getContentPane().setBackground(c);
824             FR.setExtendedState(Frame.MAXIMIZED_BOTH);
825
826             save.addActionListener(this);
827             exit.addActionListener(this);
828             logOut.addActionListener(this);
829             cancel.addActionListener(this);
830             validate();
831         }
832     //---------------------------------------------------------------
833     //This method creates a 2D array with one column blank and the second column filled
834     //with ones as the default points value and the blank column available for input. This
835     //is used for JTable initialisation. The first parameter is the number of rows in the table
836         protected String[][] records(int length)
837         {
838             String records[][] = new String[length][2];
839
840             for(int i = 0; i < length; i++)
841             {
842                 records[i][1] = "1";
843                 records[i][0] = "";
844             }
845             return records;
846         }
847     //---------------------------------------------------------------
848     //This method finds the system date and returns the string in the format "DD/MM/YYYY"
849         protected static String systemDateSet()
850         {
851             Calendar cal = Calendar.getInstance();  //Access calendar object from utility library
852             SimpleDateFormat sdf = new SimpleDateFormat(DATE_FORMAT_NOW);   //Creates an object of the format
853             String a = sdf.format(cal.getTime());       //Retrieves time
854             return (a.substring(8,10) + "/" + a.substring(5,7) + "/" + a.substring(0,4));   //Returns date part of the
    string
855         }
856     //---------------------------------------------------------------
857     //This method is used to reference the location of each form class. This part of the
858     //system is now automated.
859         private String[][] referenceFormLocation(String[][] tableData, int rows)
860         {
861             try
862             {
863                 String problemReferences = "";          //problemReferences is used to store records that could not be
    referenced
864                 File file = new File("FormClassroomLocation.txt");          //Creates an object of the File
865                 RandomAccessFile RAF = new RandomAccessFile(file, "r");     //Creates an object of the Random Access File
866                 int recordSize = 13;                    //The size of each record in the random access file is 13 charact
867
868                 for(int c = 0; c < rows; c++)               //First loop iterates each array index
```

```
869             {
870                 boolean found = false;                      //Sets flag to false. This variable is used to identify
    problem input and reject them
871                 int length = tableData[c][0].length();      //Finds length of the string stored in the table current
    index
872
873                 int  j = 0;                                 //Loop counter
874                 while(j < tableData[c][0].length())         //Analyses string until
875                 {
876                     if(tableData[c][0].substring(j, j+1).equals("("))       //Checks if the user has input form class
    in full form; 7A(MC) opposed to just 7A
877                     {
878                         tableData[c][0] = tableData[c][0].substring(0, j);  //Removes the latter part if full form ha
    been used
879                         break;              //Terminate loop
880                     }
881                     j++;            //Positive iteration to counter
882                 }
883                 RAF.seek(0);                                //Goes to the beginning of the file
884
885                 for(int i = 0; i < 38; i++)                 //Second loop iterates the record number being searched in th
    file
886                 {
887                     if(tableData[c][0] == null)             //Tests if the index in the array storing the table data
    blank
888                     {
889                         break;                              //Terminates the loop if the table is blank
890                     }
891
892                     String currentLine = "", sCurrentLine = "";     //currentLine will store each record on each loop
    and sCurrentLine will look at certain fields of each record
893                     RAF.seek(i * recordSize);               //File pointer goes to the beginning of the each record o
    every loop
894
895                     for(int ct = 0; ct < recordSize; ct++)      //Reads each character of a record one by one
896                     {
897                         byte b = RAF.readByte();            //Read one character
898                         currentLine += (char) b;            //Convert to character from byte
899                     }
900
901                     sCurrentLine = currentLine.substring(0,3);      //Reads first part of record
902
903                     if((sCurrentLine.substring(2,3)).equals("("))   //Tests if the first field is meant to be two
    characters long; 7A opposed to 13A
904                     {
905                         sCurrentLine = currentLine.substring(0,2);      //Takes the first two characters of the recor
906                     }
907
908                     if(tableData[c][0].equalsIgnoreCase(sCurrentLine))  //Checks if the reference in the table and fi
    match
909                     {
910                         tableData[c][0] = (currentLine.substring(8,13)).trim();     //Assigns reference file data to
    table array
911                         found = true;   //Activates found flag
912                         break;          //Terminates loop when the record and table data is matched
913                     }
914                 }
915
916                 if(!found)
```

```
917                     {
918                         problemReferences += tableData[c][0] + " ";     //Adds any input problems
919                         tableData[c][0] = "";       //Clears problem index field
920                     }
921                 }
922                 RAF.close();            //Closes the Random Access File
923 
924                 if(!problemReferences.equals(""))       //Tests if there were no problems
925                 {
926                     JOptionPane.showMessageDialog(this, "The following classrooms could not be processed because they d
    exist.\n" + problemReferences,"Error Message", JOptionPane.WARNING_MESSAGE);   //Outputs problem input
927                 }
928                 return tableData;       //Returns array
929             }
930             catch(FileNotFoundException e)
931             {
932                 Toolkit.getDefaultToolkit().beep();     //Makes error sound
933                 JOptionPane.showMessageDialog(this, "The FormClassroomLocation.txt notepad file is missing from the cur
    directory. This process cannot function without this file.\nError Code: " + e,"File is Missing!",
    JOptionPane.ERROR_MESSAGE);   //Output any error if a file is not found
934             }
935             catch(Exception e)
936             {
937                 Toolkit.getDefaultToolkit().beep();
938                 JOptionPane.showMessageDialog(this, "An error has occured. Please contact Harris Rasheed to deal with t
    issue\nError Code: " + e,"Error Message", JOptionPane.ERROR_MESSAGE);    //Output any errors caught
939             }
940             return null;
941         }
942     //-----------------------------------------------------------------
943     //This method is used to store recycling statistics information. It updates existing
944     //data in the random access file. The first parameter is the array with records to
945     //be stored and the second parameter is the number of rows in the 2D array. This
946     //method finds the desired record to be updated, processes it and the moves it to
947     //the end of the file which allows the next search to be executed faster.
948         public void storeRecyStats(String[][] tableData, int rows)
949         {
950             try
951             {
952                 File file = new File("RecyclingActivityStats.txt");         //Creates object of file
953                 RandomAccessFile RAF = new RandomAccessFile(file, "rw");    //Creates object of Random Access File
954 
955                 int recordSize = 8;                     //The size of each record in the random access file is 8 charac
956                 int records = (int)(RAF.length())/recordSize;   //Calculates the number of records in the random access
957                 String errorStorage = "| - ";                   //Stores erroneous input data
958                 boolean found = false;                          //Creates flag
959 
960                 for(int c = 0; c < rows; c++)
961                 {
962                     for(int i = 0; i < records; i++)
963                     {
964                         String line = "";
965                         RAF.seek(i * recordSize);
966 
967                         for(int ct = 0; ct < recordSize; ct++)
968                         {
969                             byte b = RAF.readByte();
970                             line += (char)b;
971                         }
```

```
972
973                     String roomNo = (line.substring(0,5)).trim();
974                     int points = Integer.parseInt((line.substring(5,8)).trim());
975
976                     if(roomNo.equalsIgnoreCase(tableData[c][0]))        //Checks if the String is equal to the dele
    parameter
977                     {
978                         RAF.seek((records - 1) * (recordSize));         //File pointer looks at the last record
979                         byte[] ba = new byte[recordSize];               //Creates array with the size of one record
980                         RAF.readFully(ba);                              //Reads entire line and places it in the ar
981                         RAF.seek(i * recordSize);                       //File pointer looks at the place where the
    record was found
982                         RAF.write(ba);                                  //Overwrites the record location with the l
    record since it is to be deleted
983                         RAF.setLength(((records - 1)* (recordSize)));   //Truncates file and removes the last recor
    from the end of the file which has been moved to the deleted record's space
984                         RAF.seek(RAF.length());                         //File pointer looks at the end of the file

985
986                         for(int a = roomNo.length(); a < 5; a++)        //Adds spacing to the room number field so
    the record size is always of a set length
987                         {
988                             roomNo = roomNo + " ";
989                         }
990
991                         points += Integer.parseInt(tableData[c][1]);    //Adds points to points field in the record
992                         String StrPoints = Integer.toString(points);    //Converts points number to string
993
994                         for(int a = StrPoints.length(); a < 3; a++)     //Adds spacing to the points field so that
    record size is always of a set length
995                         {
996                             StrPoints = StrPoints + " ";
997                         }
998                         RAF.writeBytes(roomNo + StrPoints);             //Writes the room number and points fields
    together as a record to the file
999                         found = true;                                   //Activates flag
1000                        break;                                          //Terminates loop
1001                    }
1002                }
1003                if(!found && tableData[c][0] != null && tableData[c][0].trim() != "")   //Tests if the flag is stil
    false and if the field is not blank
1004                {
1005                    errorStorage += tableData[c][0] + " - ";        //Records erroneous input data
1006                }
1007            }
1008
1009            if(errorStorage.length() > 4)       //Tests if any erroneous data was input. The variable starts of wit
    characters
1010            {
1011                JOptionPane.showMessageDialog(this, "The following classrooms could not be processed because they d
    exist.\n" + errorStorage + "|","Error Message", JOptionPane.WARNING_MESSAGE);  //Output any errors caught
1012            }
1013            RAF.close();                            //Closes Random Access File
1014        }
1015        catch(FileNotFoundException e)
1016        {
1017            Toolkit.getDefaultToolkit().beep();
1018            JOptionPane.showMessageDialog(this, "The RecyclingActivityStats.txt notepad file is missing from the cu
    directory. This process cannot function without this file.\nError Code: " + e,"File is Missing!",
```

```
1018    JOptionPane.ERROR_MESSAGE);  //Output error if a file is not found
1019            }
1020            catch(Exception e)
1021            {
1022                Toolkit.getDefaultToolkit().beep();
1023                JOptionPane.showMessageDialog(this, "An error has occured. Please contact Harris Rasheed to deal with t
        issue\nError Code: " + e,"Error Message", JOptionPane.ERROR_MESSAGE);    //Output any errors caught
1024            }
1025        }
1026    //-----------------------------------------------------------------
1027    //This method keeps a log of all recycling statistics that are processed
1028    //The first parameter is the array of data that will be processed, the
1029    //second parameter is the number of rows in the table and the third
1030    //parameter stores the date that was input by the user
1031        public void logRecyStats(String[][] tableData, int rows, String date)
1032        {
1033            try
1034            {
1035                File file = new File("RecyclingActivityLog.txt");
1036                RandomAccessFile RAF = new RandomAccessFile(file, "rw");
1037
1038                int recordSize = 17;                        //The set record size is 17 characters in this RAF
1039                int records = (int)(RAF.length())/recordSize;   //Calculates the number of records in the RAF
1040
1041                RAF.seek(file.length());            //File pointer looks at the end of the file
1042
1043                for(int c = 0; c < rows; c++)
1044                {
1045                    if(tableData[c][0] != "")           //Tests if the current table row is blank
1046                    {
1047                        String roomNo = tableData[c][0];        //Assigns current index room number
1048                        String points = tableData[c][1];        //Assigns current index points
1049
1050                        for(int i = roomNo.length(); i < 5; i++)    //Adds spacing to make the room number field have s
        length
1051                        {
1052                            roomNo += " ";
1053                        }
1054
1055                        for(int i = points.length(); i < 2; i++)    //Adds spacing to make the points field have set le
1056                        {
1057                            points += " ";
1058                        }
1059                        RAF.writeBytes(roomNo + points + date);     //Writes the room number, points and date fields
        together as a record to the file
1060                    }
1061                }
1062                RAF.close();            //Closes Random Access File
1063            }
1064            catch(FileNotFoundException e)
1065            {
1066                Toolkit.getDefaultToolkit().beep();
1067                JOptionPane.showMessageDialog(this, "The RecyclingActivityLog.txt notepad file is missing from the curr
        directory. This process cannot function without this file.\nError Code: " + e,"File is Missing!",
        JOptionPane.ERROR_MESSAGE);    //Output any error if a file is not found
1068            }
1069            catch(Exception e)
1070            {
1071                Toolkit.getDefaultToolkit().beep();
```

```
1072                JOptionPane.showMessageDialog(this, "An error has occured. Please contact Harris Rasheed to deal with t
    issue\nError Code: " + e,"Error Message", JOptionPane.ERROR_MESSAGE);    //Output any errors caught
1073            }
1074        }
1075    //-----------------------------------------------------------------
1076    //This method is used to execute the appropriate method when the user performs an action event
1077        public void actionPerformed(ActionEvent ae)
1078        {
1079            if(ae.getSource()==save)
1080            {
1081                int rows = table.getRowCount();                         //Returns number of rows in the table
1082                int columns = table.getColumnCount();                   //Returns number of columns in the table
1083                String[][] tableData = new String[rows][columns];       //Creates array to store JTable's data
1084                String date = txtDate.getText();                        //Retrieves date from text field
1085
1086                for(int i = 0; i < rows; i++)
1087                {
1088                    for(int j = 0; j < columns; j++)
1089                    {
1090                        tableData[i][j] = (String) table.getValueAt(i,j);   //Loop which reads all the elements of JTab
    and stores it in a 2D array
1091                    }
1092                }
1093
1094                int count = 0;              //Used to count the number of blank rows in the table
1095                for(int i = 0; i < rows; i++)
1096                {
1097                    if(tableData[i][0].equals(""))      //Tests if the field is blank
1098                    {
1099                        count++;
1100                    }
1101                }
1102                if(count == rows)       //Checks if the entire table is blank
1103                {
1104                    JOptionPane.showMessageDialog(this, "The table is blank. Please input values to be processed.","Err
    Message", JOptionPane.ERROR_MESSAGE); //Output any errors caught
1105                    return;
1106                }
1107                referenceFormLocation(tableData, rows);     //Executes code and references form classes with their room
    numbers
1108                storeRecyStats(tableData, rows);            //Processes the table and stores it in a random access file
1109                logRecyStats(tableData, rows, date);        //Keeps a log of statistics that were input and the date
1110
1111                int ch = JOptionPane.showConfirmDialog(this, "Save Successful!\nWould you like to return to the Thursda
    Recycling Quota screen?", "Save Successful", JOptionPane.YES_NO_OPTION);
1112                if(ch  == JOptionPane.YES_OPTION)       //Tests if the user has pressed Yes
1113                {
1114                    FR.setVisible(false);
1115                    thursdayRecyclingQuota TRQ = new thursdayRecyclingQuota("");
1116                    TRQ.FR.setVisible(true);
1117                }
1118            }
1119            if(ae.getSource()==cancel)
1120            {
1121                FR.setVisible(false);
1122                thursdayRecyclingQuota TRQ = new thursdayRecyclingQuota("");
1123                TRQ.FR.setVisible(true);
1124            }
1125            if(ae.getSource()==logOut)
```

```
1126            {
1127                FR.setVisible(false);
1128                Program EP = new Program("");
1129                EP.FR.setVisible(true);
1130            }
1131            if(ae.getSource()==exit)
1132            {
1133                System.exit(0);
1134            }
1135        }
1136    }
1137    //-------------------------------------------------------------------------------------------------------------
1138    //Lunch Collection Rounds Screen; This is the screen where the user can input data
1139    //from the Thursday Recycling Rota slips used by volunteer recyclers
1140    class lunchCollRounds extends JFrame implements ActionListener
1141    {
1142        JFrame FR = new JFrame("Recycling Activity Monitoring System - Lunch Collection Rounds");
1143        Container Obj1 = getContentPane();
1144        GridBagLayout GBL = new GridBagLayout();
1145        GridBagConstraints GBC = new GridBagConstraints();
1146        JMenuBar MB = new JMenuBar();
1147        JMenu file = new JMenu("File");
1148        JMenuItem logOut = new JMenuItem("Log out");
1149        JMenuItem exit = new JMenuItem("Exit");
1150        JButton save = new JButton("Save");
1151        JButton btnRecyRegist = new JButton("Recycler Registration");
1152        JButton cancel = new JButton("Cancel");
1153        JLabel lblLunchColl = new JLabel("Lunch Collection Rounds");
1154        JLabel lblDate = new JLabel("Date: ");
1155        JTextField txtDate = new JTextField(10);
1156        Color c = new Color(6,69,1);
1157        Font f = new Font("Comic Sans MS", Font.BOLD, 26);
1158        String[] colNames = {"Room Number", "Points"};
1159        morningSkipMonitor MSM = new morningSkipMonitor("");
1160        JTable table = new JTable(MSM.records(120), colNames);
1161        JScrollPane scroll = new JScrollPane(table);
1162
1163    //-----------------------------------------------------------------
1164    //Constructor for the Lunch Collection Rounds Screen that places components on the Frame
1165        public lunchCollRounds(String str)
1166        {
1167            super(str);
1168
1169            FR.setJMenuBar(MB);
1170            MB.add(file);
1171            file.add(logOut);
1172            file.add(exit);
1173
1174            getContentPane().setLayout(GBL);
1175            FR.add(Obj1);
1176
1177            GBC.fill = GridBagConstraints.BOTH;
1178            GBC.anchor = GridBagConstraints.CENTER;
1179            GBC.gridwidth = 2;
1180            GBC.gridy = 1;
1181            GBC.gridx = 1;
1182            GBC.insets = new Insets(10,10,10,10);
1183            GBL.setConstraints(lblLunchColl,GBC);
1184            lblLunchColl.setFont(f);
```

```
1185            lblLunchColl.setHorizontalAlignment(JLabel.CENTER);
1186            lblLunchColl.setForeground(Color.white);
1187            getContentPane().add(lblLunchColl);
1188
1189            GBC.gridy = 2;
1190            GBC.gridwidth = 1;
1191            GBL.setConstraints(lblDate,GBC);
1192            lblDate.setForeground(Color.white);
1193            lblDate.setLabelFor(txtDate);
1194            getContentPane().add(lblDate);
1195
1196            GBC.gridx = 2;
1197            GBL.setConstraints(txtDate,GBC);
1198            getContentPane().add(txtDate);
1199            morningSkipMonitor MSM = new morningSkipMonitor("");
1200            txtDate.setText(MSM.systemDateSet());
1201
1202            GBC.gridy = 3;
1203            GBC.gridx = 1;
1204            GBC.gridwidth = 2;
1205            GBL.setConstraints(scroll,GBC);
1206            getContentPane().add(scroll);
1207
1208            GBC.gridheight = 1;
1209            GBC.gridy = 12;
1210            GBC.ipady = 10;
1211            GBC.ipadx = 100;
1212            GBL.setConstraints(save,GBC);
1213            getContentPane().add(save);
1214
1215            GBC.gridy = 14;
1216            GBL.setConstraints(cancel,GBC);
1217            getContentPane().add(cancel);
1218
1219            GBC.gridy = 16;
1220            GBL.setConstraints(btnRecyRegist, GBC);
1221            getContentPane().add(btnRecyRegist);
1222
1223            getContentPane().setBackground(c);
1224            FR.setExtendedState(Frame.MAXIMIZED_BOTH);
1225
1226            save.addActionListener(this);
1227            exit.addActionListener(this);
1228            logOut.addActionListener(this);
1229            cancel.addActionListener(this);
1230            btnRecyRegist.addActionListener(this);
1231            validate();
1232        }
1233    //-----------------------------------------------------------------
1234    //This method is used to read all the recycler names from the random access file
1235    //and returns them in an array so that they can be displayed to the user in a
1236    //input dialog choice box. This is used for recycler attendance and deletion.
1237        public String[] addRecyclers()
1238        {
1239            try
1240            {
1241                RandomAccessFile RAF = new RandomAccessFile("RecyclerAttendanceStats.txt", "r");
1242                int recordSize = 40;
1243                int records = (int) RAF.length()/(recordSize);                  //Calculates the number of records in t
```

```
1243  RAF
1244              String[] recyclerNames = new String[records + 1];               //Creates an array for the recycler nam
1245              recyclerNames[0] = "<Recycler Name>";                           //The extra index in the array is fille
      by this default String
1246
1247              for(int i = 1; i < records + 1; i++)    //Loops until all the random access file records have been read
1248              {
1249                  String currentLine = "";            //Initialises variable that will store records on every loop
1250                  RAF.seek((i-1) * recordSize);       //File pointer looks at the beginning of each record
1251
1252                  for(int ct = 0; ct < recordSize; ct++)      //The loop reads every record one by one
1253                  {
1254                      byte b = RAF.readByte();
1255                      currentLine += (char)b;
1256                  }
1257
1258                  currentLine = currentLine.substring(0,30);      //Reads first field of record
1259                  currentLine = currentLine.trim();               //Removes extra spacing from field
1260                  recyclerNames[i] = currentLine;                 //Assigns field to the array
1261              }
1262              RAF.close();
1263              return recyclerNames;           //Returns the array containing recycler names
1264          }
1265          catch(FileNotFoundException e)
1266          {
1267              Toolkit.getDefaultToolkit().beep();
1268              JOptionPane.showMessageDialog(this, "The RecyclerAttendanceStats.txt notepad file is missing from the
      current directory. This process cannot function without this file.\nError Code: " + e,"File is Missing!",
      JOptionPane.ERROR_MESSAGE); //Output any error if a file is not found
1269          }
1270          catch(Exception e)
1271          {
1272              Toolkit.getDefaultToolkit().beep();
1273              JOptionPane.showMessageDialog(this, "An error has occured. Please contact Harris Rasheed to deal with t
      issue\nError Code: " + e,"Error Message", JOptionPane.ERROR_MESSAGE);    //Output any errors caught
1274          }
1275          return null;
1276      }
1277  //-----------------------------------------------------------------
1278  //This method is used to process recycler attendance. When their names are selected by the
1279  //user, their attendance statistics are updated in the random access file so that the tally
1280  //report can be calculated. The first parameter is an array that contains the names of the
1281  //recyclers who attended the rounds and the second parameter is the number of recyclers who attended
1282      private void processRecyclerAttendance(String[] rNames, int numRecyclers)
1283      {
1284          try
1285          {
1286              File file = new File("RecyclerAttendanceStats.txt");
1287              RandomAccessFile RAF = new RandomAccessFile(file, "rw");
1288
1289              int recordSize = 40;
1290              int records = (int) RAF.length()/(recordSize);
1291
1292              for(int c = 0; c < numRecyclers; c++)       //Loops until all recyclers' attendance are processed
1293              {
1294                  RAF.seek(0);            //File pointer looks at the beginnning of the file
1295
1296                  for(int i = 0; i < records; i++)    //Loops until all records are read
1297                  {
```

```
1298                    if(rNames[c].equalsIgnoreCase("<Recycler Name>"))   //Tests if the user has tried to pass the
    default value
1299                    {
1300                        break;                  //Terminates current loop if so
1301                    }
1302
1303                    String line = "";               //Initialises variable that stores records
1304                    RAF.seek(i * recordSize);       //File pointer looks at the beginning of the file
1305
1306                    for(int ct = 0; ct < recordSize; ct++)      //Loop reads each random access file record
1307                    {
1308                        byte b = RAF.readByte();
1309                        line += (char)b;
1310                    }
1311
1312                    String recyName = (line.substring(0,30)).trim();                        //Retrieves first field
    the record
1313                    int attendance = Integer.parseInt((line.substring(38,40)).trim());      //Retrieves current
    attendance statistics
1314
1315                    if(rNames[c].equalsIgnoreCase(recyName))            //Checks if the record's field is the same
    the array's data
1316                    {
1317                        attendance++;                                       //Adds to the current attendance tally
1318                        RAF.seek((i * (recordSize)) + 38);                  //Goes the point field of the record
1319                        String strAttend = Integer.toString(attendance);    //Converts attendance tally to a string
    it can be written to an RAF
1320
1321                        for(int d = strAttend.length(); d < 2; d++)     //Loops until the string is of a set size
1322                        {
1323                            strAttend += " ";
1324                        }
1325                        RAF.writeBytes(strAttend);      //Updates the record and overwrites the old attendance tall
1326                        break;
1327                    }
1328                }
1329            }
1330            RAF.close();
1331        }
1332        catch(FileNotFoundException e)
1333        {
1334            Toolkit.getDefaultToolkit().beep();
1335            JOptionPane.showMessageDialog(this, "The RecyclerAttendanceStats.txt notepad file is missing from the
    current directory. This process cannot function without this file.\nError Code: " + e,"File is Missing!",
    JOptionPane.ERROR_MESSAGE); //Output any error if a file is not found
1336        }
1337        catch(Exception e)
1338        {
1339            Toolkit.getDefaultToolkit().beep();
1340            JOptionPane.showMessageDialog(this, "An error has occured. Please contact Harris Rasheed to deal with t
    issue\nError Code: " + e,"Error Message", JOptionPane.ERROR_MESSAGE);    //Output any errors caught
1341        }
1342    }
1343    //-----------------------------------------------------------------
1344    //This method is used to keep a log of recycler attendance. This is stored in a random access file
1345    //The first parameter is the names of the recyclers who attended, the second parameter is the
1346    //number of recyclers who attended and the third parameter is the date of their attendace. This method
1347    //is used for the Recycler of the Month Candidate Report
1348        private void logRecyclerAttendance(String[] rNames, int totalRecyclerNames, String date)
```

```
1349        {
1350            try
1351            {
1352                File file = new File("RecyclerAttendanceLog.txt");
1353                RandomAccessFile RAF = new RandomAccessFile(file, "rw");
1354
1355                int recordSize = 40;
1356                int records = (int) RAF.length()/(recordSize);
1357
1358                for(int c = 0; c < totalRecyclerNames; c++)
1359                {
1360                    RAF.seek(file.length());        //File pointer looks at the end of the file
1361
1362                    for(int i = 0; i < records; i++)
1363                    {
1364                        if(rNames[c].equalsIgnoreCase("<Recycler Name>"))   //Checks if the default value has been pass
1365                        {
1366                            break;      //Terminates loop if so
1367                        }
1368
1369                        for(int j = rNames[c].length(); j < 30; j++)    //Sets length of Recycler names to record's fie
    size
1370                        {
1371                            rNames[c] += " ";
1372                        }
1373                        RAF.writeBytes(rNames[c]+date);     //Writes name and date to the random access file
1374                    }
1375                }
1376                RAF.close();
1377            }
1378            catch(FileNotFoundException e)
1379            {
1380                Toolkit.getDefaultToolkit().beep();
1381                JOptionPane.showMessageDialog(this, "The RecyclerAttendanceLog.txt notepad file is missing from the cur
    directory. This process cannot function without this file.\nError Code: " + e,"File is Missing!",
    JOptionPane.ERROR_MESSAGE);   //Output any error if a file is not found
1382            }
1383            catch(Exception e)
1384            {
1385                Toolkit.getDefaultToolkit().beep();
1386                JOptionPane.showMessageDialog(this, "An error has occured. Please contact Harris Rasheed to deal with t
    issue\nError Code: " + e,"Error Message", JOptionPane.ERROR_MESSAGE);    //Output any errors caught
1387            }
1388        }
1389    //-----------------------------------------------------------------
1390    //This method is used to execute the appropriate method when the user performs an action event
1391        public void actionPerformed(ActionEvent ae)
1392        {
1393            if(ae.getSource()==save)
1394            {
1395                int rows = table.getRowCount();
1396                int columns = table.getColumnCount();
1397                String date = txtDate.getText();
1398                String[][] tableData = new String[rows][columns];
1399
1400                for(int i = 0; i < rows; i++)
1401                {
1402                    for(int j = 0; j < columns; j++)
1403                    {
```

```
1404                        tableData[i][j] = (String) table.getValueAt(i,j);   //Loops until all the values of the table h
    been assigned to a 2D array
1405                    }
1406                }
1407
1408                int count = 0;              //Used to count the number of blank rows in the table
1409                for(int i = 0; i < rows; i++)
1410                {
1411                    if(tableData[i][0].equals(""))      //Tests if the field is blank
1412                    {
1413                        count++;
1414                    }
1415                }
1416                if(count == rows)       //Checks if the entire table is blank
1417                {
1418                    JOptionPane.showMessageDialog(this, "The table is blank. Please input values to be processed.","Err
    Message", JOptionPane.ERROR_MESSAGE); //Output any errors caught
1419                    return;
1420                }
1421
1422                int totalRecyclerNames;
1423
1424                try
1425                {
1426                    totalRecyclerNames = Integer.parseInt((String)JOptionPane.showInputDialog(null, "How many recyclers
    participated this week?","Number of Recyclers", 3));        //Prompts the user for the number of recyclers that att
1427                }
1428                catch(Exception e)
1429                {
1430                    JOptionPane.showMessageDialog(this, "Error! Please input a number.","Error", JOptionPane.ERROR_MESS
1431                    return;
1432                }
1433
1434                morningSkipMonitor MSM = new morningSkipMonitor("");        //Creates object of class in order to acces
    method
1435                MSM.storeRecyStats(tableData, rows);                //Accesses method of morningSkipMonitor class to st
    recycling statistics
1436                Frame frame = new Frame();                          //Creates frame for dialog box
1437
1438                String[] recyNames = new String[totalRecyclerNames];        //Creates array to store names of recyclers
    attended
1439
1440                for(int i = 0; i < totalRecyclerNames; i++)     //Loops the message box until the appropriate number
1441                {
1442                    recyNames[i] = (String) JOptionPane.showInputDialog(frame, "Please select a recycler name", "Recycl
    Attendance " + (i + 1), JOptionPane.PLAIN_MESSAGE, null, addRecyclers(),"<Recycler Name>");
1443                }
1444                processRecyclerAttendance(recyNames, totalRecyclerNames);       //Processes recycler attendance
1445                logRecyclerAttendance(recyNames, totalRecyclerNames, date);     //Logs recycler attendance
1446            }
1447            if(ae.getSource()==cancel)
1448            {
1449                FR.setVisible(false);
1450                thursdayRecyclingQuota TRQ = new thursdayRecyclingQuota("");
1451                TRQ.FR.setVisible(true);
1452            }
1453            if(ae.getSource()==btnRecyRegist)
1454            {
1455                int ch = JOptionPane.showConfirmDialog(this, "Would you like to go to the Recycler Registration screen
```

```
1455   without saving your changes?", "Exit without Save Changes?", JOptionPane.YES_NO_OPTION);
1456                if(ch  == JOptionPane.YES_OPTION)       //Checks if the user pressed Yes
1457                {
1458                    FR.setVisible(false);
1459                    recyclerRegistration RR = new recyclerRegistration("");
1460                    RR.FR.setVisible(true);
1461                }
1462            }
1463            if(ae.getSource()==logOut)
1464            {
1465                FR.setVisible(false);
1466                Program EP = new Program("");
1467                EP.FR.setVisible(true);
1468            }
1469            if(ae.getSource()==exit)
1470            {
1471                System.exit(0);
1472            }
1473        }
1474   }
1475   //-------------------------------------------------------------------------------------------------------------------
1476   //Recycler Attendance Report Screen; This is the screen where the user can view
1477   //the recycler attendance report that displays the attendance in tally format.
1478   class recyclerAttendanceReport extends JFrame implements ActionListener
1479   {
1480       JFrame FR = new JFrame("Recycling Activity Monitoring System - Recycler Attendance Report");
1481       Container Obj1 = getContentPane();
1482       GridBagLayout GBL = new GridBagLayout();
1483       GridBagConstraints GBC = new GridBagConstraints();
1484       JMenuBar MB = new JMenuBar();
1485       JMenu file = new JMenu("File");
1486       JMenuItem logOut = new JMenuItem("Log out");
1487       JMenuItem exit = new JMenuItem("Exit");
1488       JButton print = new JButton("Print");
1489       JButton cancel = new JButton("Cancel");
1490       JLabel lblRecyAttendRep = new JLabel("Recycler Attendance Report");
1491       Color c = new Color(6,69,1);
1492       Font f = new Font("Comic Sans MS", Font.BOLD, 26);
1493       String[] colNames = {"Recycler", "Form Class", "Attendance Tally"};
1494       File raf = new File("RecyclerAttendanceStats.txt");
1495       JTable table = new JTable(loadTable(raf, 40, 30, 38), colNames);
1496       JScrollPane scroll = new JScrollPane(table);        //Allow a large table to be viewed using a scroll pane
1497
1498   //-----------------------------------------------------------------
1499   //Constructor for the Recycler Attendance Report Screen that places components on the Frame
1500       public recyclerAttendanceReport(String str)
1501       {
1502           super(str);
1503
1504           FR.setJMenuBar(MB);
1505           MB.add(file);
1506           file.add(logOut);
1507           file.add(exit);
1508
1509           getContentPane().setLayout(GBL);
1510           FR.add(Obj1);
1511
1512           GBC.fill = GridBagConstraints.BOTH;
1513           GBC.anchor = GridBagConstraints.CENTER;
```

```
1514            GBC.gridwidth = 2;
1515            GBC.gridy = 1;
1516            GBC.gridx = 1;
1517            GBC.insets = new Insets(10,10,10,10);
1518            GBL.setConstraints(lblRecyAttendRep,GBC);
1519            lblRecyAttendRep.setFont(f);
1520            lblRecyAttendRep.setHorizontalAlignment(JLabel.CENTER);
1521            lblRecyAttendRep.setForeground(Color.white);
1522            getContentPane().add(lblRecyAttendRep);
1523
1524            GBC.gridy = 4;
1525            GBC.gridx = 1;
1526            GBC.gridwidth = 2;
1527            GBL.setConstraints(scroll,GBC);
1528            getContentPane().add(scroll);
1529
1530            GBC.gridy = 10;
1531            GBC.ipady = 20;
1532            GBC.ipadx = 100;
1533            GBL.setConstraints(print,GBC);
1534            getContentPane().add(print);
1535
1536            GBC.gridy = 12;
1537            GBL.setConstraints(cancel,GBC);
1538            getContentPane().add(cancel);
1539
1540            getContentPane().setBackground(c);
1541            FR.setExtendedState(Frame.MAXIMIZED_BOTH);
1542
1543            print.addActionListener(this);
1544            exit.addActionListener(this);
1545            logOut.addActionListener(this);
1546            cancel.addActionListener(this);
1547            validate();
1548        }
1549    //-----------------------------------------------------------------
1550    //This method is used to load the table when the Recycler Attendance Report is accessed
1551    //It reads the random access file for data which is output to JTable
1552        protected String[][] loadTable(File file, int recordSize, int firstField, int secondField)
1553        {
1554            try
1555            {
1556                RandomAccessFile RAF = new RandomAccessFile(file, "r");     //Creates an object of the Random Access Fi
1557                int records = (int) RAF.length()/recordSize;                //Calculates the number of records exist in
      file
1558                String[][] tableData = new String[records][3];
1559
1560                for(int c = 0; c < records; c++)                //First loop iterates each array index
1561                {
1562                    RAF.seek(c * recordSize);           //File pointer goes to the start of each record on every iterat
1563                    String currentLine = "";            //currentLine is used to store each record being searched
1564                    for(int i = 0; i < recordSize; i++)
1565                    {
1566                        byte b = RAF.readByte();        //Reads each character and stores it as a byte
1567                        currentLine += (char) b;        //Converts each character from byte to character and then to st
      and collects each character on each loop
1568                    }
1569
1570                    String first = (currentLine.substring(0,firstField)).trim();            //Reads first field
```

```
1571                String second = (currentLine.substring(firstField,secondField)).trim(); //Reads second field
1572                String third = (currentLine.substring(secondField,recordSize)).trim();  //Reads third field
1573
1574                tableData[c][0] = first;                //Assigns first field to 2D array
1575                tableData[c][1] = second;               //Assigns second field to 2D array
1576                tableData[c][2] = third;                //Assigns third field to 2D array
1577            }
1578            RAF.close();
1579            quickSort(tableData, 0, records-1, records);        //Performs a quick sort on the array
1580            return tableData;                       //returns 2D array of Random Access file records
1581        }
1582        catch(FileNotFoundException e)
1583        {
1584            Toolkit.getDefaultToolkit().beep();
1585            JOptionPane.showMessageDialog(this, "An important system file is missing from the current directory. Th
    process cannot function without this file.\nError Code: " + e,"File is Missing!", JOptionPane.ERROR_MESSAGE); //Outp
    any error if a file is not found
1586        }
1587        catch(Exception e)
1588        {
1589            Toolkit.getDefaultToolkit().beep();
1590            JOptionPane.showMessageDialog(this, "An error has occured. Please contact Harris Rasheed to deal with t
    issue\nError Code: " + e,"Error Message", JOptionPane.ERROR_MESSAGE);    //Output any errors caught
1591        }
1592        return null;
1593    }
1594    //-----------------------------------------------------------------
1595    //This method uses quick sort to sort the array parameter in descending order according to the third column.
1596    //The first parameter is the array that needs to be sorted, the second parameter is the start of the unsorted
1597    //part of the array's index, the third parameter is the end of the unsorted array's index and the fourth parameter
1598    //is the total number of records being sorted which needs to be monitored in order to know when the sort is complet
1599        protected String[][] quickSort(String[][] tableData, int start, int finish, int records)
1600        {
1601            int left, right, count = 0;     //left is the most left unsorted index and right is the most right unsorted
    index, count is used to tell when the sort is completed
1602            String temp1, temp2, temp3;     //temporary variables used to store the array data when it is being swapped
1603            left = start;
1604            right = finish;
1605            int pos = (left + right)/2;     //the middle position is obtained
1606            int pivot = Integer.parseInt(tableData[pos][2]);    //this is the middle position of the unsorted part of t
    array
1607
1608            while(right > left)     //While both half of the arrays are being sorted. This condition becomes nullified
    the right and left variables cross over to the other half of the array
1609            {
1610                while(Integer.parseInt(tableData[left][2]) > pivot)     //Executes loop until it finds an element that
    smaller than the pivot in the first half of the array
1611                {
1612                    left++;
1613                    count++;        //This variable is used to count the number of sorted items
1614                }
1615
1616                while(pivot > Integer.parseInt(tableData[right][2]))    //Executes loop until it finds an element that
    greater than the pivot in the second half of the array
1617                {
1618                    right--;
1619                    count++;        //This variable is used to count the number of sorted items
1620                }
1621
```

```
1622                if(count==records)      //Tests if the entire array has been sorted
1623                {
1624                    return tableData;   //Returns sorted array
1625                }
1626
1627                if(left <= right)       //Checks if the left position is smaller than the right position
1628                {
1629                    temp1 = tableData[left][0];         //Copies Column 1 contents of the array to a temporary variable
1630                    temp2 = tableData[left][1];         //Copies Column 2 contents of the array to a temporary variable
1631                    temp3 = tableData[left][2];         //Copies Column 3 contents of the array to a temporary variable
1632                    tableData[left][0] = tableData[right][0];       //Overwrites the first half of the array index that
      found unsorted with the larger element
1633                    tableData[left][1] = tableData[right][1];
1634                    tableData[left][2] = tableData[right][2];
1635                    tableData[right][0] = temp1;            //Overwrites the second half of the array index that was fo
      unsorted with the smaller element
1636                    tableData[right][1] = temp2;
1637                    tableData[right][2] = temp3;
1638                    left++;         //Reduces the size of the array for the next sort sequence
1639                    right--;        //Reduces the size of the array for the next sort sequence
1640                }
1641
1642                if(start < right)           //Tests if the right boundary of the unsorted array has not reach the start
      the sorted array
1643                {
1644                    quickSort(tableData, start, right, records);    //Recursion of sorting sequence with smaller array
      parameters
1645                }
1646
1647                if(left < finish)           //Tests if the left boundary of the unsorted array has not reach the end of
      sorted array
1648                {
1649                    quickSort(tableData, left, finish, records);    //Recursion of sorting sequence with smaller array
      parameters
1650                }
1651            }
1652            return null;        //Returns null if the sort fails because of file reading problems which would pass an
      erroneous number of records
1653        }
1654    //-----------------------------------------------------------------
1655    //This method is used to execute the appropriate method when the user performs an action event
1656        public void actionPerformed(ActionEvent ae)
1657        {
1658            if(ae.getSource()==print)       //Checks if the print button has been pressed
1659            {
1660                try
1661                {
1662                    if (!table.print())     //Checks if the user has cancelled the print job and initiates the print me
1663                    {
1664                        JOptionPane.showMessageDialog(this, "The user has cancelled the print job.", "Cancelled Print J
      JOptionPane.INFORMATION_MESSAGE);
1665                    }
1666                }
1667                catch (java.awt.print.PrinterException e)           //Catches printer exception
1668                {
1669                    JOptionPane.showMessageDialog(this, "Unable to print due to " + e, "Print Job Error",
      JOptionPane.ERROR_MESSAGE);
1670                    Toolkit.getDefaultToolkit().beep();
1671                }
```

```
1672            }
1673            if(ae.getSource()==cancel)
1674            {
1675                FR.setVisible(false);
1676                menuPage MP = new menuPage("");
1677                MP.FR.setVisible(true);
1678            }
1679            if(ae.getSource()==logOut)
1680            {
1681                FR.setVisible(false);
1682                Program EP = new Program("");
1683                EP.FR.setVisible(true);
1684                Toolkit.getDefaultToolkit().beep();
1685            }
1686            if(ae.getSource()==exit)
1687            {
1688                System.exit(0);
1689            }
1690        }
1691    }
1692    //-------------------------------------------------------------------------------------------------------------
1693    //Recycling Activity Report Screen; This is the screen where the user can view the recycling
1694    //activity statistics of each classroom in school over the current academic year.
1695    class recyclingActivityReport extends JFrame implements ActionListener
1696    {
1697        JFrame FR = new JFrame("Recycling Activity Monitoring System - Recycling Activity Report");
1698        Container Obj1 = getContentPane();
1699        GridBagLayout GBL = new GridBagLayout();
1700        GridBagConstraints GBC = new GridBagConstraints();
1701        JMenuBar MB = new JMenuBar();
1702        JMenu file = new JMenu("File");
1703        JMenuItem logOut = new JMenuItem("Log out");
1704        JMenuItem exit = new JMenuItem("Exit");
1705        JButton print = new JButton("Print");
1706        JButton cancel = new JButton("Cancel");
1707        JLabel lblRecyActivityRep = new JLabel("Recycling Activity Report");
1708        Color c = new Color(6,69,1);
1709        Font f = new Font("Comic Sans MS", Font.BOLD, 26);
1710        String[] colNames = {"Room Number", "Teacher", "Points"};
1711        JTable table = new JTable(loadTable(), colNames);
1712        JScrollPane scroll = new JScrollPane(table);
1713
1714    //-----------------------------------------------------------------
1715    //Constructor for the Recycling Activity Report Screen that places components on the Frame
1716        public recyclingActivityReport(String str)
1717        {
1718            super(str);
1719
1720            FR.setJMenuBar(MB);
1721            MB.add(file);
1722            file.add(logOut);
1723            file.add(exit);
1724
1725            getContentPane().setLayout(GBL);
1726            FR.add(Obj1);
1727
1728            GBC.fill = GridBagConstraints.BOTH;
1729            GBC.anchor = GridBagConstraints.CENTER;
1730            GBC.gridwidth = 2;
```

```
1731            GBC.gridy = 1;
1732            GBC.gridx = 1;
1733            GBC.insets = new Insets(10,10,10,10);
1734            GBL.setConstraints(lblRecyActivityRep,GBC);
1735            lblRecyActivityRep.setFont(f);
1736            lblRecyActivityRep.setHorizontalAlignment(JLabel.CENTER);
1737            lblRecyActivityRep.setForeground(Color.white);
1738            getContentPane().add(lblRecyActivityRep);
1739
1740            GBC.gridy = 4;
1741            GBC.gridx = 1;
1742            GBC.gridwidth = 2;
1743            GBL.setConstraints(scroll,GBC);
1744            getContentPane().add(scroll);
1745
1746            GBC.gridy = 10;
1747            GBC.ipady = 20;
1748            GBC.ipadx = 100;
1749            GBL.setConstraints(print,GBC);
1750            getContentPane().add(print);
1751
1752            GBC.gridy = 12;
1753            GBL.setConstraints(cancel,GBC);
1754            getContentPane().add(cancel);
1755
1756            getContentPane().setBackground(c);
1757            FR.setExtendedState(Frame.MAXIMIZED_BOTH);
1758
1759            print.addActionListener(this);
1760            exit.addActionListener(this);
1761            logOut.addActionListener(this);
1762            cancel.addActionListener(this);
1763            validate();
1764        }
1765    //-----------------------------------------------------------------
1766    //This method is used to load the table when the Recycling Activity Report is accessed
1767    //It reads the random access file for data which is output to the JTable
1768        private String[][] loadTable()
1769        {
1770            try
1771            {
1772                String[][] tableData = new String[122][3];
1773                int recordSize = 8;                         //Each record in the RAF is 8 characters long
1774                RandomAccessFile raf = new RandomAccessFile("RecyclingActivityStats.txt", "r");
1775                int records = (int) raf.length()/recordSize;
1776
1777                for(int c = 0; c < records; c++)        //First loop iterates each array index
1778                {
1779                    raf.seek(c * recordSize);           //File pointer goes to the start of each record on evert iterat
1780                    String currentLine = "";            //currentLine is used to store each record being searched
1781                    for(int i = 0; i < recordSize; i++)
1782                    {
1783                        byte b = raf.readByte();        //Reads each character and stores it as a byte
1784                        currentLine += (char) b;        //Converts each character from byte to character and then to st
      and collects each character on each loop
1785                    }
1786
1787                    String roomNo = (currentLine.substring(0,5)).trim();        //Takes a substring of the current reco
      get the room number part
```

```
1788                    String points = (currentLine.substring(5,8)).trim();        //Takes a substring of the current reco
    get the points segment
1789
1790                    tableData[c][0] = roomNo;               //Assigns room number to 2D array
1791                    tableData[c][2] = points;               //Assigns points to 2D array
1792                }
1793                raf.close();
1794                referenceTeachersClass(tableData, records);     //Passes tableData to 'referenceTeachersClass' method t
    reference and fill out the empty second column
1795                return tableData;
1796            }
1797            catch(FileNotFoundException e)
1798            {
1799                Toolkit.getDefaultToolkit().beep();
1800                JOptionPane.showMessageDialog(this, "The RecyclingActivityStats.txt notepad file is missing from the cu
    directory. This process cannot function without this file.\nError Code: " + e,"File is Missing!",
    JOptionPane.ERROR_MESSAGE);  //Output any error if a file is not found
1801            }
1802            catch(Exception e)
1803            {
1804                Toolkit.getDefaultToolkit().beep();
1805                JOptionPane.showMessageDialog(this, "An error has occured. Please contact Harris Rasheed to deal with t
    issue\nError Code: " + e,"Error Message", JOptionPane.ERROR_MESSAGE);    //Output any errors caught
1806            }
1807            return null;
1808        }
1809    //-----------------------------------------------------------------
1810    //This method is used to reference the room number of classrooms with
1811    //the teacher who occupies the room. This is to provide more detail on
1812    //the report. The first parameter is the array with an empty column and
1813    //a column of room numbers to be referenced and the second parameter is
1814    //the number of rows in the array.
1815        private String[][] referenceTeachersClass(String[][] tableData, int rows)
1816        {
1817            try
1818            {
1819                for(int c = 0; c < rows; c++)
1820                {
1821                    RandomAccessFile RAF = new RandomAccessFile("TeacherClassroomPlan.txt", "r");
1822                    int recordSize = 35;                    //Each record in the RAF is 35 characters long
1823                    int records = ((int) RAF.length())/recordSize;
1824                    String currentLine = "";                //currentLine is used to store each record of the file
1825                    RAF.seek(0);
1826
1827                    for(int i = 0; i < records; i++)
1828                    {
1829                        if(tableData[c][0] == null)         //Checks if there is a blank record in the array/table
1830                        {
1831                            break;                          //Terminates the loop if a record is blank
1832                        }
1833
1834                        String line = "";               //Initialises variable that stores records
1835                        RAF.seek(i * recordSize);       //File pointer looks at the beginning of each record
1836
1837                        for(int ct = 0; ct < recordSize; ct++)      //Reads each record one by one
1838                        {
1839                            byte b = RAF.readByte();
1840                            line += (char)b;
1841                        }
```

```
1842
1843                    currentLine = line.substring(0,5).trim();               //Reads first field of the record
1844
1845                    if((tableData[c][0].trim()).equalsIgnoreCase(currentLine))  //Checks if the reference in the ta
    and file match
1846                    {
1847                        tableData[c][1] = (line.substring(5,35)).trim();        //Assigns reference file data to th
    table's current row
1848                        break;                                  //Terminates current loop
1849                    }
1850                }
1851                RAF.close();
1852            }
1853            bubbleSort(tableData, rows);        //Bubble sorts array before being presented on the report
1854            return tableData;                   //Returns sorted table
1855        }
1856        catch(FileNotFoundException e)
1857        {
1858            Toolkit.getDefaultToolkit().beep();
1859            JOptionPane.showMessageDialog(this, "The TeacherClassroomPlan.txt notepad file is missing from the curr
    directory. This process cannot function without this file.\nError Code: " + e,"File is Missing!",
    JOptionPane.ERROR_MESSAGE);    //Output any error if a file is not found
1860        }
1861        catch(Exception e)
1862        {
1863            Toolkit.getDefaultToolkit().beep();
1864            JOptionPane.showMessageDialog(this, "An error has occured. Please contact Harris Rasheed to deal with t
    issue\nError Code: " + e,"Error Message", JOptionPane.ERROR_MESSAGE);    //Output any errors caught
1865        }
1866        return null;
1867    }
1868    //-----------------------------------------------------------------
1869    //This method uses the bubble sort algorithm to sort the array parameter in DESCENDING order according to the 3rd
    column.
1870    //The first parameter is the array that needs to be sorted, the second parameter is the number of records in the ar
1871        protected String[][] bubbleSort(String [][] tableData, int rows)
1872        {
1873            for(int top = tableData.length - 1; top > 0; top--)     //Loops and focuses sort on the unsorted part
1874            {
1875                for(int upper = 1; upper <= top; upper++)
1876                {
1877                    int lower = upper - 1;
1878                    if(tableData[lower][2] == null||tableData[upper][2] == null)    //Checks if the table row is blank
1879                    {
1880                        break;
1881                    }
1882
1883                    if(Integer.parseInt(tableData[lower][2]) < Integer.parseInt(tableData[upper][2]))   //Compares two
    indexes of the table
1884                    {
1885                        String temp1 = tableData[upper][0];     //Stores each field in a temporary variable
1886                        String temp2 = tableData[upper][1];
1887                        String temp3 = tableData[upper][2];
1888                        tableData[upper][0] = tableData[lower][0];      //Overwrites the previous index with the larger
    value
1889                        tableData[upper][1] = tableData[lower][1];
1890                        tableData[upper][2] = tableData[lower][2];
1891                        tableData[lower][0] = temp1;            //Writes the temporary variables' value to the higher a
    index
```

```
1892                        tableData[lower][1] = temp2;
1893                        tableData[lower][2] = temp3;
1894                    }
1895                }
1896            }
1897            return tableData;       //Returns sorted table
1898        }
1899    //-----------------------------------------------------------------
1900    //This method is used to execute the appropriate method when the user performs an action event
1901        public void actionPerformed(ActionEvent ae)
1902        {
1903            if(ae.getSource()==print)
1904            {
1905                try
1906                {
1907                    if(!table.print())      //Checks if the user has cancelled the print job
1908                    {
1909                        JOptionPane.showMessageDialog(this, "The user has cancelled the print job.", "Cancelled Print J
    JOptionPane.INFORMATION_MESSAGE);
1910                    }
1911                }
1912                catch(java.awt.print.PrinterException e)            //Catches printer exception
1913                {
1914                    JOptionPane.showMessageDialog(this, "Unable to print due to " + e, "Print Job Error",
    JOptionPane.ERROR_MESSAGE);
1915                    Toolkit.getDefaultToolkit().beep();
1916                }
1917
1918            }
1919            if(ae.getSource()==cancel)
1920            {
1921                FR.setVisible(false);
1922                menuPage MP = new menuPage("");
1923                MP.FR.setVisible(true);
1924            }
1925            if(ae.getSource()==logOut)
1926            {
1927                FR.setVisible(false);
1928                Program EP = new Program("");
1929                EP.FR.setVisible(true);
1930                Toolkit.getDefaultToolkit().beep();
1931            }
1932            if(ae.getSource()==exit)
1933            {
1934                System.exit(0);
1935            }
1936        }
1937    }
1938    //-----------------------------------------------------------------------------------------------------------------
1939    //Recycler Registration Screen; This is the screen where the user can add a recycler
1940    //to the system which will allow them to process their attendance.
1941    class recyclerRegistration extends JFrame implements ActionListener
1942    {
1943        JFrame FR = new JFrame("Recycling Activity Monitoring System - Recycler Registration");
1944        Container Obj1 = getContentPane();
1945        GridBagLayout GBL = new GridBagLayout();
1946        GridBagConstraints GBC = new GridBagConstraints();
1947        JMenuBar MB = new JMenuBar();
1948        JMenu file = new JMenu("File");
```

```
1949        JMenuItem logOut = new JMenuItem("Log out");
1950        JMenuItem Exit = new JMenuItem("Exit");
1951        JButton btnRegister = new JButton("Register");
1952        JButton btnCancel = new JButton("Cancel");
1953        JButton btnDelRecy = new JButton ("Delete Recycler Registration");
1954        JLabel lblRecyName = new JLabel("Recycler Name: ");
1955        JLabel lblForm = new JLabel("Form: ");
1956        JLabel lblRecyRegist = new JLabel("Recycler Registration");
1957        JTextField txtRecyName = new JTextField(30);
1958        Choice chForm = new Choice();
1959        Color c = new Color(6,69,1);
1960        Font f = new Font("Comic Sans MS", Font.BOLD, 22);
1961        Font lbls = new Font("Comic Sans MS", Font. BOLD, 18);
1962
1963    //-----------------------------------------------------------------
1964    //Constructor for the Recycler Registration Screen that places components on the Frame
1965        public recyclerRegistration(String str)
1966        {
1967            super(str);
1968
1969            FR.setJMenuBar(MB);
1970            MB.add(file);
1971            file.add(logOut);
1972            file.add(Exit);
1973
1974            getContentPane().setLayout(GBL);
1975            FR.add(Obj1);
1976
1977            GBC.fill = GridBagConstraints.BOTH;
1978            GBC.anchor = GridBagConstraints.CENTER;
1979            GBC.gridwidth = 4;
1980            GBC.gridheight = 2;
1981            GBC.gridy = 1;
1982            GBC.gridx = 1;
1983            GBC.fill = GridBagConstraints.VERTICAL;
1984            GBC.ipady = 10;
1985            GBC.insets = new Insets(10,10,10,10);
1986            GBL.setConstraints(lblRecyRegist,GBC);
1987            lblRecyRegist.setFont(f);
1988            lblRecyRegist.setHorizontalAlignment(JLabel.CENTER);
1989            lblRecyRegist.setForeground(Color.white);
1990            getContentPane().add(lblRecyRegist);
1991
1992            GBC.gridheight = 1;
1993            GBC.gridwidth = 1;
1994            GBC.gridy = 5;
1995            lblRecyName.setFont(lbls);
1996            GBL.setConstraints(lblRecyName, GBC);
1997            lblRecyName.setForeground(Color.white);
1998            getContentPane().add(lblRecyName);
1999
2000            GBC.gridy = 7;
2001            lblForm.setFont(lbls);
2002            GBL.setConstraints(lblForm, GBC);
2003            lblForm.setForeground(Color.white);
2004            getContentPane().add(lblForm);
2005
2006            GBC.gridheight = 2;
2007            GBC.gridy = 9;
```

```
2008            GBC.ipady = 30;
2009            GBC.ipadx = 150;
2010            GBL.setConstraints(btnRegister,GBC);
2011            getContentPane().add(btnRegister);
2012
2013            GBC.gridheight = 1;
2014            GBC.gridwidth = 2;
2015            GBC.gridx = 3;
2016            GBC.gridy = 5;
2017            GBC.ipady = 0;
2018            GBC.ipadx = 0;
2019            lblRecyName.setLabelFor(txtRecyName);
2020            GBL.setConstraints(txtRecyName, GBC);
2021            getContentPane().add(txtRecyName);
2022
2023            GBC.gridy = 7;
2024            GBC.gridwidth = 2;
2025            lblForm.setLabelFor(chForm);
2026            addFormClasses();
2027            GBL.setConstraints(chForm, GBC);
2028            getContentPane().add(chForm);
2029
2030            GBC.gridheight = 2;
2031            GBC.gridy = 9;
2032            GBC.ipady = 30;
2033            GBC.ipadx = 100;
2034            GBL.setConstraints(btnCancel,GBC);
2035            getContentPane().add(btnCancel);
2036
2037            GBC.gridy = 11;
2038            GBC.gridx = 2;
2039            GBL.setConstraints(btnDelRecy, GBC);
2040            getContentPane().add(btnDelRecy);
2041
2042            getContentPane().setBackground(c);
2043            FR.setExtendedState(Frame.MAXIMIZED_BOTH);
2044
2045            btnRegister.addActionListener(this);
2046            btnCancel.addActionListener(this);
2047            btnDelRecy.addActionListener(this);
2048            logOut.addActionListener(this);
2049            Exit.addActionListener(this);
2050            validate();
2051        }
2052    //-----------------------------------------------------------------
2053    //This method reads the random access file and retrieves the names of form classes.
2054    //They are then added to a choice box.
2055        private void addFormClasses()
2056        {
2057            try
2058            {
2059                RandomAccessFile RAF = new RandomAccessFile("FormClassroomLocation.txt", "r");
2060                int recordSize = 13;            //Each record in this random access file takes up 13 characters
2061                int records = (int) RAF.length()/recordSize;    //Calculates the number of records in the RAF
2062
2063                for(int i = 0; i < records; i++)
2064                {
2065                    String line = "";
2066                    RAF.seek(i * recordSize);   //File pointer goes to the beginning of each record
```

```
2067
2068                    for(int d = 0; d < 8; d++)
2069                    {
2070                        byte b = RAF.readByte();
2071                        line += (char) b;
2072                    }
2073                chForm.addItem(line.trim());        //Adds form class to choice box
2074                }
2075            }
2076            catch(Exception e)
2077            {
2078                Toolkit.getDefaultToolkit().beep();
2079                JOptionPane.showMessageDialog(this, "An error has occured. Please contact Harris Rasheed to deal with t
    issue\nError Code: " + e,"Error Message", JOptionPane.ERROR_MESSAGE);    //Output any errors caught
2080            }
2081        }
2082    //-----------------------------------------------------------------
2083    //This method is used to register a brand new recycler to the system.
2084    //The first parameter is their name and the second parameter is the form
2085    //class that they are in.
2086        private void registerRecycler(String rName, String formClass)
2087        {
2088            try
2089            {
2090                RandomAccessFile RAF = new RandomAccessFile("RecyclerAttendanceStats.txt","rw");
2091                int recordSize = 40;
2092                int records = (int) RAF.length()/recordSize;
2093
2094                for(int i = 0; i < records; i++)
2095                {
2096                    String line = "";           //Variable stores each record
2097                    RAF.seek(i * recordSize);   //File pointer looks at the beginning of each record
2098
2099                    for(int d = 0; d < 40; d++)     //Reads each record of the array
2100                    {
2101                        byte b = RAF.readByte();
2102                        line += (char) b;
2103                    }
2104
2105                    if(line.substring(0,30).trim().equals(rName))       //Tests if the first field matches the existing
    recycler names
2106                    {
2107                        if(line.substring(30,38).trim().equals(formClass))  //Tests if the form class is also the same
2108                        {
2109                            JOptionPane.showMessageDialog(this, "This recycler already exists.", "Error",
    JOptionPane.ERROR_MESSAGE);
2110                            return;     //Returns void and gives an error message because the recycler already exists o
    file
2111                        }
2112                    }
2113                }
2114
2115                for(int a = rName.length(); a < 30; a++)        //Sets length of String to 30 characters
2116                {
2117                    rName += " ";
2118                }
2119
2120                for(int a = formClass.length(); a < 8; a++)     //Sets length of String to 8 characters
2121                {
```

```
2122                    formClass += " ";
2123                }
2124
2125                String record = rName + formClass + "0 ";   //Adds the recycler attendance tally of zero for the new
    recycler
2126                RAF.seek(RAF.length());                     //File pointer goes to the end of the file
2127                RAF.writeBytes(record);                     //Writes the record
2128                JOptionPane.showMessageDialog(this, "Recycler Registered","Registration Successful",
    JOptionPane.INFORMATION_MESSAGE);
2129                RAF.close();
2130            }
2131            catch(FileNotFoundException e)
2132            {
2133                Toolkit.getDefaultToolkit().beep();
2134                JOptionPane.showMessageDialog(this, "The RecyclerAttendanceStats.txt notepad file is missing from the
    current directory. This process cannot function without this file.\nError Code: " + e,"File is Missing!",
    JOptionPane.ERROR_MESSAGE); //Output error if a file is not found
2135            }
2136            catch(Exception e)
2137            {
2138                Toolkit.getDefaultToolkit().beep();
2139                JOptionPane.showMessageDialog(this, "An error has occured. Please contact Harris Rasheed to deal with t
    issue\nError Code: " + e,"Error Message", JOptionPane.ERROR_MESSAGE);    //Output any errors caught
2140            }
2141        }
2142    //-----------------------------------------------------------------
2143    //Deletes recycler name from the system. The first parameter is the
2144    //recycler's name that has been selected by the user.
2145        private void deleteRecycler(String rName)
2146        {
2147            try
2148            {
2149                File file = new File("RecyclerAttendanceStats.txt");
2150                RandomAccessFile RAF = new RandomAccessFile(file, "rw");
2151                int recordSize = 40;
2152                int records = (int)(RAF.length())/recordSize;
2153
2154                for(int i = 0; i < records; i++)
2155                {
2156                    String line = "";               //Variable stores each record
2157                    RAF.seek(i * recordSize);       //File pointer goes to the beginning of each line
2158
2159                    for(int ct = 0; ct < recordSize; ct++)  //Reads each record
2160                    {
2161                        byte b = RAF.readByte();
2162                        line += (char) b;
2163                    }
2164
2165                    String recyName = (line.substring(0,30)).trim();
2166
2167                    if(recyName.equals(rName))          //Checks if the String is equal to the delete parameter
2168                    {
2169                        RAF.seek((records - 1) * (recordSize));                 //File pointer looks at the last record
2170                        byte[] ba = new byte[recordSize];                       //Creates array with the size of one re
2171                        RAF.readFully(ba);                                      //Reads entire line and places it in th
    array
2172                        RAF.seek(i * recordSize);                               //File pointer looks at the place where
    record was found
2173                        RAF.write(ba);                                          //Overwrites the record location with t
```

```
2173    last record since it is to be deleted
2174                        RAF.setLength(((records - 1)* (recordSize)));           //Truncates file and removes the last r
    from the end of the file which has been moved to the deleted record's space
2175                        break;
2176                    }
2177                }
2178                RAF.close();
2179            }
2180            catch(FileNotFoundException e)
2181            {
2182                Toolkit.getDefaultToolkit().beep();
2183                JOptionPane.showMessageDialog(this, "The RecyclerAttendanceStats.txt notepad file is missing from the
    current directory. This process cannot function without this file.\nError Code: " + e,"File is Missing!",
    JOptionPane.ERROR_MESSAGE); //Output error if a file is not found
2184            }
2185            catch(Exception e)
2186            {
2187                Toolkit.getDefaultToolkit().beep();
2188                JOptionPane.showMessageDialog(this, "An error has occured. Please contact Harris Rasheed to deal with t
    issue\nError Code: " + e,"Error Message", JOptionPane.ERROR_MESSAGE);    //Output any errors caught
2189            }
2190        }
2191    //-----------------------------------------------------------------
2192    //This method is used to execute the appropriate method when the user performs an action event
2193        public void actionPerformed(ActionEvent ae)
2194        {
2195            if(ae.getSource()==btnRegister)
2196            {
2197                String recyName = txtRecyName.getText().trim();     //Reads the recycler name input
2198                String form = chForm.getSelectedItem().trim();      //Reads the form class selected
2199                registerRecycler(recyName, form);                   //Passes above variables to the registerRecycler me
2200            }
2201            if(ae.getSource()==btnDelRecy)
2202            {
2203                lunchCollRounds LCR = new lunchCollRounds("");
2204                Frame frame = new Frame();
2205                String delRecycler = (String) JOptionPane.showInputDialog(frame, "Please select the recycler you want t
    delete", "Recycler Deletion", JOptionPane.PLAIN_MESSAGE, null, LCR.addRecyclers(),"<Recycler Name>");
2206                deleteRecycler(delRecycler);
2207            }
2208            if(ae.getSource()==btnCancel)
2209            {
2210                FR.setVisible(false);
2211                menuPage MP = new menuPage("");
2212                MP.FR.setVisible(true);
2213            }
2214            if(ae.getSource()==logOut)
2215            {
2216                FR.setVisible(false);
2217                Program EP = new Program("");
2218                EP.FR.setVisible(true);
2219            }
2220            if(ae.getSource()==Exit)
2221            {
2222                System.exit(0);
2223            }
2224        }
2225    }
2226    //-----------------------------------------------------------------------------------------------------------------
```

```
2227   //Recycler of the Month Criterion Screen; This is the screen where the user specify
2228   //a criterion for a month and year for the system to produce a report of viable candidates
2229   //and their attendance statistics during the specified criterion.
2230   class RoMCriterion extends JFrame implements ActionListener
2231   {
2232       JFrame FR = new JFrame("Recycling Activity Monitoring System - Recycler of the Month Criterion");
2233       Container Obj1 = getContentPane();
2234       GridBagLayout GBL = new GridBagLayout();
2235       GridBagConstraints GBC = new GridBagConstraints();
2236       JMenuBar MB = new JMenuBar();
2237       JMenu file = new JMenu("File");
2238       JMenuItem logOut = new JMenuItem("Log out");
2239       JMenuItem Exit = new JMenuItem("Exit");
2240       JButton btnFind = new JButton("Find");
2241       JButton btnCancel = new JButton("Cancel");
2242       JButton btnBack = new JButton("Back");
2243       JLabel lblMonth = new JLabel("Month: ");
2244       JLabel lblYear = new JLabel("Year: ");
2245       JLabel lblRecycMonCrit = new JLabel("Recycler of the Month Criterion");
2246       Choice chMonth = new Choice();
2247       Choice chYear = new Choice();
2248       Color c = new Color(6,69,1);
2249       Font f = new Font("Comic Sans MS", Font.BOLD, 22);
2250       Font lbls = new Font("Comic Sans MS", Font. BOLD, 18);
2251
2252   //---------------------------------------------------------------------
2253   //Constructor for the Recycler of the Month Criterion Screen that places components on the Frame
2254       public RoMCriterion(String str)
2255       {
2256           super(str);
2257
2258           FR.setJMenuBar(MB);
2259           MB.add(file);
2260           file.add(logOut);
2261           file.add(Exit);
2262
2263           getContentPane().setLayout(GBL);
2264           FR.add(Obj1);
2265
2266           GBC.fill = GridBagConstraints.BOTH;
2267           GBC.anchor = GridBagConstraints.CENTER;
2268           GBC.gridwidth = 4;
2269           GBC.gridheight = 2;
2270           GBC.gridy = 1;
2271           GBC.gridx = 1;
2272           GBC.fill = GridBagConstraints.VERTICAL;
2273           GBC.ipady = 50;
2274           GBL.setConstraints(lblRecycMonCrit,GBC);
2275           lblRecycMonCrit.setFont(f);
2276           lblRecycMonCrit.setHorizontalAlignment(JLabel.CENTER);
2277           lblRecycMonCrit.setForeground(Color.white);
2278           getContentPane().add(lblRecycMonCrit);
2279
2280           GBC.gridheight = 1;
2281           GBC.gridwidth = 2;
2282           GBC.gridy = 5;
2283           lblMonth.setFont(lbls);
2284           GBL.setConstraints(lblMonth, GBC);
2285           lblMonth.setForeground(Color.white);
```

```
2286            getContentPane().add(lblMonth);
2287
2288            GBC.gridy = 7;
2289            lblYear.setFont(lbls);
2290            GBL.setConstraints(lblYear, GBC);
2291            lblYear.setForeground(Color.white);
2292            getContentPane().add(lblYear);
2293
2294            GBC.gridheight = 2;
2295            GBC.gridy = 9;
2296            GBC.ipady = 30;
2297            GBC.ipadx = 150;
2298            GBL.setConstraints(btnFind,GBC);
2299            getContentPane().add(btnFind);
2300
2301            GBC.gridheight = 1;
2302            GBC.gridx = 3;
2303            GBC.gridy = 5;
2304            GBC.ipady = 0;
2305            GBC.ipadx = 100;
2306            lblMonth.setLabelFor(chMonth);
2307            GBL.setConstraints(chMonth, GBC);
2308            chMonth.addItem("January");
2309            chMonth.addItem("February");
2310            chMonth.addItem("March");
2311            chMonth.addItem("April");
2312            chMonth.addItem("May");
2313            chMonth.addItem("June");
2314            chMonth.addItem("July");
2315            chMonth.addItem("August");
2316            chMonth.addItem("September");
2317            chMonth.addItem("October");
2318            chMonth.addItem("November");
2319            chMonth.addItem("December");
2320            getContentPane().add(chMonth);
2321
2322            GBC.gridy = 7;
2323            lblYear.setLabelFor(chYear);
2324            GBL.setConstraints(chYear, GBC);
2325            chYear.addItem("2010");
2326            chYear.addItem("2011");
2327            chYear.addItem("2012");
2328            chYear.addItem("2013");
2329            chYear.addItem("2014");
2330            chYear.addItem("2015");
2331            chYear.addItem("2016");
2332            chYear.addItem("2017");
2333            chYear.addItem("2018");
2334            chYear.addItem("2019");
2335            chYear.addItem("2020");
2336            getContentPane().add(chYear);
2337
2338            GBC.gridheight = 2;
2339            GBC.gridy = 9;
2340            GBC.ipady = 30;
2341            GBC.ipadx = 100;
2342            GBL.setConstraints(btnCancel,GBC);
2343            getContentPane().add(btnCancel);
2344
```

```
2345            getContentPane().setBackground(c);
2346            FR.setExtendedState(Frame.MAXIMIZED_BOTH);
2347
2348            btnFind.addActionListener(this);
2349            btnCancel.addActionListener(this);
2350            logOut.addActionListener(this);
2351            Exit.addActionListener(this);
2352            btnBack.addActionListener(this);
2353            validate();
2354        }
2355    //-----------------------------------------------------------------
2356    //This method is used to execute the appropriate method when the user performs an action event
2357        public void actionPerformed(ActionEvent ae)
2358        {
2359            if(ae.getSource()==btnFind)
2360            {
2361                String month = chMonth.getSelectedItem();
2362                String year = chYear.getSelectedItem();
2363                FR.setVisible(false);
2364                RoMCandidateReport RoMCR = new RoMCandidateReport(month, year);
2365            }
2366            if(ae.getSource()==btnCancel)
2367            {
2368                FR.setVisible(false);
2369                menuPage MP = new menuPage("");
2370                MP.FR.setVisible(true);
2371            }
2372            if(ae.getSource()==logOut)
2373            {
2374                FR.setVisible(false);
2375                Program EP = new Program("");
2376                EP.FR.setVisible(true);
2377            }
2378            if(ae.getSource()==Exit)
2379            {
2380                System.exit(0);
2381            }
2382        }
2383    }
2384    //-------------------------------------------------------------------------------------------------------------------
2385    //Recycler of the Month Candidates Report Screen; This is the screen where the user can view
2386    //the recyclers that have been shortlisted for the Recycler of the Month award according to the
2387    //time criterion of the user.
2388    class RoMCandidateReport extends JFrame implements ActionListener
2389    {
2390        JFrame FR = new JFrame("Recycling Activity Monitoring System - Recycler of the Month Candidate Report");
2391        Container Obj1 = getContentPane();
2392        GridBagLayout GBL = new GridBagLayout();
2393        GridBagConstraints GBC = new GridBagConstraints();
2394        JMenuBar MB = new JMenuBar();
2395        JMenu file = new JMenu("File");
2396        JMenuItem logOut = new JMenuItem("Log out");
2397        JMenuItem exit = new JMenuItem("Exit");
2398        JButton print = new JButton("Print");
2399        JButton cancel = new JButton("Cancel");
2400        JLabel lblRoMCandidateReport = new JLabel("Recycler of the Month Candidate Report");
2401        Color c = new Color(6,69,1);
2402        Font f = new Font("Comic Sans MS", Font.BOLD, 26);
2403        String[] colNames = {"Recycler Name", "Form Class", "Attendance Tally"};
```

```
2404        JTable table;
2405
2406    //-----------------------------------------------------------------
2407        public RoMCandidateReport(String month, String year)
2408        {
2409            table = new JTable(loadTable(month, year), colNames);
2410            JScrollPane scroll = new JScrollPane(table);
2411            FR.setJMenuBar(MB);
2412            MB.add(file);
2413            file.add(logOut);
2414            file.add(exit);
2415
2416            getContentPane().setLayout(GBL);
2417            FR.add(Obj1);
2418
2419            GBC.fill = GridBagConstraints.BOTH;
2420            GBC.anchor = GridBagConstraints.CENTER;
2421            GBC.gridwidth = 2;
2422            GBC.gridy = 1;
2423            GBC.gridx = 1;
2424            GBC.insets = new Insets(10,10,10,10);
2425            GBL.setConstraints(lblRoMCandidateReport,GBC);
2426            lblRoMCandidateReport.setFont(f);
2427            lblRoMCandidateReport.setHorizontalAlignment(JLabel.CENTER);
2428            lblRoMCandidateReport.setForeground(Color.white);
2429            getContentPane().add(lblRoMCandidateReport);
2430
2431            GBC.gridy = 4;
2432            GBC.gridx = 1;
2433            GBC.gridwidth = 2;
2434            GBL.setConstraints(scroll,GBC);
2435            getContentPane().add(scroll);
2436
2437            GBC.gridy = 10;
2438            GBC.ipady = 20;
2439            GBC.ipadx = 100;
2440            GBL.setConstraints(print,GBC);
2441            getContentPane().add(print);
2442
2443            GBC.gridy = 12;
2444            GBL.setConstraints(cancel,GBC);
2445            getContentPane().add(cancel);
2446
2447            getContentPane().setBackground(c);
2448            FR.setExtendedState(Frame.MAXIMIZED_BOTH);
2449            FR.setVisible(true);
2450
2451            print.addActionListener(this);
2452            exit.addActionListener(this);
2453            logOut.addActionListener(this);
2454            cancel.addActionListener(this);
2455            validate();
2456        }
2457    //-----------------------------------------------------------------
2458    //This method loads the JTable when the class is started up and fills it with values
2459    //The first parameter is the month and the second parameter is the year. These dates
2460    //are the criterion to find the recyclers who attended in that year and month so that
2461    //they can be displayed in the report as candidates for the recycler of the month award
2462        private String[][] loadTable(String month, String year)
```

```
2463        {
2464            try
2465            {
2466                int recordSize = 40;                //The size of each record is 40 characters long
2467                RandomAccessFile raf = new RandomAccessFile("RecyclerAttendanceLog.txt", "r");
2468                int records = (int) raf.length()/recordSize;        //Calculates the number of records exist in the fil
2469                String[][] logData = new String[records][3];        //Creates an array to store the data from the rando
    access log file
2470
2471                for(int c = 0; c < records; c++)                //First loop iterates each array index
2472                {
2473                    raf.seek(c * recordSize);           //File pointer goes to the start of each record on evert iterat
2474                    String currentLine = "";            //currentLine is used to store each record being searched
2475                    for(int i = 0; i < recordSize; i++)
2476                    {
2477                        byte b = raf.readByte();        //Reads each character and stores it as a byte
2478                        currentLine += (char) b;        //Converts each character from byte to character and then to st
    and collects each character on each loop
2479                    }
2480
2481                    String recycName = (currentLine.substring(0,30)).trim();    //Reads the first field
2482                    String attMonth = (currentLine.substring(33,35)).trim();    //Reads the second field
2483                    String attYear = (currentLine.substring(36,40)).trim();     //Reads the third field
2484
2485                    if(attMonth.substring(0,1).equals("0"))     //Checks if the month starts with a 0 e.g. 07 is July
2486                    {
2487                        attMonth = attMonth.substring(1,2);     //Takes the main number if it starts with zero
2488                    }
2489
2490                    int attMon = Integer.parseInt(attMonth);    //Converts String to integer
2491
2492                    switch(attMon)              //Switch case changes the month number to month name
2493                    {
2494                        case 1:
2495                            attMonth = "January";
2496                            break;
2497
2498                        case 2:
2499                            attMonth = "February";
2500                            break;
2501
2502                        case 3:
2503                            attMonth = "March";
2504                            break;
2505
2506                        case 4:
2507                            attMonth = "April";
2508                            break;
2509
2510                        case 5:
2511                            attMonth = "May";
2512                            break;
2513
2514                        case 6:
2515                            attMonth = "June";
2516                            break;
2517
2518                        case 7:
2519                            attMonth = "July";
```

```
2520                            break;
2521
2522                        case 8:
2523                            attMonth = "August";
2524                            break;
2525
2526                        case 9:
2527                            attMonth = "September";
2528                            break;
2529
2530                        case 10:
2531                            attMonth = "October";
2532                            break;
2533
2534                        case 11:
2535                            attMonth = "November";
2536                            break;
2537
2538                        case 12:
2539                            attMonth = "December";
2540                            break;
2541
2542                    }
2543                    logData[c][0] = recycName;      //Writes record details to array
2544                    logData[c][1] = attMonth;
2545                    logData[c][2] = attYear;
2546                }
2547                raf.close();
2548                return candidateSelection(logData, records, month, year);   //returns candidateSelection method to filt
    out recyclers who attended in different months
2549            }
2550            catch(FileNotFoundException e)
2551            {
2552                Toolkit.getDefaultToolkit().beep();
2553                JOptionPane.showMessageDialog(this, "The RecyclerAttendanceLog.txt notepad file is missing from the cur
    directory. This process cannot function without this file.\nError Code: " + e,"File is Missing!",
    JOptionPane.ERROR_MESSAGE);   //Output any error if a file is not found
2554            }
2555            catch(Exception e)
2556            {
2557                Toolkit.getDefaultToolkit().beep();
2558                JOptionPane.showMessageDialog(this, "An error has occured. Please contact Harris Rasheed to deal with t
    issue\nError Code: " + e,"Error Message", JOptionPane.ERROR_MESSAGE);    //Output any errors caught
2559            }
2560            return null;
2561        }
2562    //-----------------------------------------------------------------
2563    //This method is used to filter out all the recycler names who did not attend in the month and
2564    //year specified by the criterion. The first parameter is an array of all the recycler attendance
2565    //stored on file, the second parameter is the number of rows in the array, the third and fourth
2566    //parameter is the month and year criterion.
2567        private String[][] candidateSelection(String[][] logData, int records, String month, String year)
2568        {
2569            try
2570            {
2571                RandomAccessFile RAF = new RandomAccessFile("RecyclerAttendanceStats.txt", "r");
2572                int recordSize = 40;
2573                int rowCount = (int) RAF.length()/recordSize;       //Calculates the number of recyclers registered
2574                int pendingRowCounter = 0;              //Used to keep track of the location of the last record in the
```

```
2575                RAF.close();
2576                String[][] currentMonthAttendance = new String[rowCount][3];        //Creates array for the registered
    recyclers
2577
2578                for(int i = 0; i < records; i++)        //Loops and sets the 3rd column values to zero
2579                {
2580                    currentMonthAttendance[i][2] = Integer.toString(0);
2581                }
2582
2583                for(int c = 0; c < records; c++)
2584                {
2585                    String current = logData[c][0];     //Reads current record first field
2586                    int counter = 0;                    //Used to keep track of a double occurred record so that its
    attendance can be iterated
2587                    boolean doubleOccurence = false;    //Flag used to check if a recycler has participated more than o
2588
2589                    for(counter = 0; counter < records; counter++)
2590                    {
2591                        if(current.equals(currentMonthAttendance[counter][0]))  //Checks if the current record of the l
    file matches an existing recycler attendance
2592                        {
2593                            doubleOccurence = true;     //The flag is activated marking the double occurence
2594                            break;
2595                        }
2596                    }
2597
2598                    if(!doubleOccurence)        //Checks if the current record is a new recycler to the table
2599                    {
2600                        if(logData[c][1].equals(month) && logData[c][2].equals(year))       //Checks if the recycler's
    attendance matches the criterion specified by the user
2601                        {
2602                            currentMonthAttendance[pendingRowCounter][0] = logData[c][0];           //Writes recycler n
    to the array
2603                            currentMonthAttendance[pendingRowCounter][2] = Integer.toString(1);     //Sets the recycler
    attendance to 1
2604                            pendingRowCounter++;    //Adds to row counter
2605                        }
2606                    }
2607                    else
2608                    {
2609                        currentMonthAttendance[counter][2] =
    Integer.toString(Integer.parseInt(currentMonthAttendance[counter][2]) + 1);  //Adds to recycler attendance if the n
    already exists in the array
2610                    }
2611                }
2612                return referenceFormClass(currentMonthAttendance, records);     //Returns filtered array
2613            }
2614            catch(Exception e)
2615            {
2616                Toolkit.getDefaultToolkit().beep();
2617                JOptionPane.showMessageDialog(this, "An error has occured. Please contact Harris Rasheed to deal with t
    issue\nError Code: " + e,"Error Message", JOptionPane.ERROR_MESSAGE);    //Output any errors caught
2618            }
2619            return null;
2620        }
2621    //-----------------------------------------------------------------
2622    //This method is used to reference the recycler names with their form classes.
2623    //The first parameter is the array containing the filtered recycling record.
2624    //The second parameter is the number of rows in the array.
```

```
2625        private String[][] referenceFormClass(String[][] tableData, int rows)
2626        {
2627            try
2628            {
2629                RandomAccessFile RAF = new RandomAccessFile("RecyclerAttendanceStats.txt", "r");
2630
2631                for(int c = 0; c < rows; c++)
2632                {
2633                    int recordSize = 40;                                //Set length of each record in the random access fi
2634                    int records = (int) RAF.length()/recordSize;    //Calculates the number of records in the random ac
    file
2635                    String currentLine = "";                            //currentLine is used to store each record of the f
2636
2637                    for(int i = 0; i < records; i++)
2638                    {
2639                        if(tableData[c][0] == null)         //Checks if there is a blank record in the array/table
2640                        {
2641                            break;                          //Terminates the loop if a record is blank
2642                        }
2643
2644                        String line = "";           //Variable that stores each record
2645                        RAF.seek(i * recordSize);   //File pointer looks at the start of each record
2646
2647                        for(int ct = 0; ct < recordSize; ct++)      //Loop reads each record
2648                        {
2649                            byte b = RAF.readByte();
2650                            line += (char)b;
2651                        }
2652
2653                        currentLine = (line.substring(0,30)).trim();                //Reads first record field
2654
2655                        if((tableData[c][0].trim()).equalsIgnoreCase(currentLine))  //Checks if the reference in the ta
    and file match
2656                        {
2657                            tableData[c][1] = (line.substring(30,38)).trim();       //Assigns reference file data to ta
    array
2658                            break;
2659                        }
2660                    }
2661                }
2662                RAF.close();
2663
2664                recyclingActivityReport RAR = new recyclingActivityReport("");
2665                RAR.bubbleSort(tableData, rows);    //Bubble sorts the array
2666                return Patch(tableData, rows);      //Removes empty rows
2667            }
2668            catch(FileNotFoundException e)
2669            {
2670                Toolkit.getDefaultToolkit().beep();
2671                JOptionPane.showMessageDialog(this, "The RecyclerAttendanceStats.txt notepad file is missing from the
    current directory. This process cannot function without this file.\nError Code: " + e,"File is Missing!",
    JOptionPane.ERROR_MESSAGE); //Output any error if a file is not found
2672            }
2673            catch(Exception e)
2674            {
2675                Toolkit.getDefaultToolkit().beep();
2676                JOptionPane.showMessageDialog(this, "An error has occured. Please contact Harris Rasheed to deal with t
    issue\nError Code: " + e,"Error Message", JOptionPane.ERROR_MESSAGE);    //Output any errors caught
2677            }
```

```
2678            return null;
2679        }
2680    //-----------------------------------------------------------------
2681    //This method removes the empty rows from the bottom of the 2D array.
2682        private String[][] Patch(String [][] tableData, int rows)
2683        {
2684            int active = 0;
2685            for(int i = 0; i < rows; i++)
2686            {
2687                if(tableData[i][0]!=null)   //Tests if the row is occupied with data
2688                {
2689                    active++;           //Adds one to the active row counter
2690                }
2691            }
2692
2693            String[][] patch = new String[active][3];       //Creates an array with the specific size of the relevant
    records
2694
2695            for(int i = 0; i < active; i++)
2696            {
2697                for(int d = 0; d < 3; d++)
2698                {
2699                    patch[i][d] = tableData[i][d];          //Writes relevant records to the new array
2700                }
2701            }
2702            return patch;   //Returns new filtered and sorted array
2703        }
2704    //-----------------------------------------------------------------
2705    //This method is used to execute the appropriate method when the user performs an action event
2706        public void actionPerformed(ActionEvent ae)
2707        {
2708            if(ae.getSource()==print)
2709            {
2710                try
2711                {
2712                    if(!table.print())      //Checks if the user has cancelled the print job
2713                    {
2714                        JOptionPane.showMessageDialog(this, "The user has cancelled the print job.", "Cancelled Print J
    JOptionPane.INFORMATION_MESSAGE);
2715                    }
2716                }
2717                catch (java.awt.print.PrinterException e)           //Catches printer exception
2718                {
2719                    JOptionPane.showMessageDialog(this, "Unable to print due to " + e, "Print Job Error",
    JOptionPane.ERROR_MESSAGE);
2720                    Toolkit.getDefaultToolkit().beep();
2721                }
2722            }
2723            if(ae.getSource()==cancel)
2724            {
2725                FR.setVisible(false);
2726                RoMCriterion RoMC = new RoMCriterion("");
2727                RoMC.FR.setVisible(true);
2728            }
2729            if(ae.getSource()==logOut)
2730            {
2731                FR.setVisible(false);
2732                Program EP = new Program("");
2733                EP.FR.setVisible(true);
```

```
2734                Toolkit.getDefaultToolkit().beep();
2735            }
2736            if(ae.getSource()==exit)
2737            {
2738                System.exit(0);
2739            }
2740        }
2741    }
2742    //-------------------------------------------------------------------------------------------------------------
2743    //Teachers & Classrooms Plan Screen; This is the screen where the reference table of the database
2744    //can be viewed. It also allows the user to update data in the table.
2745    class teacherClassPlan extends JFrame implements ActionListener
2746    {
2747        JFrame FR = new JFrame("Recycling Activity Monitoring System - Teachers & Classrooms Plan");
2748        Container Obj1 = getContentPane();
2749        GridBagLayout GBL = new GridBagLayout();
2750        GridBagConstraints GBC = new GridBagConstraints();
2751        JMenuBar MB = new JMenuBar();
2752        JMenu file = new JMenu("File");
2753        JMenuItem logOut = new JMenuItem("Log out");
2754        JMenuItem exit = new JMenuItem("Exit");
2755        JButton print = new JButton("Print");
2756        JButton save = new JButton("Save");
2757        JButton cancel = new JButton("Cancel");
2758        JLabel lblTeacherClass = new JLabel("Teachers & Classrooms Plan");
2759        Color c = new Color(6,69,1);
2760        Font f = new Font("Comic Sans MS", Font.BOLD, 26);
2761        String[] colNames = {"Room Number", "Teacher"};
2762        File RAF = new File("TeacherClassroomPlan.txt");
2763        JTable table = new JTable(loadTable(122, 5, 35, RAF), colNames);
2764        JScrollPane scroll = new JScrollPane(table);
2765
2766    //-----------------------------------------------------------------
2767    //Constructor for the Teachers & Classrooms Plan Screen that places components on the Frame
2768        public teacherClassPlan(String str)
2769        {
2770            super(str);
2771
2772            FR.setJMenuBar(MB);
2773            MB.add(file);
2774            file.add(logOut);
2775            file.add(exit);
2776
2777            getContentPane().setLayout(GBL);
2778            FR.add(Obj1);
2779
2780            GBC.fill = GridBagConstraints.BOTH;
2781            GBC.anchor = GridBagConstraints.CENTER;
2782            GBC.gridwidth = 2;
2783            GBC.gridy = 1;
2784            GBC.gridx = 1;
2785            GBC.insets = new Insets(10,10,10,10);
2786            GBL.setConstraints(lblTeacherClass,GBC);
2787            lblTeacherClass.setFont(f);
2788            lblTeacherClass.setHorizontalAlignment(JLabel.CENTER);
2789            lblTeacherClass.setForeground(Color.white);
2790            getContentPane().add(lblTeacherClass);
2791
2792            GBC.gridy = 4;
```

```
2793            GBC.gridx = 1;
2794            GBC.gridwidth = 2;
2795            GBL.setConstraints(scroll,GBC);
2796            getContentPane().add(scroll);
2797
2798            GBC.ipady = 20;
2799            GBC.ipadx = 100;
2800            GBC.gridy = 10;
2801            GBL.setConstraints(save,GBC);
2802            getContentPane().add(save);
2803
2804            GBC.gridy = 12;
2805
2806            GBL.setConstraints(print,GBC);
2807            getContentPane().add(print);
2808
2809            GBC.gridy = 14;
2810            GBL.setConstraints(cancel,GBC);
2811            getContentPane().add(cancel);
2812
2813            getContentPane().setBackground(c);
2814            FR.setExtendedState(Frame.MAXIMIZED_BOTH);
2815
2816            print.addActionListener(this);
2817            save.addActionListener(this);
2818            exit.addActionListener(this);
2819            logOut.addActionListener(this);
2820            cancel.addActionListener(this);
2821            validate();
2822        }
2823    //-----------------------------------------------------------------
2824    //This method loads JTable on initialisation of the Teachers & Classrooms Plan reference table
2825        protected String[][] loadTable(int rows, int firstField, int recordSize, File referenceFile)
2826        {
2827            try
2828            {
2829                String[][] tableData = new String[rows][2];                     //Creates array with size to hold data
2830                RandomAccessFile raf = new RandomAccessFile(referenceFile, "r");    //Creates an object of the Random A
      File
2831                int records = (int) raf.length()/recordSize;
2832
2833                for(int c = 0; c < records; c++)        //First loop iterates each array index
2834                {
2835                    raf.seek(c * recordSize);           //File pointer goes to the start of each record on evert iterat
2836                    String currentLine = "";            //currentLine is used to store each record being searched
2837                    for(int i = 0; i < recordSize; i++)
2838                    {
2839                        byte b = raf.readByte();
2840                        currentLine += (char) b;
2841                    }
2842
2843                    tableData[c][0] = (currentLine.substring(0,firstField)).trim();             //Assigns room number t
      array
2844                    tableData[c][1] = (currentLine.substring(firstField,recordSize)).trim();    //Assigns teacher name
      array
2845                }
2846                raf.close();
2847                return tableData;       //returns 2D array of Random Access file records
2848            }
```

```
2849            catch(FileNotFoundException e)
2850            {
2851                Toolkit.getDefaultToolkit().beep();
2852                JOptionPane.showMessageDialog(this, "An important system file is missing from the current directory. Th
    process cannot function without this file.\nError Code: " + e,"File is Missing!", JOptionPane.ERROR_MESSAGE); //Outp
    any error if a file is not found
2853            }
2854            catch(Exception e)
2855            {
2856                Toolkit.getDefaultToolkit().beep();
2857                JOptionPane.showMessageDialog(this, "An error has occured. Please contact Harris Rasheed to deal with t
    issue\nError Code: " + e,"Error Message", JOptionPane.ERROR_MESSAGE);    //Output any errors caught
2858            }
2859            return null;
2860        }
2861    //-----------------------------------------------------------------
2862    //This method is used to store the data from the table in the
2863    //random access file when the save button is clicked. The first
2864    //parameter is the current table's data.
2865        protected void saveData(String tableData[][], int firstFieldSize, int secondFieldSize, File file)
2866        {
2867            try
2868            {
2869                RandomAccessFile raf = new RandomAccessFile(file, "rw");
2870                int recordSize = firstFieldSize + secondFieldSize;
2871                int records = (int)(raf.length())/recordSize;
2872
2873                raf.seek(0);        //File pointer looks at the beginning of the file
2874
2875                for(int i = 0; i < records; i++)
2876                {
2877                    for(int c = tableData[i][0].length(); c < firstFieldSize; c++)  //Sets the length of the room numbe
    the array
2878                    {
2879                        tableData[i][0] += " ";
2880                    }
2881                    raf.writeBytes(tableData[i][0]);                //Writes the classroom number to the array
2882
2883                    for(int c = tableData[i][1].length(); c < secondFieldSize; c++) //Sets the length of the teacher na
    in the array
2884                    {
2885                        tableData[i][1] += " ";
2886                    }
2887                    raf.writeBytes(tableData[i][1]);                //Writes the teacher names to the array
2888                }
2889                raf.close();
2890                JOptionPane.showMessageDialog(this, "Your save is successful!","Save Successful!",
    JOptionPane.INFORMATION_MESSAGE);
2891            }
2892
2893            catch(FileNotFoundException e)
2894            {
2895                Toolkit.getDefaultToolkit().beep();
2896                JOptionPane.showMessageDialog(this, "An important system file is missing from the current directory. Th
    process cannot function without this file.\nError Code: " + e,"File is Missing!", JOptionPane.ERROR_MESSAGE); //Outp
    any error if a file is not found
2897            }
2898            catch(Exception e)
2899            {
```

```
2900                Toolkit.getDefaultToolkit().beep();
2901                JOptionPane.showMessageDialog(this, "An error has occured. Please contact Harris Rasheed to deal with t
    issue\nError Code: " + e,"Error Message", JOptionPane.ERROR_MESSAGE);    //Output any errors caught
2902            }
2903        }
2904    //-----------------------------------------------------------------
2905    //This method is used to execute the appropriate method when the user performs an action event
2906        public void actionPerformed(ActionEvent ae)
2907        {
2908            if(ae.getSource()==print)
2909            {
2910                try
2911                {
2912                    if(!table.print())      //Checks if the user has cancelled the print job
2913                    {
2914                        JOptionPane.showMessageDialog(this, "The user has cancelled the print job.", "Cancelled Print J
    JOptionPane.INFORMATION_MESSAGE);
2915                    }
2916                }
2917                catch(java.awt.print.PrinterException e)            //Catches printer exception
2918                {
2919                    JOptionPane.showMessageDialog(this, "Unable to print due to " + e, "Print Job Error",
    JOptionPane.ERROR_MESSAGE);
2920                    Toolkit.getDefaultToolkit().beep();
2921                }
2922            }
2923            if(ae.getSource()==save)
2924            {
2925                int rows = table.getRowCount();             //Retrieves number of rows
2926                int columns = table.getColumnCount();       //Retrieves number of columns
2927                String[][] tableData = new String[rows][columns];       //Creates array with the size of the JTable
2928
2929                for(int i = 0; i < rows; i++)
2930                {
2931                    for(int j = 0; j < columns; j++)
2932                    {
2933                        tableData[i][j] = (String) table.getValueAt(i,j); //Reads all the values of JTable and stores i
    a 2D array
2934                    }
2935                }
2936                saveData(tableData, 5, 30, RAF);        //Executes storeTeacherClass method when the Save button is pre
2937            }
2938            if(ae.getSource()==cancel)
2939            {
2940                FR.setVisible(false);
2941                menuPage MP = new menuPage("");
2942                MP.FR.setVisible(true);
2943            }
2944            if(ae.getSource()==logOut)
2945            {
2946                FR.setVisible(false);
2947                Program EP = new Program("");
2948                EP.FR.setVisible(true);
2949                Toolkit.getDefaultToolkit().beep();
2950            }
2951            if(ae.getSource()==exit)
2952            {
2953                System.exit(0);
2954            }
```

```
2955        }
2956    }
2957    //-------------------------------------------------------------------------------------------------------------------
2958    //Form Classroom Locations Screen; This is the screen where the user can view and update the
2959    //reference table that associates form classes with the room numbers that they are located in.
2960    class formClassroomLocation extends JFrame implements ActionListener
2961    {
2962        JFrame FR = new JFrame("Recycling Activity Monitoring System - Form Classroom Locations");
2963        Container Obj1 = getContentPane();
2964        GridBagLayout GBL = new GridBagLayout();
2965        GridBagConstraints GBC = new GridBagConstraints();
2966        JMenuBar MB = new JMenuBar();
2967        JMenu file = new JMenu("File");
2968        JMenuItem logOut = new JMenuItem("Log out");
2969        JMenuItem exit = new JMenuItem("Exit");
2970        JButton print = new JButton("Print");
2971        JButton save = new JButton("Save");
2972        JButton cancel = new JButton("Cancel");
2973        JLabel lblFormClass = new JLabel("Form Classroom Locations");
2974        Color c = new Color(6,69,1);
2975        Font f = new Font("Comic Sans MS", Font.BOLD, 26);
2976        String[] colNames = {"Form Class", "Room Number"};
2977        File RAF = new File("FormClassroomLocation.txt");
2978        teacherClassPlan TCP = new teacherClassPlan("");
2979        JTable table = new JTable(TCP.loadTable(38, 8, 13, RAF), colNames);
2980        JScrollPane scroll = new JScrollPane(table);
2981
2982    //-------------------------------------------------------------------
2983    //Constructor for the Form Classroom Locations Screen that places components on the Frame
2984        public formClassroomLocation(String str)
2985        {
2986            super(str);
2987
2988            FR.setJMenuBar(MB);
2989            MB.add(file);
2990            file.add(logOut);
2991            file.add(exit);
2992
2993            getContentPane().setLayout(GBL);
2994            FR.add(Obj1);
2995
2996            GBC.fill = GridBagConstraints.BOTH;
2997            GBC.anchor = GridBagConstraints.CENTER;
2998            GBC.gridwidth = 2;
2999            GBC.gridy = 1;
3000            GBC.gridx = 1;
3001            GBC.insets = new Insets(10,10,10,10);
3002            GBL.setConstraints(lblFormClass,GBC);
3003            lblFormClass.setFont(f);
3004            lblFormClass.setHorizontalAlignment(JLabel.CENTER);
3005            lblFormClass.setForeground(Color.white);
3006            getContentPane().add(lblFormClass);
3007
3008            GBC.gridy = 4;
3009            GBC.gridx = 1;
3010            GBC.gridwidth = 2;
3011            GBL.setConstraints(scroll,GBC);
3012            getContentPane().add(scroll);
3013
```

```
3014            GBC.ipady = 20;
3015            GBC.ipadx = 100;
3016            GBC.gridy = 10;
3017            GBL.setConstraints(save,GBC);
3018            getContentPane().add(save);
3019
3020            GBC.gridy = 12;
3021
3022            GBL.setConstraints(print,GBC);
3023            getContentPane().add(print);
3024
3025            GBC.gridy = 14;
3026            GBL.setConstraints(cancel,GBC);
3027            getContentPane().add(cancel);
3028
3029            getContentPane().setBackground(c);
3030            FR.setExtendedState(Frame.MAXIMIZED_BOTH);
3031
3032            print.addActionListener(this);
3033            save.addActionListener(this);
3034            exit.addActionListener(this);
3035            logOut.addActionListener(this);
3036            cancel.addActionListener(this);
3037            validate();
3038        }
3039    //-----------------------------------------------------------------
3040    //This method is used to execute the appropriate method when the user performs an action event
3041        public void actionPerformed(ActionEvent ae)
3042        {
3043            if(ae.getSource()==print)
3044            {
3045                try
3046                {
3047                    if(!table.print())      //Checks if the user has cancelled the print job
3048                    {
3049                        JOptionPane.showMessageDialog(this, "The user has cancelled the print job.", "Cancelled Print J
    JOptionPane.INFORMATION_MESSAGE);
3050                    }
3051                }
3052                catch(java.awt.print.PrinterException e)            //Catches printer exception
3053                {
3054                    JOptionPane.showMessageDialog(this, "Unable to print due to " + e, "Print Job Error",
    JOptionPane.ERROR_MESSAGE);
3055                    Toolkit.getDefaultToolkit().beep();
3056                }
3057            }
3058            if(ae.getSource()==save)
3059            {
3060                int rows = table.getRowCount();
3061                int columns = table.getColumnCount();
3062                String[][] tableData = new String[rows][columns];
3063
3064                for(int i = 0; i < rows; i++)
3065                {
3066                    for(int j = 0; j < columns; j++)
3067                    {
3068                        tableData[i][j] = (String) table.getValueAt(i,j);       //Stores JTable values into a 2D array
3069                    }
3070                }
```

```
3071                teacherClassPlan TCP = new teacherClassPlan("");
3072                TCP.saveData(tableData, 8, 5, RAF);
3073            }
3074            if(ae.getSource()==cancel)
3075            {
3076                FR.setVisible(false);
3077                menuPage MP = new menuPage("");
3078                MP.FR.setVisible(true);
3079            }
3080            if(ae.getSource()==logOut)
3081            {
3082                FR.setVisible(false);
3083                Program EP = new Program("");
3084                EP.FR.setVisible(true);
3085                Toolkit.getDefaultToolkit().beep();
3086            }
3087            if(ae.getSource()==exit)
3088            {
3089                System.exit(0);
3090            }
3091        }
3092    }
3093    //-------------------------------------------------------------------------------------------------------------
3094    //Security Settings Screen; This is the screen where the user has the option to change the password
3095    //or assign a new secret question.
3096    class securitySett extends JFrame implements ActionListener
3097    {
3098        JFrame FR = new JFrame("Recycling Activity Monitoring System - Security Settings");
3099        Container Obj1 = getContentPane();
3100        GridBagLayout GBL = new GridBagLayout();
3101        GridBagConstraints GBC = new GridBagConstraints();
3102        JMenuBar MB = new JMenuBar();
3103        JMenu file = new JMenu("File");
3104        JMenuItem logOut = new JMenuItem("Log out");
3105        JMenuItem Exit = new JMenuItem("Exit");
3106        JButton changePassBtn = new JButton("Change Password");
3107        JButton secretQtionBtn = new JButton("Assign Secret Question");
3108        JButton btnBack = new JButton("Back");
3109        JLabel lblSecuritySettings = new JLabel("Security Settings");
3110        Color c = new Color(6,69,1);
3111        Font f = new Font("Comic Sans MS", Font.BOLD, 22);
3112
3113    //-----------------------------------------------------------------
3114    //Constructor for the Security Settings Screen that places components on the Frame
3115        public securitySett(String str)
3116        {
3117            super(str);
3118
3119            FR.setJMenuBar(MB);
3120            MB.add(file);
3121            file.add(logOut);
3122            file.add(Exit);
3123
3124            getContentPane().setLayout(GBL);
3125            FR.add(Obj1);
3126
3127            GBC.fill = GridBagConstraints.BOTH;
3128            GBC.anchor = GridBagConstraints.PAGE_START;
3129            GBC.gridwidth = 2;
```

```
3130            GBC.gridheight = 2;
3131            GBC.gridy = 1;
3132            GBC.gridx = 1;
3133            GBC.insets = new Insets(10,10,10,10);
3134            GBC.fill = GridBagConstraints.VERTICAL;
3135            GBL.setConstraints(lblSecuritySettings,GBC);
3136            lblSecuritySettings.setFont(f);
3137            lblSecuritySettings.setHorizontalAlignment(JLabel.CENTER);
3138            lblSecuritySettings.setForeground(Color.white);
3139            getContentPane().add(lblSecuritySettings);
3140
3141            GBC.anchor = GridBagConstraints.CENTER;
3142            GBC.gridy = 3;
3143            GBC.ipady = 20;
3144            GBC.ipadx = 100;
3145            GBL.setConstraints(changePassBtn,GBC);
3146            getContentPane().add(changePassBtn);
3147
3148            GBC.gridy = 5;
3149            GBL.setConstraints(secretQtionBtn,GBC);
3150            getContentPane().add(secretQtionBtn);
3151
3152            GBC.gridy = 20;
3153            GBC.anchor = GridBagConstraints.PAGE_END;
3154            GBC.insets = new Insets(250,10,10,10);
3155            GBL.setConstraints(btnBack,GBC);
3156            getContentPane().add(btnBack);
3157
3158            getContentPane().setBackground(c);
3159            FR.setExtendedState(Frame.MAXIMIZED_BOTH);
3160
3161            changePassBtn.addActionListener(this);
3162            secretQtionBtn.addActionListener(this);
3163            logOut.addActionListener(this);
3164            Exit.addActionListener(this);
3165            btnBack.addActionListener(this);
3166            validate();
3167        }
3168    //------------------------------------------------------------------
3169    //This method is used to execute the appropriate method when the user performs an action event
3170        public void actionPerformed(ActionEvent ae)
3171        {
3172            if(ae.getSource()==changePassBtn)
3173            {
3174                FR.setVisible(false);
3175                changePass CP = new changePass("");
3176                CP.FR.setVisible(true);
3177            }
3178            if(ae.getSource()==secretQtionBtn)
3179            {
3180                FR.setVisible(false);
3181                secretQtion SQ = new secretQtion("");
3182                SQ.FR.setVisible(true);
3183            }
3184            if(ae.getSource()==btnBack)
3185            {
3186                FR.setVisible(false);
3187                menuPage MP = new menuPage("");
3188                MP.FR.setVisible(true);
```

```
3189            }
3190            if(ae.getSource()==logOut)
3191            {
3192                FR.setVisible(false);
3193                Program EP = new Program("");
3194                EP.FR.setVisible(true);
3195            }
3196            if(ae.getSource()==Exit)
3197            {
3198                System.exit(0);
3199            }
3200        }
3201    }
3202    //-------------------------------------------------------------------------------------------------------------------
3203    //Change Password Screen; This screen allows the user to change the system password.
3204    class changePass extends JFrame implements ActionListener
3205    {
3206        JFrame FR = new JFrame("Recycling Activity Monitoring System - Change Password");
3207        Container Obj1 = getContentPane();
3208        GridBagLayout GBL = new GridBagLayout();
3209        GridBagConstraints GBC = new GridBagConstraints();
3210        JMenuBar MB = new JMenuBar();
3211        JMenu file = new JMenu("File");
3212        JMenuItem logOut = new JMenuItem("Log out");
3213        JMenuItem Exit = new JMenuItem("Exit");
3214        JButton btnSubmit = new JButton("Submit");
3215        JButton btnCancel = new JButton("Cancel");
3216        JButton btnBack = new JButton("Back");
3217        JLabel lblCurrentPass = new JLabel("Current Password: ");
3218        JLabel lblNewPass = new JLabel("New Password: ");
3219        JLabel lblNewConfPass = new JLabel("Confirm New Password: ");
3220        JLabel lblChangePassword = new JLabel("Change Password");
3221        JPasswordField currentPass = new JPasswordField(20);
3222        JPasswordField newPass = new JPasswordField(20);
3223        JPasswordField newConfPass = new JPasswordField(20);
3224        Color c = new Color(6,69,1);
3225        Font f = new Font("Comic Sans MS", Font.BOLD, 22);
3226
3227    //-----------------------------------------------------------------
3228    //Constructor for the Change Password Screen that places components on the Frame
3229        public changePass(String str)
3230        {
3231            super(str);
3232
3233            FR.setJMenuBar(MB);
3234            MB.add(file);
3235            file.add(logOut);
3236            file.add(Exit);
3237
3238            getContentPane().setLayout(GBL);
3239            FR.add(Obj1);
3240
3241            GBC.fill = GridBagConstraints.BOTH;
3242            GBC.anchor = GridBagConstraints.PAGE_START;
3243            GBC.gridwidth = 4;
3244            GBC.gridheight = 2;
3245            GBC.gridy = 1;
3246            GBC.gridx = 1;
3247            GBC.insets = new Insets(10,10,10,10);
```

```
3248            GBC.fill = GridBagConstraints.VERTICAL;
3249            GBL.setConstraints(lblChangePassword,GBC);
3250            lblChangePassword.setFont(f);
3251            lblChangePassword.setHorizontalAlignment(JLabel.CENTER);
3252            lblChangePassword.setForeground(Color.white);
3253            getContentPane().add(lblChangePassword);
3254
3255            GBC.gridwidth = 2;
3256            GBC.gridy = 3;
3257            GBC.gridx = 1;
3258            GBC.anchor = GridBagConstraints.CENTER;
3259            GBL.setConstraints(lblCurrentPass, GBC);
3260            lblCurrentPass.setForeground(Color.white);
3261            getContentPane().add(lblCurrentPass);
3262
3263            GBC.gridy = 5;
3264            GBL.setConstraints(lblNewPass, GBC);
3265            lblNewPass.setForeground(Color.white);
3266            getContentPane().add(lblNewPass);
3267
3268            GBC.gridy = 7;
3269            GBL.setConstraints(lblNewConfPass, GBC);
3270            lblNewConfPass.setForeground(Color.white);
3271            getContentPane().add(lblNewConfPass);
3272
3273            GBC.gridy = 9;
3274            GBL.setConstraints(btnSubmit,GBC);
3275            GBC.ipady = 20;
3276            GBC.ipadx = 100;
3277            getContentPane().add(btnSubmit);
3278
3279            GBC.gridx = 3;
3280            GBC.gridy = 3;
3281            GBC.ipady = 0;
3282            GBC.ipadx = 0;
3283            GBL.setConstraints(currentPass, GBC);
3284            currentPass.setEchoChar('*');
3285            lblCurrentPass.setLabelFor(currentPass);
3286            getContentPane().add(currentPass);
3287
3288            GBC.gridy = 5;
3289            GBL.setConstraints(newPass, GBC);
3290            lblNewPass.setLabelFor(newPass);
3291            getContentPane().add(newPass);
3292
3293            GBC.gridy = 7;
3294            GBL.setConstraints(newConfPass, GBC);
3295            lblNewConfPass.setLabelFor(newConfPass);
3296            getContentPane().add(newConfPass);
3297
3298            GBC.gridx = 3;
3299            GBC.gridy = 9;
3300            GBC.ipady = 20;
3301            GBC.ipadx = 100;
3302            GBL.setConstraints(btnCancel,GBC);
3303            getContentPane().add(btnCancel);
3304
3305            getContentPane().setBackground(c);
3306            FR.setExtendedState(Frame.MAXIMIZED_BOTH);
```

```
3307
3308            btnSubmit.addActionListener(this);
3309            btnCancel.addActionListener(this);
3310            logOut.addActionListener(this);
3311            Exit.addActionListener(this);
3312            btnBack.addActionListener(this);
3313            validate();
3314        }
3315    //-----------------------------------------------------------------
3316    //This method is used to execute the appropriate method when the user performs an action event
3317        public void actionPerformed(ActionEvent ae)
3318        {
3319            if(ae.getSource()==btnSubmit)
3320            {
3321                String currentPassword = (currentPass.getText()).trim();
3322                String newPassword = (newPass.getText()).trim();
3323                String confNewPassword = (newConfPass.getText()).trim();
3324
3325                changePassword(currentPassword, newPassword, confNewPassword);
3326
3327                FR.setVisible(false);
3328                securitySett SP = new securitySett("");
3329                SP.FR.setVisible(true);
3330            }
3331            if(ae.getSource()==btnCancel)
3332            {
3333                FR.setVisible(false);
3334                securitySett SP = new securitySett("");
3335                SP.FR.setVisible(true);
3336            }
3337            if(ae.getSource()==logOut)
3338            {
3339                FR.setVisible(false);
3340                Program EP = new Program("");
3341                EP.FR.setVisible(true);
3342            }
3343            if(ae.getSource()==Exit)
3344            {
3345                System.exit(0);
3346            }
3347        }
3348    //-----------------------------------------------------------------
3349    //This method changes the system password. The first parameter is the
3350    //current password of the system. This is used to authorise the password change.
3351    //The second parameter is the new password to be assigned and the third parameter is
3352    //the second input of the new password for verification and to prevent a typo from occuring
3353        private void changePassword(String currentPass, String newPass, String newPassConf)
3354        {
3355            File PasswordStore = new File("SystemSecurity.dat");        //Creates .dat file that stores the program pas
3356            String pass = "";               //Initialises variable to store the password that will be read from the RAF
3357
3358            if((newPass.equals(""))||(newPassConf.equals(""))||(currentPass.equals("")))        //Checks if the user le
    any field blank
3359            {
3360                JOptionPane.showMessageDialog(this, "Error. You have left a mandatory field blank.","Error",
    JOptionPane.ERROR_MESSAGE);
3361                return;
3362            }
3363
```

```
3364            if(!PasswordStore.exists())     //Checks if the file storing the password exists in the same directory
3365            {
3366                Toolkit.getDefaultToolkit().beep();
3367                JOptionPane.showMessageDialog(this, "Error. The file that stores the password does not exist.","Error
        Message", JOptionPane.ERROR_MESSAGE);
3368                System.exit(1);             //Shuts down the program because the password file is required for system l
        and changing the password
3369            }
3370            else
3371            {
3372                try
3373                {
3374                    RandomAccessFile RAF = new RandomAccessFile(PasswordStore, "rw");       //Creates object to read Ra
        Access File
3375                    RAF.seek(0);                                                            //Sets pointer to start of
3376
3377                    for(int i = 0; i < 20; i++)
3378                    {
3379                        byte letter = RAF.readByte();           //Reads each character from the first 20 characters of
        file
3380                        pass = pass + (char) letter;            //Adds each character of the first 20 in the file to
        variable 'pass'
3381                    }
3382
3383                    pass = pass.trim();                         //Removes spaces from the string
3384
3385                    if(pass.equals(currentPass))                //Tests if the current system password was correctly in
3386                    {
3387                        if(newPass.equals(newPassConf))         //Tests if the new password is equal to the second inpu
3388                        {
3389                            RAF.seek(0);                                            //Sets file pointer to the start of
        file
3390                            for(int i = newPass.length(); i < 20; i++)              //Sets String length to 20 characte
3391                            {
3392                                newPass += " ";
3393                            }
3394                            RAF.writeBytes(newPass);                                //Add new password to the beginning
        the file
3395                            JOptionPane.showMessageDialog(this, "The password was successfully changed.","Change
        Successful!", JOptionPane.PLAIN_MESSAGE);
3396                        }
3397                        else
3398                        {
3399                            JOptionPane.showMessageDialog(this, "The new password that you entered in both fields do no
        match. Please try again.","Error", JOptionPane.ERROR_MESSAGE);
3400                        }
3401                    }
3402                    else
3403                    {
3404                        JOptionPane.showMessageDialog(this, "The current password that you entered is incorrect.\nPleas
        note that this field is case-sensitive.","Error", JOptionPane.ERROR_MESSAGE);
3405                    }
3406
3407                    RAF.close();                    //Closes RAF
3408                }
3409                catch(Exception e)
3410                {
3411                    Toolkit.getDefaultToolkit().beep();
3412                    JOptionPane.showMessageDialog(this, "An error has occured. Please contact Harris Rasheed to deal wi
```

```
3412   this issue\nError Code: " + e,"Error Message", JOptionPane.ERROR_MESSAGE);    //Output any errors caught
3413               }
3414           }
3415       }
3416   }
3417   //-------------------------------------------------------------------------------------------------------------
3418   //Assign Secret Question Screen; This is the screen where the user can change the secret question
3419   //and answer of the system.
3420   class secretQtion extends JFrame implements ActionListener
3421   {
3422       JFrame FR = new JFrame("Recycling Activity Monitoring System - Assign Secret Question");
3423       Container Obj1 = getContentPane();
3424       GridBagLayout GBL = new GridBagLayout();
3425       GridBagConstraints GBC = new GridBagConstraints();
3426       JMenuBar MB = new JMenuBar();
3427       JMenu file = new JMenu("File");
3428       JMenuItem logOut = new JMenuItem("Log out");
3429       JMenuItem Exit = new JMenuItem("Exit");
3430       JButton btnSubmit = new JButton("Submit");
3431       JButton btnCancel = new JButton("Cancel");
3432       JButton btnBack = new JButton("Back");
3433       JLabel lblCurrentPass = new JLabel("Current Password: ");
3434       JLabel lblSQtion = new JLabel("Secret Question: ");
3435       JLabel lblSAnswer = new JLabel("Answer: ");
3436       JLabel lblAssignSQtion = new JLabel("Assign Secret Question");
3437       JPasswordField currentPass = new JPasswordField(20);
3438       JTextField secQtion = new JTextField(20);
3439       JTextField secAnswer = new JTextField(20);
3440       Color c = new Color(6,69,1);
3441       Font f = new Font("Comic Sans MS", Font.BOLD, 22);
3442
3443   //-------------------------------------------------------------------
3444   //Constructor for the Secret Question Screen that places components on the Frame
3445       public secretQtion(String str)
3446       {
3447           super(str);
3448
3449           FR.setJMenuBar(MB);
3450           MB.add(file);
3451           file.add(logOut);
3452           file.add(Exit);
3453
3454           getContentPane().setLayout(GBL);
3455           FR.add(Obj1);
3456
3457           GBC.fill = GridBagConstraints.BOTH;
3458           GBC.anchor = GridBagConstraints.PAGE_START;
3459           GBC.gridwidth = 4;
3460           GBC.gridheight = 2;
3461           GBC.gridy = 1;
3462           GBC.gridx = 1;
3463           GBC.insets = new Insets(10,10,10,10);
3464           GBC.fill = GridBagConstraints.VERTICAL;
3465           GBL.setConstraints(lblAssignSQtion,GBC);
3466           lblAssignSQtion.setFont(f);
3467           lblAssignSQtion.setHorizontalAlignment(JLabel.CENTER);
3468           lblAssignSQtion.setForeground(Color.white);
3469           getContentPane().add(lblAssignSQtion);
3470
```

```
3471            GBC.gridwidth = 2;
3472            GBC.gridy = 3;
3473            GBC.anchor = GridBagConstraints.CENTER;
3474            GBL.setConstraints(lblCurrentPass, GBC);
3475            lblCurrentPass.setForeground(Color.white);
3476            getContentPane().add(lblCurrentPass);
3477
3478            GBC.gridy = 5;
3479            GBL.setConstraints(lblSQtion, GBC);
3480            lblSQtion.setForeground(Color.white);
3481            getContentPane().add(lblSQtion);
3482
3483            GBC.gridy = 7;
3484            GBL.setConstraints(lblSAnswer, GBC);
3485            lblSAnswer.setForeground(Color.white);
3486            getContentPane().add(lblSAnswer);
3487
3488            GBC.gridy = 9;
3489            GBL.setConstraints(btnSubmit,GBC);
3490            GBC.ipady = 20;
3491            GBC.ipadx = 100;
3492            getContentPane().add(btnSubmit);
3493
3494            GBC.gridx = 3;
3495            GBC.gridy = 3;
3496            GBC.ipady = 0;
3497            GBC.ipadx = 0;
3498            GBL.setConstraints(currentPass, GBC);
3499            currentPass.setEchoChar('*');
3500            lblCurrentPass.setLabelFor(currentPass);
3501            getContentPane().add(currentPass);
3502
3503            GBC.gridy = 5;
3504            lblSQtion.setLabelFor(secQtion);
3505            GBL.setConstraints(secQtion, GBC);
3506            getContentPane().add(secQtion);
3507
3508            GBC.gridy = 7;
3509            lblSAnswer.setLabelFor(secAnswer);
3510            GBL.setConstraints(secAnswer, GBC);
3511            getContentPane().add(secAnswer);
3512
3513            GBC.gridy = 9;
3514            GBC.ipady = 20;
3515            GBC.ipadx = 100;
3516            GBL.setConstraints(btnCancel,GBC);
3517            getContentPane().add(btnCancel);
3518
3519            getContentPane().setBackground(c);
3520            FR.setExtendedState(Frame.MAXIMIZED_BOTH);
3521
3522            btnSubmit.addActionListener(this);
3523            btnCancel.addActionListener(this);
3524            logOut.addActionListener(this);
3525            Exit.addActionListener(this);
3526            btnBack.addActionListener(this);
3527            validate();
3528        }
3529    //-----------------------------------------------------------------
```

```
3530   //This method is used to execute the appropriate method when the user performs an action event
3531       public void actionPerformed(ActionEvent ae)
3532       {
3533           if(ae.getSource()==btnSubmit)
3534           {
3535               String currentPassword = (currentPass.getText()).trim();
3536               String sQtion = (secQtion.getText()).trim();
3537               String sAnswer = (secAnswer.getText()).trim();
3538               changeSecretQtion(currentPassword, sQtion, sAnswer);
3539               FR.setVisible(false);
3540               securitySett SP = new securitySett("");
3541               SP.FR.setVisible(true);
3542           }
3543           if(ae.getSource()==btnCancel)
3544           {
3545               FR.setVisible(false);
3546               securitySett SP = new securitySett("");
3547               SP.FR.setVisible(true);
3548           }
3549           if(ae.getSource()==logOut)
3550           {
3551               FR.setVisible(false);
3552               Program EP = new Program("");
3553               EP.FR.setVisible(true);
3554           }
3555           if(ae.getSource()==Exit)
3556           {
3557               System.exit(0);
3558           }
3559       }
3560   //-----------------------------------------------------------------
3561   //This method changes the secret question of the system when the user
3562   //selects to do so. The first parameter is the current password of the system.
3563   //This is used to authorise the ssecret question change. The second parameter is the new
3564   //secret question to be assigned and the third parameter is the answer to the new secret question
3565       private void changeSecretQtion(String currentPass, String sQtion, String sAnswer)
3566       {
3567           File PasswordStore = new File("SystemSecurity.dat");        //Creates .dat file that stores the program pas
3568           String pass = "";           //Initialises variable to store the password that will be read from the random
       access file
3569
3570           if(!PasswordStore.exists())     //Checks if the file storing the password exists in the same directory
3571           {
3572               Toolkit.getDefaultToolkit().beep();
3573               JOptionPane.showMessageDialog(this, "Error. The file that stores the password does not exist.","Error
       Message", JOptionPane.ERROR_MESSAGE);
3574               System.exit(1);             //Shuts down the program because the password file is required for system l
       and changing the password
3575           }
3576           else
3577           {
3578               try
3579               {
3580                   RandomAccessFile RAF = new RandomAccessFile(PasswordStore, "rw");       //Creates object to read Ra
       Access File
3581                   RAF.seek(0);                                                            //Sets pointer to start of
3582
3583                   for(int i = 0; i < 20; i++)
3584                   {
```

```
3585                    byte letter = RAF.readByte();           //Reads each character from the first 20 characters of
    file
3586                    pass = pass + (char) letter;            //Adds each character of the first 20 in the file to
    variable 'pass'
3587                }
3588
3589                pass = pass.trim();                         //Removes spaces from the string
3590
3591                if(pass.equals(currentPass))
3592                {
3593                    RAF.seek(20);                                       //Sets file pointer to the start of the fil
3594                    for(int i = sQtion.length(); i < 60; i++)           //Sets length of secret question string to
    characters
3595                    {
3596                        sQtion += " ";
3597                    }
3598                    RAF.writeBytes(sQtion);                             //Add new secret question to the 20th posit
3599
3600                    RAF.seek(80);
3601                    for(int i = sAnswer.length(); i < 20; i++)          //Sets length of secret answer string to 20
    characters
3602                    {
3603                        sAnswer += " ";
3604                    }
3605                    RAF.writeBytes(sAnswer);                            //Add new secret answer to the 80th positio
3606
3607                    JOptionPane.showMessageDialog(this, "The secret question was successfully changed.","Change
    Successful!", JOptionPane.PLAIN_MESSAGE);
3608                }
3609                else if((currentPass == "")||(sQtion == "")||(sAnswer == ""))       //Tests if any fields have been
    blank
3610                {
3611                    JOptionPane.showMessageDialog(this, "Error. You have left a mandatory field blank.","Error",
    JOptionPane.ERROR_MESSAGE);
3612                }
3613                else
3614                {
3615                    JOptionPane.showMessageDialog(this, "The current password that you entered is incorrect.\nPleas
    note that this field is case-sensitive.","Error", JOptionPane.ERROR_MESSAGE);
3616                }
3617                RAF.close();                    //Closes Random Access File
3618            }
3619            catch(Exception e)
3620            {
3621                Toolkit.getDefaultToolkit().beep();
3622                JOptionPane.showMessageDialog(this, "An error has occured. Please contact Harris Rasheed to deal wi
    this issue\nError Code: " + e,"Error Message", JOptionPane.ERROR_MESSAGE);    //Output any errors caught
3623            }
3624        }
3625    }
3626 }
3627 //-----------------------------------------------------------------------------------------------------------------
```