```
 1 import java.awt.BorderLayout;
39
40 public class GUI extends JFrame {
41     private int accountIndexUsed;
42     private JTextField textField;
43     private JTextField textField_1;
44
45     public static void main(String[] args) throws IOException {
46         EventQueue.invokeLater(new Runnable() {
47             public void run() {
48                 try { //using a try-catch allows the program to robustly attempt running the
   GUI
49                     GUI realEstateTrackerFrame = new GUI();
50                     realEstateTrackerFrame.setVisible(true);
51                 } catch (Exception e) {
52                     e.printStackTrace();
53                 }
54             }
55         });
56     }
57
58     public GUI() throws IOException {
59         setTitle("Real Estate Tracker"); //GUI extends JFrame so these methods reference the
   GUI object
60         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
61         setLocation(200,100);
62         setSize(750,475);
63         setResizable(false); //ensures all components are always visible  (requested by
   client)
64         getContentPane().setLayout(new CardLayout(0, 0));
65
66         JPanel loginPane = new JPanel();
67         loginPane.setForeground(Color.BLACK);
68         getContentPane().add(loginPane, "Login Pane");
69         loginPane.setLayout(null);
70
71         JPanel mainMenuPane = new JPanel();
72         getContentPane().add(mainMenuPane, "Main Menu Pane");
73         mainMenuPane.setLayout(null);
74
75         JPanel newProjectPane = new JPanel();
76         getContentPane().add(newProjectPane, "New Project Pane");
77         newProjectPane.setLayout(null);
78
79         JPanel projectPane = new JPanel();
80         getContentPane().add(projectPane, "Project Pane");
81         projectPane.setLayout(null);
82
83         JTabbedPane tabbedPane = new JTabbedPane(JTabbedPane.TOP);
84         tabbedPane.setBounds(0, 60, 744, 386);
85         projectPane.add(tabbedPane);
86
87             JPanel detailsPanel = new JPanel();
88             tabbedPane.addTab("Details", null, detailsPanel, null);
89             detailsPanel.setLayout(null);
90
91                 JLabel projectAddressLabel = new JLabel("Address: ");
```

```
92                projectAddressLabel.setFont(new Font("Times New Roman", Font.BOLD, 12));
93                projectAddressLabel.setBounds(25, 20, 91, 16);
94                detailsPanel.add(projectAddressLabel);
95
96                JLabel projectCityLabel = new JLabel("City: ");
97                projectCityLabel.setFont(new Font("Times New Roman", Font.BOLD, 12));
98                projectCityLabel.setBounds(25, 40, 91, 16);
99                detailsPanel.add(projectCityLabel);
100
101               JLabel projectStateLabel = new JLabel("State: ");
102               projectStateLabel.setFont(new Font("Times New Roman", Font.BOLD, 12));
103               projectStateLabel.setBounds(25, 60, 91, 16);
104               detailsPanel.add(projectStateLabel);
105
106               JLabel projectTotalLotsInDevelopmentLabel = new JLabel("Total Lots in
    Development: ");
107               projectTotalLotsInDevelopmentLabel.setFont(new Font("Times New Roman",
    Font.BOLD, 12));
108               projectTotalLotsInDevelopmentLabel.setBounds(25, 100, 159, 16);
109               detailsPanel.add(projectTotalLotsInDevelopmentLabel);
110
111               JLabel projectTotalLotsDevelopedToDateLabel = new JLabel("Total Lots
    Developed To Date: ");
112               projectTotalLotsDevelopedToDateLabel.setFont(new Font("Times New Roman",
    Font.BOLD, 12));
113               projectTotalLotsDevelopedToDateLabel.setBounds(25, 120, 178, 16);
114               detailsPanel.add(projectTotalLotsDevelopedToDateLabel);
115
116               JLabel projectTotalLotsUnderConstrustionLabel = new JLabel("Total Lots Under
    Construction: ");
117               projectTotalLotsUnderConstrustionLabel.setFont(new Font("Times New Roman",
    Font.BOLD, 12));
118               projectTotalLotsUnderConstrustionLabel.setBounds(25, 140, 178, 16);
119               detailsPanel.add(projectTotalLotsUnderConstrustionLabel);
120
121               JLabel projectTotalLotsRemainingLabel = new JLabel("Total Lots Remaining: ");
122               projectTotalLotsRemainingLabel.setFont(new Font("Times New Roman", Font.BOLD,
    12));
123               projectTotalLotsRemainingLabel.setBounds(25, 160, 130, 16);
124               detailsPanel.add(projectTotalLotsRemainingLabel);
125
126               JLabel projectLotsSoldWithinLast30DaysLabel = new JLabel("Lots Sold Within
    the Last 30 Days: ");
127               projectLotsSoldWithinLast30DaysLabel.setFont(new Font("Times New Roman",
    Font.BOLD, 12));
128               projectLotsSoldWithinLast30DaysLabel.setBounds(25, 200, 197, 16);
129               detailsPanel.add(projectLotsSoldWithinLast30DaysLabel);
130
131               JLabel projectSelloutLabel = new JLabel("Project Sellout: ");
132               projectSelloutLabel.setFont(new Font("Times New Roman", Font.BOLD, 12));
133               projectSelloutLabel.setBounds(25, 220, 106, 16);
134               detailsPanel.add(projectSelloutLabel);
135
136               JLabel projectAvgCostPerLotLabel = new JLabel("Average Cost Per Lot: ");
137               projectAvgCostPerLotLabel.setFont(new Font("Times New Roman", Font.BOLD,
    12));
138               projectAvgCostPerLotLabel.setBounds(25, 260, 144, 16);
```

```
139                 detailsPanel.add(projectAvgCostPerLotLabel);
140
141                 JLabel projectAvgPricePerLotLabel = new JLabel("Average Sales Price Per Lot:
    ");
142                 projectAvgPricePerLotLabel.setFont(new Font("Times New Roman", Font.BOLD,
    12));
143                 projectAvgPricePerLotLabel.setBounds(25, 280, 178, 16);
144                 detailsPanel.add(projectAvgPricePerLotLabel);
145
146                 JLabel projectTotalRemainingRevenueLabel = new JLabel("Total Remaining
    Revenue: ");
147                 projectTotalRemainingRevenueLabel.setFont(new Font("Times New Roman",
    Font.BOLD, 12));
148                 projectTotalRemainingRevenueLabel.setBounds(25, 300, 178, 16);
149                 detailsPanel.add(projectTotalRemainingRevenueLabel);
150
151                 JLabel projectProjectedRemainingProfitLabel = new JLabel("Projected Remaining
    Profit: ");
152                 projectProjectedRemainingProfitLabel.setFont(new Font("Times New Roman",
    Font.BOLD, 12));
153                 projectProjectedRemainingProfitLabel.setBounds(25, 320, 197, 16);
154                 detailsPanel.add(projectProjectedRemainingProfitLabel);
155
156                 JButton btnEdit = new JButton("Edit");
157                 btnEdit.setBounds(622, 317, 89, 23);
158                 detailsPanel.add(btnEdit);
159
160                 JLabel lblPartnersInvolved = new JLabel("Partners Involved");
161                 lblPartnersInvolved.setBounds(475, 13, 178, 23);
162                 lblPartnersInvolved.setFont(new Font("Times New Roman", Font.BOLD |
    Font.ITALIC, 22));
163                 detailsPanel.add(lblPartnersInvolved);
164
165                 JLabel[] projectData = new JLabel[16];
166                 for(int x=0;x<projectData.length;x++) {
167                     projectData[x] = new JLabel("This is a test message");
168                     detailsPanel.add(projectData[x]);
169                     projectData[x].setFont(new Font("Times New Roman", Font.BOLD, 12));}
170                 projectData[0].setBounds(231, 20, 200, 16);
171                 projectData[1].setBounds(231, 40, 91, 16);
172                 projectData[2].setBounds(231, 60, 91, 16);
173                 projectData[3].setBounds(231, 100, 91, 16);
174                 projectData[4].setBounds(231, 120, 91, 16);
175                 projectData[5].setBounds(231, 140, 91, 16);
176                 projectData[6].setBounds(231, 160, 91, 16);
177                 projectData[7].setBounds(231, 200, 91, 16);
178                 projectData[8].setBounds(231, 220, 91, 16);
179                 projectData[9].setBounds(231, 260, 91, 16);
180                 projectData[10].setBounds(231, 280, 91, 16);
181                 projectData[11].setBounds(231, 300, 91, 16);
182                 projectData[12].setBounds(231, 320, 91, 16);
183                 projectData[13].setBounds(270, 120, 91, 16);
184                 projectData[14].setBounds(270, 140, 91, 16);
185                 projectData[15].setBounds(270, 160, 91, 16);
186
187                 JLabel[] partnersInvoled = new JLabel[10];
188                 for(int x=0;x<10;x++) {
```

```
189                     partnersInvoled[x] = new JLabel("Partner Involed");
190                     partnersInvoled[x].setFont(new Font("Times New Roman", Font.BOLD, 12));
191                     partnersInvoled[x].setBounds(485, 40+(x*20), 200, 15);
192                 }
193 
194             JPanel tasksPanel = new JPanel();
195             tabbedPane.addTab("Tasks", null, tasksPanel, null);
196             tasksPanel.setLayout(null);
197 
198                 JLabel lblCreateNewTask = new JLabel("Create New Task");
199                 lblCreateNewTask.setBounds(10, 291, 206, 28);
200                 lblCreateNewTask.setFont(new Font("Times New Roman", Font.BOLD | Font.ITALIC,
    24));
201                 tasksPanel.add(lblCreateNewTask);
202 
203                 JLabel lblTask = new JLabel("Task");
204                 lblTask.setBounds(10, 330, 46, 14);
205                 lblTask.setFont(new Font("Times New Roman", Font.BOLD, 12));
206                 tasksPanel.add(lblTask);
207 
208                 JLabel lblContractor = new JLabel("Contractor");
209                 lblContractor.setBounds(176, 330, 74, 14);
210                 lblContractor.setFont(new Font("Times New Roman", Font.BOLD, 12));
211                 tasksPanel.add(lblContractor);
212 
213                 JLabel lblProjectedCompletion = new JLabel("Projected Completion");
214                 lblProjectedCompletion.setBounds(367, 330, 125, 14);
215                 lblProjectedCompletion.setFont(new Font("Times New Roman", Font.BOLD, 12));
216                 tasksPanel.add(lblProjectedCompletion);
217 
218                 JComboBox tasksComboBox = new JComboBox();
219                 tasksComboBox.setModel(new DefaultComboBoxModel(new String[] {" ", "Aminity",
    "Furniture", "Landscaping", "Mailboxes"}));
220                 tasksComboBox.setBounds(48, 327, 111, 20);
221                 tasksPanel.add(tasksComboBox);
222 
223                 JTextField contractorTextField = new JTextField();
224                 contractorTextField.setBounds(242, 327, 111, 20);
225                 tasksPanel.add(contractorTextField);
226                 contractorTextField.setColumns(10);
227 
228                 JTextField projCompMMTextField = new JTextField();
229                 projCompMMTextField.setBounds(488, 327, 31, 20);
230                 tasksPanel.add(projCompMMTextField);
231                 projCompMMTextField.setColumns(10);
232 
233                 JTextField projCompDDTextField = new JTextField();
234                 projCompDDTextField.setColumns(10);
235                 projCompDDTextField.setBounds(529, 327, 31, 20);
236                 tasksPanel.add(projCompDDTextField);
237 
238                 JTextField projCompYYYYTextField = new JTextField();
239                 projCompYYYYTextField.setColumns(10);
240                 projCompYYYYTextField.setBounds(570, 327, 65, 20);
241                 tasksPanel.add(projCompYYYYTextField);
242 
243                 JButton btnExport = new JButton("Export");
```

```
244                 btnExport.setBounds(641, 279, 87, 23);
245                 tasksPanel.add(btnExport);
246 
247                 JButton btnAdd = new JButton("Add");
248                 btnAdd.setBounds(657, 327, 57, 20);
249                 tasksPanel.add(btnAdd);
250 
251                 String[] columnNames = {"Task","Contractor","Projected Completion"};
252                 DefaultTableModel tasksTableModel = new DefaultTableModel(columnNames,0){
253                     //makes cells uneditable by ovveriding DefaultTableModel's
    isCellEditable() method
254                     public boolean isCellEditable(int rows, int columns) {return false;}
255                 };
256 
257                 JTable tasksTable = new JTable(tasksTableModel);
258                 tasksTable.setBounds(10, 11, 719, 270);
259                 tasksTable.getTableHeader().setReorderingAllowed(false);
260                 tasksPanel.add(tasksTable);
261 
262                 JScrollPane scrollpane = new JScrollPane(tasksTable);
263                 scrollpane.setBounds(10, 11, 719, 270);
264                 tasksPanel.add(scrollpane);
265 
266             JPanel commentsPanel = new JPanel();
267             tabbedPane.addTab("Comments", null, commentsPanel, null);
268             commentsPanel.setLayout(null);
269 
270                 JLabel lblCreateNewComment = new JLabel("Create New");
271                 lblCreateNewComment.setFont(new Font("Times New Roman", Font.BOLD |
    Font.ITALIC, 24));
272                 lblCreateNewComment.setBounds(10, 292, 128, 28);
273                 commentsPanel.add(lblCreateNewComment);
274 
275                 JLabel lblCreateNewCommentpt2 = new JLabel("Comment");
276                 lblCreateNewCommentpt2.setFont(new Font("Times New Roman", Font.BOLD |
    Font.ITALIC, 24));
277                 lblCreateNewCommentpt2.setBounds(10, 319, 128, 28);
278                 commentsPanel.add(lblCreateNewCommentpt2);
279 
280                 JButton btnSubmit_1 = new JButton("Submit");
281                 btnSubmit_1.setBounds(640, 309, 89, 23);
282                 commentsPanel.add(btnSubmit_1);
283 
284                 JPanel commentBorderPanel = new JPanel();
285                 commentBorderPanel.setBackground(Color.GRAY);
286                 commentBorderPanel.setBounds(137, 292, 493, 60);
287                 commentsPanel.add(commentBorderPanel);
288                 commentBorderPanel.setLayout(null);
289 
290                 JTextArea commentTextArea = new JTextArea();
291                 commentTextArea.setBounds(2, 2, 489, 56);
292                 //commentBorderPanel.add(commentTextArea);
293                 commentTextArea.setBackground(Color.WHITE);
294                 commentTextArea.setLineWrap(true);
295                     JScrollPane commentTextareaScrollPane = new JScrollPane(commentTextArea,
296                                                                             ScrollPaneConstan
    ts.VERTICAL_SCROLLBAR_AS_NEEDED,
```

```
297                                                                                 ScrollPaneConstan
   ts.HORIZONTAL_SCROLLBAR_NEVER);
298                     commentTextareaScrollPane.setBounds(2, 2, 489, 56);
299                     commentBorderPanel.add(commentTextareaScrollPane);
300
301
302                 JLabel lblName = new JLabel("Name");
303                 lblName.setHorizontalAlignment(SwingConstants.TRAILING);
304                 lblName.setFont(new Font("Times New Roman", Font.BOLD | Font.ITALIC, 18));
305                 lblName.setBounds(10, 1, 186, 20);
306                 commentsPanel.add(lblName);
307
308                 JLabel lblComment = new JLabel("Comment");
309                 lblComment.setFont(new Font("Times New Roman", Font.BOLD | Font.ITALIC, 18));
310                 lblComment.setBounds(211, 1, 158, 20);
311                 commentsPanel.add(lblComment);
312
313                 int numberOfCommentsSupported = 50;
314
315                     JPanel commentBoardPanel = new JPanel();
316                     commentBoardPanel.setLayout(null);
317                     commentBoardPanel.setBounds(10, 20, 719,
   20+(30*numberOfCommentsSupported));
318                     commentBoardPanel.setPreferredSize(new
   Dimension(719,(20+(30*numberOfCommentsSupported))));
319
320                     JLabel[] nameLabelArray = new JLabel[numberOfCommentsSupported];
321                     JLabel[] commentLabelArray = new JLabel[numberOfCommentsSupported];
322                         for(int x=0;x<numberOfCommentsSupported;x++) {
323                             nameLabelArray[x] = new JLabel("This is a name #"+x);
324                             nameLabelArray[x].setBounds(10, 10+(30*x), 175, 15);
325                             nameLabelArray[x].setFont(new Font("Times New Roman", Font.BOLD,
   12));
326                             nameLabelArray[x].setHorizontalAlignment(SwingConstants.TRAILING)
   ;
327                             nameLabelArray[x].setVisible(false);
328                             commentBoardPanel.add(nameLabelArray[x]);
329
330                             commentLabelArray[x] = new JLabel("This is a comment #"+x);
331                             commentLabelArray[x].setBounds(200, 10+(30*x), 500, 15);
332                             commentLabelArray[x].setFont(new Font("Times New Roman",
   Font.BOLD, 12));
333                             commentLabelArray[x].setVisible(false);
334                             commentBoardPanel.add(commentLabelArray[x]);
335                         }
336
337                     JScrollPane commentGridScrollPane = new JScrollPane(commentBoardPanel,
   ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS,
   ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
338                     commentGridScrollPane.setBounds(10, 20, 719, 250);
339                     commentGridScrollPane.setPreferredSize(new Dimension(719,250));
340                     commentsPanel.add(commentGridScrollPane);
341
342             JPanel problemsPanel = new JPanel();
343             tabbedPane.addTab("Problems", null, problemsPanel, null);
344             problemsPanel.setLayout(null);
345
```

```
346                 JLabel lblCreateNewproblem = new JLabel("Add New problem");
347                 lblCreateNewproblem.setBounds(10, 291, 206, 28);
348                 lblCreateNewproblem.setFont(new Font("Times New Roman", Font.BOLD |
    Font.ITALIC, 24));
349                 problemsPanel.add(lblCreateNewproblem);
350 
351                 JLabel lblproblem = new JLabel("Title");
352                 lblproblem.setBounds(10, 330, 46, 14);
353                 lblproblem.setFont(new Font("Times New Roman", Font.BOLD, 12));
354                 problemsPanel.add(lblproblem);
355 
356                 String[] problemColumnNames = {"Title","Description","Priority"};
357                 DefaultTableModel problemsTableModel = new
    DefaultTableModel(problemColumnNames,0){
358                     //makes cells uneditable by ovveriding DefaultTableModel's
    isCellEditable() method
359                     public boolean isCellEditable(int rows, int columns) {return false;}
360                 };
361 
362                 JTable problemsTable = new JTable(problemsTableModel);
363                 problemsTable.setBounds(10, 11, 719, 270);
364                 problemsTable.getTableHeader().setReorderingAllowed(false);
365                 problemsPanel.add(problemsTable);
366 
367                 JScrollPane scrollpaneProblems = new JScrollPane(problemsTable);
368                 scrollpaneProblems.setBounds(10, 11, 719, 270);
369                 problemsPanel.add(scrollpaneProblems);
370 
371                 JTextField titleTextField = new JTextField();
372                 titleTextField.setBounds(48, 327, 117, 20);
373                 problemsPanel.add(titleTextField);
374                 titleTextField.setColumns(10);
375 
376                 JLabel lblDescription = new JLabel("Description");
377                 lblDescription.setFont(new Font("Times New Roman", Font.BOLD, 12));
378                 lblDescription.setBounds(192, 330, 72, 14);
379                 problemsPanel.add(lblDescription);
380 
381                 JTextField descriptionTextField = new JTextField();
382                 descriptionTextField.setBounds(274, 327, 206, 20);
383                 problemsPanel.add(descriptionTextField);
384                 descriptionTextField.setColumns(10);
385 
386                 JLabel lblPriority = new JLabel("Priority");
387                 lblPriority.setFont(new Font("Times New Roman", Font.BOLD, 12));
388                 lblPriority.setBounds(507, 330, 52, 14);
389                 problemsPanel.add(lblPriority);
390 
391                 JComboBox priorityComboBox = new JComboBox();
392                 priorityComboBox.setModel(new DefaultComboBoxModel(new String[] {"1", "2",
    "3", "4", "5"}));
393                 priorityComboBox.setBounds(559, 327, 44, 20);
394                 problemsPanel.add(priorityComboBox);
395 
396                 JButton btnAddProblem = new JButton("Add");
397                 btnAddProblem.setBounds(640, 326, 89, 23);
398                 problemsPanel.add(btnAddProblem);
```

```
399
400                 JButton btnRecommendAProblem = new JButton("Solve a Problem");
401                 btnRecommendAProblem.setBounds(559, 291, 170, 23);
402                 problemsPanel.add(btnRecommendAProblem);
403
404
405
406
407         JLabel lblNewProjectTitle = new JLabel("New Project");
408         lblNewProjectTitle.setFont(new Font("Times New Roman", Font.BOLD | Font.ITALIC, 36));
409         lblNewProjectTitle.setBounds(71, 0, 235, 49);
410         newProjectPane.add(lblNewProjectTitle);
411
412         JButton btnBack = new JButton("\u2190");
413         btnBack.setToolTipText("Going to the main menu won't delete your data");
414         btnBack.setBounds(12, 12, 46, 19);
415         newProjectPane.add(btnBack);
416         btnBack.setFocusPainted(false);
417
418         JLabel lblProjectName = new JLabel("Project Name:");
419         lblProjectName.setFont(new Font("Times New Roman", Font.BOLD, 12));
420         lblProjectName.setBounds(12, 102, 91, 16);
421         newProjectPane.add(lblProjectName);
422
423         JTextField projectNameTextfield = new JTextField();
424         projectNameTextfield.setBounds(96, 99, 167, 20);
425         newProjectPane.add(projectNameTextfield);
426         projectNameTextfield.setColumns(10);
427
428         JLabel lblAdress = new JLabel("Adress: ");
429         lblAdress.setFont(new Font("Times New Roman", Font.BOLD, 12));
430         lblAdress.setBounds(12, 133, 91, 16);
431         newProjectPane.add(lblAdress);
432
433         JTextField adressTextfield = new JTextField();
434         adressTextfield.setColumns(10);
435         adressTextfield.setBounds(96, 130, 167, 20);
436         newProjectPane.add(adressTextfield);
437
438         JLabel lblCity = new JLabel("City: ");
439         lblCity.setFont(new Font("Times New Roman", Font.BOLD, 12));
440         lblCity.setBounds(12, 164, 91, 16);
441         newProjectPane.add(lblCity);
442
443         JTextField cityTextfield = new JTextField();
444         cityTextfield.setColumns(10);
445         cityTextfield.setBounds(96, 161, 167, 20);
446         newProjectPane.add(cityTextfield);
447
448         JLabel lblState = new JLabel("State: ");
449         lblState.setFont(new Font("Times New Roman", Font.BOLD, 12));
450         lblState.setBounds(12, 194, 91, 16);
451         newProjectPane.add(lblState);
452
453         JTextField stateTextfield = new JTextField();
454         stateTextfield.setColumns(10);
455         stateTextfield.setBounds(96, 192, 167, 20);
```

```
456         newProjectPane.add(stateTextfield);
457
458         JLabel lblTotalLotsInDev = new JLabel("Total lots in development: ");
459         lblTotalLotsInDev.setFont(new Font("Times New Roman", Font.BOLD, 12));
460         lblTotalLotsInDev.setBounds(12, 225, 147, 16);
461         newProjectPane.add(lblTotalLotsInDev);
462
463         JTextField totalLotsInDevTextfield = new JTextField();
464         totalLotsInDevTextfield.setColumns(10);
465         totalLotsInDevTextfield.setBounds(188, 222, 75, 20);
466         newProjectPane.add(totalLotsInDevTextfield);
467
468         JLabel lblPartersInvolved = new JLabel("Parters Involved: ");
469         lblPartersInvolved.setFont(new Font("Times New Roman", Font.BOLD, 12));
470         lblPartersInvolved.setBounds(12, 269, 104, 16);
471         newProjectPane.add(lblPartersInvolved);
472
473         JCheckBox chckbxAccount[] = new JCheckBox[9];
474         for(int x = 0; x < chckbxAccount.length; x++) {
475             chckbxAccount[x] = new JCheckBox();
476             if(x<5) chckbxAccount[x].setBounds(22,293+28*x,132,24);
477             else chckbxAccount[x].setBounds(172, 293+28*(x-5), 132, 24);
478             newProjectPane.add(chckbxAccount[x]);
479         }
480
481         JLabel lblTotalDevelopedTo = new JLabel("Total lots developed to date: ");
482         lblTotalDevelopedTo.setFont(new Font("Times New Roman", Font.BOLD, 12));
483         lblTotalDevelopedTo.setBounds(316, 101, 167, 16);
484         newProjectPane.add(lblTotalDevelopedTo);
485
486         JTextField lblTotalDevelopedTextfield = new JTextField();
487         lblTotalDevelopedTextfield.setColumns(10);
488         lblTotalDevelopedTextfield.setBounds(482, 99, 167, 20);
489         newProjectPane.add(lblTotalDevelopedTextfield);
490
491         JLabel TotalLotsUnderConstruction = new JLabel("Total lots under construction:  ");
492         TotalLotsUnderConstruction.setFont(new Font("Times New Roman", Font.BOLD, 12));
493         TotalLotsUnderConstruction.setBounds(316, 132, 167, 16);
494         newProjectPane.add(TotalLotsUnderConstruction);
495
496         JTextField totalLotsUnderConstructionTextfield = new JTextField();
497         totalLotsUnderConstructionTextfield.setColumns(10);
498         totalLotsUnderConstructionTextfield.setBounds(482, 130, 167, 20);
499         newProjectPane.add(totalLotsUnderConstructionTextfield);
500
501         JLabel lblTotalLotsRemaining = new JLabel("Total lots remaining:   ");
502         lblTotalLotsRemaining.setFont(new Font("Times New Roman", Font.BOLD, 12));
503         lblTotalLotsRemaining.setBounds(316, 163, 167, 16);
504         newProjectPane.add(lblTotalLotsRemaining);
505
506         JTextField totalLotsRemainingTextfield = new JTextField();
507         totalLotsRemainingTextfield.setColumns(10);
508         totalLotsRemainingTextfield.setBounds(482, 161, 167, 20);
509         newProjectPane.add(totalLotsRemainingTextfield);
510
511         JLabel lblLotsSoldWithin = new JLabel("Lots sold within the last 30 days: ");
512         lblLotsSoldWithin.setFont(new Font("Times New Roman", Font.BOLD, 12));
```

```
513         lblLotsSoldWithin.setBounds(316, 193, 195, 16);
514         newProjectPane.add(lblLotsSoldWithin);
515 
516         JTextField lotsSoldWithinTextfield = new JTextField();
517         lotsSoldWithinTextfield.setColumns(10);
518         lotsSoldWithinTextfield.setBounds(516, 191, 133, 20);
519         newProjectPane.add(lotsSoldWithinTextfield);
520 
521         JLabel lblAverageCostPer = new JLabel("Average cost per lot:                $");
522         lblAverageCostPer.setFont(new Font("Times New Roman", Font.BOLD, 12));
523         lblAverageCostPer.setBounds(316, 225, 195, 16);
524         newProjectPane.add(lblAverageCostPer);
525 
526         JTextField averageCostPerTextfield = new JTextField();
527         averageCostPerTextfield.setColumns(10);
528         averageCostPerTextfield.setBounds(482, 222, 167, 20);
529         newProjectPane.add(averageCostPerTextfield);
530 
531         JLabel lblAveragePricePer = new JLabel("Average sales price per lot:     $");
532         lblAveragePricePer.setFont(new Font("Times New Roman", Font.BOLD, 12));
533         lblAveragePricePer.setBounds(316, 250, 195, 16);
534         newProjectPane.add(lblAveragePricePer);
535 
536         JTextField averagePricePerTextfield = new JTextField();
537         averagePricePerTextfield.setColumns(10);
538         averagePricePerTextfield.setBounds(482, 247, 167, 20);
539         newProjectPane.add(averagePricePerTextfield);
540 
541         JButton btnSave = new JButton("Save");
542         btnSave.setBounds(634, 404, 98, 26);
543         newProjectPane.add(btnSave);
544 
545         JButton btnReset = new JButton("Reset");
546         btnReset.setBounds(516, 404, 98, 26);
547         newProjectPane.add(btnReset);
548 
549         JTextField usernameField = new JTextField();
550         usernameField.setBounds(107, 156, 218, 20);
551         loginPane.add(usernameField);
552         usernameField.setColumns(10);
553 
554         JPasswordField passwordField = new JPasswordField();
555         passwordField.setBounds(107, 195, 218, 20);
556         loginPane.add(passwordField);
557 
558         JLabel lblPassword = new JLabel("Password:");
559         lblPassword.setFont(new Font("Times New Roman", Font.BOLD, 12));
560         lblPassword.setBounds(35, 198, 62, 14);
561         loginPane.add(lblPassword);
562 
563         JLabel lblUsername = new JLabel("Username:");
564         lblUsername.setFont(new Font("Times New Roman", Font.BOLD, 12));
565         lblUsername.setBounds(35, 159, 62, 14);
566         loginPane.add(lblUsername);
567 
568         JLabel lblIncorrectUandP = new JLabel("Incorrect username and password, please try
    again");
```

```
569         lblIncorrectUandP.setForeground(Color.RED);
570         lblIncorrectUandP.setFont(new Font("Times New Roman", Font.BOLD, 14));
571         lblIncorrectUandP.setBounds(27, 303, 358, 20);
572         loginPane.add(lblIncorrectUandP);
573         lblIncorrectUandP.setVisible(false);
574 
575         JLabel lblWelcome = new JLabel();
576         lblWelcome.setText("Welcome client");
577         lblWelcome.setFont(new Font("Times New Roman", Font.BOLD | Font.ITALIC, 36));
578         lblWelcome.setBounds(10, 0, 468, 62);
579         mainMenuPane.add(lblWelcome);
580 
581         JButton btnSubmit = new JButton("Submit");
582         btnSubmit.setBounds(89, 258, 161, 23);
583         loginPane.add(btnSubmit);
584 
585         JLabel lblRealEstate = new JLabel("Real Estate");
586         lblRealEstate.setFont(new Font("Times New Roman", Font.BOLD | Font.ITALIC, 64));
587         lblRealEstate.setBounds(390, 124, 310, 73);
588         loginPane.add(lblRealEstate);
589 
590         JLabel labelTracker = new JLabel("Tracker");
591         labelTracker.setFont(new Font("Times New Roman", Font.BOLD | Font.ITALIC, 64));
592         labelTracker.setBounds(434, 191, 298, 73);
593         loginPane.add(labelTracker);
594 
595         JLabel lblLogin = new JLabel("Login");
596         lblLogin.setFont(new Font("Times New Roman", Font.BOLD, 24));
597         lblLogin.setBounds(127, 116, 109, 28);
598         loginPane.add(lblLogin);
599 
600         JLabel lblCurrentProjects = new JLabel("Current Projects");
601         lblCurrentProjects.setFont(new Font("Times New Roman", Font.BOLD, 20));
602         lblCurrentProjects.setBounds(10, 135, 189, 27);
603         mainMenuPane.add(lblCurrentProjects);
604 
605         JLabel lblNewProject = new JLabel("New Project");
606         lblNewProject.setFont(new Font("Times New Roman", Font.BOLD, 20));
607         lblNewProject.setBounds(453, 135, 189, 27);
608         mainMenuPane.add(lblNewProject);
609 
610         JLabel lblLogout = new JLabel("Logout");
611         lblLogout.setFont(new Font("Times New Roman", Font.BOLD, 20));
612         lblLogout.setBounds(453, 292, 189, 27);
613         mainMenuPane.add(lblLogout);
614 
615         JButton btnLogout = new JButton("logout");
616         btnLogout.setBounds(478, 330, 130, 23);
617         mainMenuPane.add(btnLogout);
618 
619         JButton btnNewProject = new JButton("new project");
620         btnNewProject.setBounds(478, 173, 130, 23);
621         mainMenuPane.add(btnNewProject);
622 
623         JButton btnEventLog = new JButton("Event Log");
624         btnEventLog.setBounds(616, 11, 118, 23);
625         mainMenuPane.add(btnEventLog);
```

```
626
627         JButton listOfProjectButtons[] = new JButton[50];
628         for(int x = 0; x < listOfProjectButtons.length; x++) {
629             listOfProjectButtons[x] = new JButton("EMPTY");
630             listOfProjectButtons[x].setBounds(30, (170+(30*x)), 150, 25);
631             mainMenuPane.add(listOfProjectButtons[x]);
632             listOfProjectButtons[x].setVisible(false);
633         }
634
635         JLabel lblProjectTitle = new JLabel("Project X");
636         lblProjectTitle.setFont(new Font("Times New Roman", Font.BOLD | Font.ITALIC, 36));
637         lblProjectTitle.setBounds(71, 0, 496, 49);
638         projectPane.add(lblProjectTitle);
639
640         JButton btnBack1 = new JButton("\u2190");
641         btnBack1.setBounds(12, 12, 46, 19);
642         projectPane.add(btnBack1);
643         btnBack1.setFocusPainted(false);
644
645         JPanel editProjectPane = new JPanel();
646         getContentPane().add(editProjectPane, "Edit Project Pane");
647         editProjectPane.setLayout(null);
648
649         JLabel lblEditProject = new JLabel("Edit Project");
650         lblEditProject.setFont(new Font("Times New Roman", Font.BOLD | Font.ITALIC, 36));
651         lblEditProject.setBounds(71, 0, 235, 49);
652         editProjectPane.add(lblEditProject);
653
654         JButton editProjectPaneEditButton = new JButton("\u2190");
655         editProjectPaneEditButton.setToolTipText("Going to the project menu WILL remove your
    revisions");
656         editProjectPaneEditButton.setFocusPainted(false);
657         editProjectPaneEditButton.setBounds(12, 12, 46, 19);
658         editProjectPane.add(editProjectPaneEditButton);
659
660         JLabel editProjectPaneProjectName = new JLabel("Project Name:");
661         editProjectPaneProjectName.setFont(new Font("Times New Roman", Font.BOLD, 12));
662         editProjectPaneProjectName.setBounds(12, 102, 91, 16);
663         editProjectPane.add(editProjectPaneProjectName);
664
665         JTextField editProjectPaneProjectNameTextfield = new JTextField();
666         editProjectPaneProjectNameTextfield.setColumns(10);
667         editProjectPaneProjectNameTextfield.setBounds(96, 99, 167, 20);
668         editProjectPane.add(editProjectPaneProjectNameTextfield);
669         editProjectPaneProjectNameTextfield.setEditable(false);
670
671         JLabel editProjectPaneAdress = new JLabel("Adress: ");
672         editProjectPaneAdress.setFont(new Font("Times New Roman", Font.BOLD, 12));
673         editProjectPaneAdress.setBounds(12, 133, 91, 16);
674         editProjectPane.add(editProjectPaneAdress);
675
676         JTextField editProjectPaneAdressTextfield = new JTextField();
677         editProjectPaneAdressTextfield.setColumns(10);
678         editProjectPaneAdressTextfield.setBounds(96, 130, 167, 20);
679         editProjectPane.add(editProjectPaneAdressTextfield);
680         editProjectPaneAdressTextfield.setEditable(false);
681
```

```
682         JLabel editProjectPaneCity = new JLabel("City: ");
683         editProjectPaneCity.setFont(new Font("Times New Roman", Font.BOLD, 12));
684         editProjectPaneCity.setBounds(12, 164, 91, 16);
685         editProjectPane.add(editProjectPaneCity);
686 
687         JTextField editProjectPaneCityTextfield = new JTextField();
688         editProjectPaneCityTextfield.setColumns(10);
689         editProjectPaneCityTextfield.setBounds(96, 161, 167, 20);
690         editProjectPane.add(editProjectPaneCityTextfield);
691         editProjectPaneCityTextfield.setEditable(false);
692 
693         JLabel editProjectPaneState = new JLabel("State: ");
694         editProjectPaneState.setFont(new Font("Times New Roman", Font.BOLD, 12));
695         editProjectPaneState.setBounds(12, 194, 91, 16);
696         editProjectPane.add(editProjectPaneState);
697 
698         JTextField editProjectPaneStateTextfield = new JTextField();
699         editProjectPaneStateTextfield.setColumns(10);
700         editProjectPaneStateTextfield.setBounds(96, 192, 167, 20);
701         editProjectPane.add(editProjectPaneStateTextfield);
702         editProjectPaneStateTextfield.setEditable(false);
703 
704         JLabel editProjectPaneTLID = new JLabel("Total lots in development: ");
705         editProjectPaneTLID.setFont(new Font("Times New Roman", Font.BOLD, 12));
706         editProjectPaneTLID.setBounds(12, 225, 147, 16);
707         editProjectPane.add(editProjectPaneTLID);
708 
709         JTextField  editProjectPaneTLIDTextfield = new JTextField();
710         //editProjectPaneTLIDTextfield.setHorizontalAlignment(JTextField.RIGHT);
711         editProjectPaneTLIDTextfield.setColumns(10);
712         editProjectPaneTLIDTextfield.setBounds(188, 222, 75, 20);
713         editProjectPane.add(editProjectPaneTLIDTextfield);
714 
715         JLabel editProjectPaneTLDTD = new JLabel("Total lots developed to date: ");
716         editProjectPaneTLDTD.setFont(new Font("Times New Roman", Font.BOLD, 12));
717         editProjectPaneTLDTD.setBounds(316, 101, 167, 16);
718         editProjectPane.add(editProjectPaneTLDTD);
719 
720         JTextField editProjectPaneTLDTDTextfield = new JTextField();
721         editProjectPaneTLDTDTextfield.setColumns(10);
722         editProjectPaneTLDTDTextfield.setBounds(482, 99, 167, 20);
723         editProjectPane.add(editProjectPaneTLDTDTextfield);
724 
725         JLabel editProjectPaneTLUC = new JLabel("Total lots under construction:  ");
726         editProjectPaneTLUC.setFont(new Font("Times New Roman", Font.BOLD, 12));
727         editProjectPaneTLUC.setBounds(316, 132, 167, 16);
728         editProjectPane.add(editProjectPaneTLUC);
729 
730         JTextField editProjectPaneTLUCTextfield = new JTextField();
731         editProjectPaneTLUCTextfield.setColumns(10);
732         editProjectPaneTLUCTextfield.setBounds(482, 130, 167, 20);
733         editProjectPane.add(editProjectPaneTLUCTextfield);
734 
735         JLabel editProjectPaneTLR = new JLabel("Total lots remaining:   ");
736         editProjectPaneTLR.setFont(new Font("Times New Roman", Font.BOLD, 12));
737         editProjectPaneTLR.setBounds(316, 163, 167, 16);
738         editProjectPane.add(editProjectPaneTLR);
```

```
739
740         JTextField editProjectPaneTLRTextfield = new JTextField();
741         editProjectPaneTLRTextfield.setColumns(10);
742         editProjectPaneTLRTextfield.setBounds(482, 161, 167, 20);
743         editProjectPane.add(editProjectPaneTLRTextfield);
744
745         JLabel editProjectPaneLSWTLTD = new JLabel("Lots sold within the last 30 days: ");
746         editProjectPaneLSWTLTD.setFont(new Font("Times New Roman", Font.BOLD, 12));
747         editProjectPaneLSWTLTD.setBounds(316, 193, 195, 16);
748         editProjectPane.add(editProjectPaneLSWTLTD);
749
750         JTextField editProjectPaneLSWTLTDTextfield = new JTextField();
751         editProjectPaneLSWTLTDTextfield.setColumns(10);
752         editProjectPaneLSWTLTDTextfield.setBounds(516, 191, 133, 20);
753         editProjectPane.add(editProjectPaneLSWTLTDTextfield);
754
755         JLabel editProjectPaneACPL = new JLabel("Average cost per lot:                $");
756         editProjectPaneACPL.setFont(new Font("Times New Roman", Font.BOLD, 12));
757         editProjectPaneACPL.setBounds(316, 225, 195, 16);
758         editProjectPane.add(editProjectPaneACPL);
759
760         JTextField editProjectPaneACPLTextfield = new JTextField();
761         editProjectPaneACPLTextfield.setColumns(10);
762         editProjectPaneACPLTextfield.setBounds(482, 222, 167, 20);
763         editProjectPane.add(editProjectPaneACPLTextfield);
764
765         JLabel editProjectPaneAPPL = new JLabel("Average sales price per lot:     $");
766         editProjectPaneAPPL.setFont(new Font("Times New Roman", Font.BOLD, 12));
767         editProjectPaneAPPL.setBounds(316, 250, 195, 16);
768         editProjectPane.add(editProjectPaneAPPL);
769
770         JTextField editProjectPaneAPPLFextfield = new JTextField();
771         editProjectPaneAPPLFextfield.setColumns(10);
772         editProjectPaneAPPLFextfield.setBounds(482, 247, 167, 20);
773         editProjectPane.add(editProjectPaneAPPLFextfield);
774
775         JButton editProjectPaneSaveButton = new JButton("Save");
776         editProjectPaneSaveButton.setBounds(634, 404, 98, 26);
777         editProjectPane.add(editProjectPaneSaveButton);
778
779
780
781 /*
782  *
783  * B A C K G R O U N D   P R O C E S S E S
784  *
785  */
786     //ACCOUNT CREATOR
787         //reads white list file and prints a list similar to the file's list
788         FileReader accountWhitelist = new FileReader("accountWhitelist.txt");
789         int charReader;
790         int count=0;
791         String[] accountStringList = new String[10];
792         for(int x=0;x<10;x++) accountStringList[x]="";
793         while ((charReader=accountWhitelist.read()) != -1) { //accountWhiteList is entirely
    read
794             if((char)charReader != ';')
```

```
795                 accountStringList[count]+=(char)charReader; //non semi colons are read and
    added
796             if((char)charReader == ';') { //semi colons indicate the account has been
    completely read
797                 count++;} //changes the accountStringList index
798         }
799         accountWhitelist.close();
800 
801         //creates the new Account classes
802         Account[] account = new Account[10];
803         for(int x=0;x<10;x++) {
804             StringTokenizer st = new StringTokenizer(accountStringList[x],":"); //splits each
    account's line using the colon delimiter
805             String[] items=new String[4];
806             for(int xx=0;xx<=3;xx++) items[xx]=st.nextToken(); //adds each data point to the
    array
807             account[x] = new
    Account(Integer.parseInt(items[0].trim()),items[1],items[2],items[3]);
808         }
809 
810         //creates the event stack
811         EventScripter.buildStackFromEventLogFile();
812 
813 /*
814  *
815  * C L A S S E S
816  *
817  */
818         class DisplayData{
819             public void projectData() {
820                 //READ DATA
821                 NumberFormat cf = NumberFormat.getCurrencyInstance();
822                 //^creates a currency number formatter for financial data values
823                 String input = lblProjectTitle.getText();
824                 StringTokenizer st = new StringTokenizer(input);
825                 String nameBuilder="";
826                 while(st.hasMoreTokens()) nameBuilder+=st.nextToken();
827                 //^creates the project's data name through tokenizing the title label
828                 File projectDataFile = new File(nameBuilder+"Data.txt");
829                 Scanner reader=null;
830                     try {reader = new Scanner(projectDataFile);}
831                     catch (FileNotFoundException e) {e.printStackTrace();}
832                 reader.nextLine();
833                 //^creates a scanner for the project's data file
834 
835                 for(int x=0;x<=7;x++) projectData[x].setText(reader.nextLine());
836                 //^the first seven data points are directly converted into labels
837 
838                 double totalRemaining = (Integer.parseInt(projectData[5].getText()) +
839                                          Integer.parseInt(projectData[6].getText()));
840                 //^calculates a total remaining value, used in future calculations
841 
842                 int selloutInteger = (int) Math.ceil(totalRemaining /
843                                           Integer.parseInt(projectData[7].getText()));
844                 //^calculates the maximum number of months needed to completely sell out
845                 String selloutString =""+selloutInteger;
846                 projectData[8].setText(selloutString+" months");
```

```
847                 //^converts calculated number into a string and into a label
848
849                 double[] costsUnformatted = new double[2];
850                 for(int x=9;x<=10;x++){ //reads specifically 2 lines
851                     double value = Double.parseDouble(reader.nextLine());
852                     //^reads two lines
853                     costsUnformatted[x-9]=value;
854                     projectData[x].setText(""+cf.format(value));
855                     //^converts the data into currencies and puts them into labels
856                 }
857
858                 double revenue = totalRemaining * costsUnformatted[1];
859                 projectData[11].setText(""+cf.format(revenue));
860                 //^calculates total revenue and places it in a label
861
862                 double profit = costsUnformatted[1] - costsUnformatted[0];
863                 double totalProfit = profit * totalRemaining;
864                 projectData[12].setText(""+cf.format(totalProfit));
865                 //^subtracts costs from revenue to calculate profit; places in label
866
867                 double percentDeveloped = (Double.parseDouble(projectData[4].getText()) /
868                                            Double.parseDouble(projectData[3].getText()))*100;
869                 percentDeveloped = Math.round(percentDeveloped*100)/100;
870                 projectData[13].setText("("+(int)percentDeveloped+"%)");
871                 //^calculates the percentage of properties that are in development
872
873                 double percentUnderConstruction =
    (Double.parseDouble(projectData[5].getText()) /
874
    Double.parseDouble(projectData[3].getText()))*100;
875                 percentUnderConstruction = Math.round(percentUnderConstruction*100)/100;
876                 projectData[14].setText("("+(int)percentUnderConstruction+"%)");
877                 //^calculates the percentage of properties that are under construction
878
879                 double percentRemaining = (Double.parseDouble(projectData[6].getText()) /
880                                            Double.parseDouble(projectData[3].getText()))*100;
881                 percentRemaining = Math.round(percentRemaining*100)/100;
882                 projectData[15].setText("("+(int)percentRemaining+"%)");
883                 //^calculates the percentage of properties that are remaining
884
885                 reader.close();
886
887                 //FIND PARTNERS INVOLVED
888                 File projectAccountFile = new File(nameBuilder+"Accounts.txt");
889                 Scanner r1=null;
890                     try {r1 = new Scanner(projectAccountFile);}
891                     catch (FileNotFoundException e) {e.printStackTrace();}
892                 //^creates a new scanner in the project's account file
893                 ArrayList<String> accountNames = new ArrayList<String>();
894                 while(r1.hasNextLine()) accountNames.add(r1.nextLine());
895                 //^makes an ArrayList containing all the accounts involved in the project
896                 int JLabelCounter = 0;
897                 for(int x=0;x<accountNames.size();x++) {
898                     partnersInvoled[x].setText("- "+accountNames.get(x));
899                     partnersInvoled[x].setVisible(true);
900                     detailsPanel.add(partnersInvoled[x]);
901                     JLabelCounter++;
```

```
902                     //^makes partnersInvoled labels only visible for the needed partners
903                 }
904                 for(int x = JLabelCounter;x<10;x++){
905                     partnersInvoled[x].setVisible(false);
906                     //^ensures the rest are reset and non-visible
907                 }
908                 r1.close();
909             }
910 
911             public void editProjectData(){
912                 StringTokenizer st = new StringTokenizer(lblProjectTitle.getText());
913                 String nameBuilder="";
914                 while(st.hasMoreTokens()) nameBuilder+=st.nextToken();
915                 //^builds the project's name based on the project's title label
916 
917                 File projectDataFile = new File(nameBuilder+"Data.txt");
918                 Scanner r1=null;
919                     try {r1 = new Scanner(projectDataFile);}
920                     catch (FileNotFoundException e) {e.printStackTrace();}
921                 //^makes a scanner for the project's data file
922                 editProjectPaneProjectNameTextfield.setText(r1.nextLine());
923                 editProjectPaneAdressTextfield.setText(r1.nextLine());
924                 editProjectPaneCityTextfield.setText(r1.nextLine());
925                 editProjectPaneStateTextfield.setText(r1.nextLine());
926                 editProjectPaneTLIDTextfield.setText(r1.nextLine());
927                 editProjectPaneTLDTDTextfield.setText(r1.nextLine());
928                 editProjectPaneTLUCTextfield.setText(r1.nextLine());
929                 editProjectPaneTLRTextfield.setText(r1.nextLine());
930                 editProjectPaneLSWTLTDTextfield.setText(r1.nextLine());
931                 editProjectPaneACPLTextfield.setText(r1.nextLine());
932                 editProjectPaneAPPLFextfield.setText(r1.nextLine());
933                 //^transfers each line in the file to the according textfield
934                 r1.close();
935 
936                 editProjectPane.setVisible(true);
937                 projectPane.setVisible(false);
938                 //^sets the edit project pane visible, after the textfields have been filled
939             }
940         } DisplayData displayData = new DisplayData();
941 
942 
943         //class used to clear JPanels and their components, such as text boxes and check
    boxes
944         class Clear {
945             public void loginPage() {
946                 passwordField.setText("");
947                 lblIncorrectUandP.setVisible(false);
948             }
949             public void newProjectPage() {
950                 projectNameTextfield.setText("");
951                 adressTextfield.setText("");
952                 cityTextfield.setText("");
953                 stateTextfield.setText("");
954                 totalLotsInDevTextfield.setText("");
955                 lblTotalDevelopedTextfield.setText("");
956                 totalLotsUnderConstructionTextfield.setText("");
957                 totalLotsRemainingTextfield.setText("");
```

```
958                 lotsSoldWithinTextfield.setText("");
959                 averageCostPerTextfield.setText("");
960                 averagePricePerTextfield.setText("");
961                 //^clears all the text fields
962                 for(JCheckBox x : chckbxAccount) x.setSelected(false);
963                 //^clears all the partner involved checkboxes
964             }
965             public void newTasks(){
966                 projCompDDTextField.setText("");
967                 projCompMMTextField.setText("");
968                 projCompYYYYTextField.setText("");
969                 contractorTextField.setText("");
970                 tasksComboBox.setSelectedItem(" ");
971             }
972             public void newProblems(){
973                 titleTextField.setText("");
974                 descriptionTextField.setText("");
975                 priorityComboBox.setSelectedItem("1");
976             }
977         } /*Creates an instance of the Clear class for the rest of the program*/ Clear clear
    = new Clear();
978
979         JButton[] aryBtnProject = new JButton[15];
980         for(int x=0;x<15;x++) {
981             aryBtnProject[x] = new JButton();
982             aryBtnProject[x].setText("BUTTON");
983             aryBtnProject[x].setFont(new Font("Times New Roman", Font.BOLD, 14));
984             aryBtnProject[x].setBounds(44, (174+(35*x)), 200, 20);
985             aryBtnProject[x].setVisible(true);
986         }
987
988         class ProjectListGenerator{
989             public void createProjectList() {
990                 for(int x=0;x<listOfProjectButtons.length;x++){ //resets all buttons
991                     listOfProjectButtons[x].setText("EMPTY");
992                     listOfProjectButtons[x].setBounds(30, (170+(30*x)), 150, 25);
993                     mainMenuPane.add(listOfProjectButtons[x]);
994                     listOfProjectButtons[x].setVisible(false);
995                 }
996
997                 String[] projectArray = null;
998                 try {
999                     projectArray =
    RandomAccessFileEditor.getProjects("ListOfProjectsRAF.txt");
1000                    //Recursively reads the ListOfProjectsRAF file
1001                }
1002                catch (Exception e1) {
1003                    e1.printStackTrace();
1004                }
1005                int count=0;
1006                for(int i=0;i<projectArray.length;i++) { //reads each line of the
    ListOfProjects.txt
1007                    String projectFileName = projectArray[i];
1008                    File accountFile = new File(projectFileName+"Accounts.txt");
1009                    Scanner r2=null;
1010                        try { //creates a scanner to search the accounts of each project in
    the ListOfProjects.txt
```

```
1011                            r2 = new Scanner(accountFile);
1012                        } catch (FileNotFoundException e) {
1013                            e.printStackTrace();
1014                        }
1015                    ArrayList<String> listOfAccounts = new ArrayList<String>();
1016                    while(r2.hasNext()) listOfAccounts.add(r2.nextLine()); //makes a list of
    accounts per project
1017                    r2.close();
1018                    for(int x=0;x<listOfAccounts.size();x++) {
1019                        if(listOfAccounts.get(x).equals(account[accountIndexUsed].getName()))
    {
1020                            //^checks if any of the names in the ArrayList match the logged
    in client
1021                            File dataFile = new File(projectFileName+"Data.txt");
1022                            Scanner r3=null;
1023                                try { //creates a scanner for the project's data file
1024                                    r3 = new Scanner(dataFile);
1025                                } catch (FileNotFoundException e) {
1026                                    e.printStackTrace();
1027                                }
1028                            String projectName = r3.nextLine(); //locates the project's name
1029                            r3.close();
1030                            listOfProjectButtons[count].setText(projectName);
1031                            listOfProjectButtons[count].setVisible(true); //creates a button
    for the respective project
1032                            count++;
1033                        }else {
1034                            listOfProjectButtons[count].setVisible(false);
1035                        }
1036                    }
1037                }
1038            }
1039        }  ProjectListGenerator projectListMaker = new ProjectListGenerator();
1040
1041
1042        //creates the login system
1043        class Login{
1044            public void runLogin() {
1045                for(int x=0;x<10;x++){
1046                    if(account[x].getUsername().equals(usernameField.getText()) &&
    account[x].getPassword().equals(
1047                            //generates the hash code for the inputed password
1048                            PasswordHasher.generateHash(new
    String(passwordField.getPassword())))) {
1049                        lblIncorrectUandP.setVisible(false);
1050                        loginPane.setVisible(false);
1051                        mainMenuPane.setVisible(true); //changes panels visible in the card
    layout
1052                        accountIndexUsed = x; //records the index of the logged in client
1053                        projectListMaker.createProjectList(); //creates the list of projects
    in the main menu
1054                        lblWelcome.setText("Welcome " + account[accountIndexUsed].getName());
    //builds the welcome JLabel
1055                        }
1056                    else lblIncorrectUandP.setVisible(true);
1057            }}
1058        } Login login = new Login(); //creates an instance of the login class
```

```
1059
1060        abstract class TableManager implements DataTable{
1061            public String getFileName(){
1062                StringTokenizer st = new StringTokenizer(lblProjectTitle.getText()," ");
1063                String fileName="";
1064                while(st.hasMoreTokens()) fileName+=st.nextToken();
1065                return fileName;
1066            }
1067            public void generateData(){}
1068            public void addData() {}
1069        }
1070
1071        //creates the table for the tasks page and adds new values to the table
1072        class TasksTable extends TableManager implements DataTable{
1073            public void generateData() {
1074
1075                //removes the preexisting table from another project
1076                tasksTableModel.setRowCount(0);
1077
1078                File csvFile = new File(super.getFileName()+"Tasks.csv");
1079                Scanner reader=null;
1080                    try {reader = new Scanner(csvFile);}
1081                    catch (FileNotFoundException e) {e.printStackTrace();}
1082
1083                //creates the data ArrayList
1084                ArrayList<String> dataList = new ArrayList<String>();
1085
1086                //reads through the header values (they have already been set and only...
1087                //...exist in the text file for exporting reasons)
1088                reader.nextLine();
1089
1090                //reads the actual task data
1091                while(reader.hasNextLine()) dataList.add(reader.nextLine());
1092
1093                //converts from an ArrayList to a normal array
1094                String[] dataArray = new String[dataList.size()];
1095                for(int x=0;x<dataArray.length;x++) dataArray[x] = dataList.get(x);
1096
1097                //adds the data to the JTable
1098                for(int x=0;x<dataArray.length;x++){
1099                    String[] rowData = new String[3];
1100                    int rowIndex = 0;
1101                    StringTokenizer st2 = new StringTokenizer(dataArray[x], ",");
1102                    while(st2.hasMoreTokens()) {
1103                        rowData[rowIndex] = st2.nextToken();
1104                        rowIndex++;}
1105                    tasksTableModel.addRow(rowData); //uses defaultTableModel
1106                }
1107                reader.close();
1108            }
1109
1110            public void addData() {
1111                //gets the project's name
1112                StringTokenizer st = new StringTokenizer(lblProjectTitle.getText()," ");
1113                String fileName="";
1114                while(st.hasMoreTokens()) fileName+=st.nextToken();
1115
```

```
1116                //makes the file and filewriter and printwriter
1117                File csvFile = new File(fileName+"Tasks.csv");
1118                FileWriter fileWriter = null;
1119                    try {fileWriter = new FileWriter(csvFile, true);}
1120                    catch (IOException e1) {e1.printStackTrace();}
1121                PrintWriter printWriter = new PrintWriter(fileWriter);
1122
1123                //writes the new task to the text file
1124                String task = (String) tasksComboBox.getSelectedItem();
1125                String contractor = contractorTextField.getText();
1126                String date = projCompMMTextField.getText() +"/"+
1127                              projCompDDTextField.getText() +"/"+
1128                              projCompYYYYTextField.getText();
1129                printWriter.print("\r\n");
1130                printWriter.print(task +", "+ contractor +", "+ date);
1131                printWriter.close();
1132
1133                //adds the data to the JTable
1134                String[] rowData = {task, contractor, date};
1135                tasksTableModel.addRow(rowData);
1136
1137            }
1138        } DataTable taskTable = new TasksTable();
1139
1140        class CommentBoard{
1141            public void generateCommentBoard() {
1142                //creates array lists of the ID's of the accounts and the comments themselves
1143                ArrayList<Integer> indexArray = new ArrayList<Integer>();
1144                ArrayList<String> commentArray = new ArrayList<String>();
1145
1146                //reads the text file
1147                StringTokenizer st = new StringTokenizer(lblProjectTitle.getText()," ");
1148                String fileName="";
1149                while(st.hasMoreTokens()) fileName+=st.nextToken();
1150                File commentsFile = new File(fileName+"Comments.txt");
1151                Scanner reader=null;
1152                    try {reader = new Scanner(commentsFile);}
1153                    catch (FileNotFoundException e) {e.printStackTrace();}
1154                //^makes a scanner for the project's comments file
1155                while(reader.hasNextLine()) {
1156                    //each line in the text file has an ID and a comment
1157                    StringTokenizer st2 = new StringTokenizer(reader.nextLine(),";");
1158                    //^account indexes and comments are split by the semi-colon delimiter
1159                    indexArray.add(Integer.parseInt(st2.nextToken()));
1160                    commentArray.add(st2.nextToken());
1161                }
1162
1163                //erases previous comments from other opened projects
1164                for(int x=0;x<numberOfCommentsSupported;x++) {
1165                    nameLabelArray[x].setText("");
1166                    nameLabelArray[x].setVisible(false);
1167                    commentLabelArray[x].setText("");
1168                    commentLabelArray[x].setVisible(false);
1169                }
1170
1171                //converts the information from the array lists into JLabels
1172                for(int x=0;x<indexArray.size();x++) {
```

```
1173                    nameLabelArray[x].setText(""+account[indexArray.get(x)].getName());
1174                    //^finds the respective name of the account index recorded
1175                    nameLabelArray[x].setVisible(true);
1176                    commentLabelArray[x].setText(""+commentArray.get(x));
1177                    commentLabelArray[x].setToolTipText(""+commentArray.get(x));
1178                    commentLabelArray[x].setVisible(true);}
1179            }
1180            public void addComment() {
1181                //adds the new comment to the comment text file
1182                StringTokenizer st = new StringTokenizer(lblProjectTitle.getText()," ");
1183                String fileName="";
1184                while(st.hasMoreTokens()) fileName+=st.nextToken();
1185                //^builds the project's name based on the project's title label
1186                File commentsFile = new File(fileName+"Comments.txt");
1187                FileWriter dataWriter = null;
1188                    try {dataWriter = new FileWriter(commentsFile, true);} //supports
    amending
1189                    catch (IOException e1) {e1.printStackTrace();}
1190                PrintWriter dpw = new PrintWriter(dataWriter);
1191                //^creates a File and PrintWriter for the project's comments file
1192                dpw.print(""+account[accountIndexUsed].getIndex()+";"+commentTextArea.getText
    ());
1193                //^prints the client's index and the comment itself, using the semi-colon
    delimiter
1194                dpw.println("");
1195                dpw.close();
1196                commentTextArea.setText("");
1197                //^resets the comment's textarea without invoking a Clear method
1198
1199                generateCommentBoard(); //regenerates the comment board by invoking the
    method
1200            }
1201        } CommentBoard commentBoard = new CommentBoard();
1202
1203        //creates the table for the tasks page and adds new values to the table
1204        class ProblemsTable extends TableManager implements DataTable{
1205
1206            private PQueue problemQueue;
1207
1208            public void generateProblemQueue() {
1209                //removes the preexisting table from another project
1210                problemsTableModel.setRowCount(0);
1211
1212                //reads the tasks csv file
1213                File csvFile = new File(super.getFileName()+"Problems.csv");
1214                Scanner reader=null;
1215                    try {reader = new Scanner(csvFile);}
1216                    catch (FileNotFoundException e) {e.printStackTrace();}
1217
1218                //creates the data ArrayList
1219                ArrayList<String> dataList = new ArrayList<String>();
1220
1221                //reads through the header values (they have already been set and only...
1222                //...exist in the text file for exporting reasons)
1223                reader.nextLine();
1224
1225                //reads the actual task data
```

```
1226                while(reader.hasNextLine()) dataList.add(reader.nextLine());
1227
1228                //converts from an ArrayList to a normal array
1229                String[] dataArray = new String[dataList.size()];
1230                for(int x=0;x<dataArray.length;x++) dataArray[x] = dataList.get(x);
1231
1232                //creates an array of Problem instances
1233                problemQueue = new PQueue(dataList.size()*3);
1234
1235                //adds the data to the Problem Queue
1236                for(int x=0;x<dataArray.length;x++){
1237                    String[] rowData = new String[3];
1238                    int rowIndex = 0;
1239                    StringTokenizer st2 = new StringTokenizer(dataArray[x], ",");
1240                    String title = st2.nextToken();
1241                    String description = st2.nextToken();
1242                    int priority = Integer.parseInt(st2.nextToken());
1243                    problemQueue.enqueue(new Problem(title, description, priority));
1244                }
1245                reader.close();
1246            }
1247
1248            public void generateData() {
1249
1250                generateProblemQueue();
1251
1252                //adds the data to the JTable
1253                while(!problemQueue.isEmpty()){
1254                    Problem temp = problemQueue.dequeue();
1255                    String[] rowData = new String[3];
1256                    rowData[0] = temp.getTitle();
1257                    rowData[1] = temp.getDescription();
1258                    rowData[2] = ""+temp.getPriority();
1259                    problemsTableModel.addRow(rowData); //uses defaultTableModel
1260                }
1261            }
1262
1263            public void addData() {
1264                //makes the file and filewriter and printwriter
1265                File csvFile = new File(super.getFileName()+"Problems.csv");
1266                FileWriter fileWriter = null;
1267                    try {fileWriter = new FileWriter(csvFile, true);}
1268                    catch (IOException e1) {e1.printStackTrace();}
1269                PrintWriter printWriter = new PrintWriter(fileWriter);
1270
1271                //writes the new task to the text file
1272                String title = (String) titleTextField.getText();
1273                String description = (String) descriptionTextField.getText();
1274                int priority = Integer.parseInt((String) priorityComboBox.getSelectedItem());
1275                printWriter.print("\r\n");
1276                printWriter.print(title +","+ description +","+ priority);
1277                printWriter.close();
1278
1279                //adds the data to the JTable simply by adding to the row (not by actually
    enqueue/dequeue'ing the queue)
1280                //String[] rowData = {title, description, ""+priority};
1281                //problemsTableModel.addRow(rowData);
```

```
1282
1283                //adds the data to the JTable by actually enqueue/dequeue'ing the queue
1284                generateData();
1285            }
1286
1287            public Problem removeProblem(){
1288                //makes the altered queue
1289                generateProblemQueue();
1290                Problem removedProblem = problemQueue.dequeue();
1291
1292                //makes the file
1293                File csvFile = new File(super.getFileName()+"Problems.csv");
1294
1295                //deletes the file's contents
1296                PrintWriter writer = null;
1297                    try {writer = new PrintWriter(csvFile);}
1298                    catch (FileNotFoundException e) {e.printStackTrace();}
1299                writer.print("");
1300
1301                //makes the file editor classes
1302                FileWriter fileWriter = null;
1303                    try {fileWriter = new FileWriter(csvFile, true);}
1304                    catch (IOException e1) {e1.printStackTrace();}
1305                PrintWriter printWriter = new PrintWriter(fileWriter);
1306
1307                //sets up header
1308                printWriter.print("Title,Description,Priority");
1309
1310                //prints the new queue
1311                while(!problemQueue.isEmpty()) {
1312                    Problem temp = problemQueue.dequeue();
1313                    printWriter.print("\r\n");
1314                    printWriter.print(temp.getTitle()+","+temp.getDescription()+","+temp.getP
    riority());
1315                }
1316                printWriter.close();
1317
1318                return removedProblem;
1319            }
1320
1321        } ProblemsTable problemTable = new ProblemsTable();
1322
1323
1324        class ChangeChecker{
1325            public String originalProjectPaneProjectNameTextfield;
1326            public String originalProjectPaneAdressTextfield;
1327            public String originalProjectPaneCityTextfield;
1328            public String originalProjectPaneStateTextfield;
1329            public String originalProjectPaneTLIDTextfield;
1330            public String originalProjectPaneTLDTDTextfield;
1331            public String originalProjectPaneTLUCTextfield;
1332            public String originalProjectPaneTLRTextfield;
1333            public String originalProjectPaneLSWTLTDTextfield;
1334            public String originalProjectPaneACPLTextfield;
1335            public String originalProjectPaneAPPLFextfield;
1336
1337            public void captureOriginalDataValues() {
```

```
1338                originalProjectPaneProjectNameTextfield =
    editProjectPaneProjectNameTextfield.getText();
1339                originalProjectPaneAdressTextfield =
    editProjectPaneAdressTextfield.getText();
1340                originalProjectPaneCityTextfield = editProjectPaneCityTextfield.getText();
1341                originalProjectPaneStateTextfield = editProjectPaneStateTextfield.getText();
1342                originalProjectPaneTLIDTextfield = editProjectPaneTLIDTextfield.getText();
1343                originalProjectPaneTLDTDTextfield = editProjectPaneTLDTDTextfield.getText();
1344                originalProjectPaneTLUCTextfield = editProjectPaneTLUCTextfield.getText();
1345                originalProjectPaneTLRTextfield = editProjectPaneTLRTextfield.getText();
1346                originalProjectPaneLSWTLTDTextfield =
    editProjectPaneLSWTLTDTextfield.getText();
1347                originalProjectPaneACPLTextfield = editProjectPaneACPLTextfield.getText();
1348                originalProjectPaneAPPLFextfield = editProjectPaneAPPLFextfield.getText();
1349            }
1350        } ChangeChecker checker = new ChangeChecker();
1351
1352        /*class ProjectClassCreator{
1353            public void createProjectClasses() {
1354                Scanner r1 = new Scanner("listOfProjects.txt");
1355                int c=0;
1356                while(r1.hasNext()) {
1357                    String projectNameFromFile=r1.next();
1358                    boolean projectPreviouslyCreated = false;
1359                    for(int x=0;x<listOfProjects.size();x++) {
1360                        StringTokenizer st = new
    StringTokenizer(listOfProjects.get(x).getName()," ");
1361                        String projectNameFromArray = "";
1362                        while(st.hasMoreTokens()) projectNameFromArray+=st.nextToken();
1363                        //System.out.println("Array: "+projectNameFromArray+" File:
    "+projectNameFromFile);
1364                        if (projectNameFromFile.equals(projectNameFromArray))
    projectPreviouslyCreated=true;
1365                    }
1366                    if(projectPreviouslyCreated==false) {
1367                        Scanner r2 = new Scanner(projectNameFromFile+"Data.txt");
1368                        listOfProjects.add(new Project(r2.nextLine()));
1369                        listOfProjects.get(c).setAddress(r2.nextLine());
1370                        listOfProjects.get(c).setCity(r2.nextLine());
1371                        listOfProjects.get(c).setState(r2.nextLine());
1372                        listOfProjects.get(c).setTotalLotsInDevelopment(r2.nextInt());
1373                        listOfProjects.get(c).setTotalLotsDeveloped(r2.nextInt());
1374                        listOfProjects.get(c).setTotalLotsUnderConstruction(r2.nextInt());
1375                        listOfProjects.get(c).setTotalLotsRemaining(r2.nextInt());
1376                        listOfProjects.get(c).setTotalLotsSoldIn30Days(r2.nextInt());
1377                        listOfProjects.get(c).setAvgCostPerLot(r2.nextInt());
1378                        listOfProjects.get(c).setAvgPricePerLot(r2.nextInt());
1379                        r2.close();
1380                        c++;
1381                    }
1382                }
1383                r1.close();
1384                System.out.println("Project maker has been run");
1385        }} ProjectClassCreator projecClassMaker = new ProjectClassCreator();*/
1386 /*
1387  *
1388  * A C T I O N   L I S T E N E R S
```

```
1389 *
1390 */
1391         //back buttons
1392         btnBack.addActionListener(new ActionListener() {
1393             public void actionPerformed(ActionEvent e) {
1394                 newProjectPane.setVisible(false);
1395                 mainMenuPane.setVisible(true);
1396                 setTitle("Real Estate Tracker");
1397                 //^removes specific project from the window title
1398                 projectListMaker.createProjectList();
1399                 //^regenerates the client's list of projects
1400             }
1401         });
1402         btnBack1.addActionListener(new ActionListener() {
1403             public void actionPerformed(ActionEvent e) {
1404                 projectPane.setVisible(false);
1405                 mainMenuPane.setVisible(true);
1406                 setTitle("Real Estate Tracker");
1407                 //^removes specific project from the window title
1408                 projectListMaker.createProjectList();
1409                 //^regenerates the client's list of projects
1410             }
1411         });
1412         editProjectPaneEditButton.addActionListener(new ActionListener() {
1413             public void actionPerformed(ActionEvent e) {
1414                 editProjectPane.setVisible(false);
1415                 projectPane.setVisible(true);
1416                 //^returns to the project pane
1417                 displayData.projectData();
1418                 //^refreshes the project data
1419             }
1420         });
1421 
1422         //save edits to project data
1423         editProjectPaneSaveButton.addActionListener(new ActionListener() {
1424             public void actionPerformed(ActionEvent e) {
1425 
1426                 boolean emptyTextFields = false;
1427                 boolean noNumberTextFields = false; //sets all error booleans to false
1428 
1429                 if(editProjectPaneProjectNameTextfield.getText().isEmpty() ||
1430                         editProjectPaneAdressTextfield.getText().isEmpty() ||
1431                         editProjectPaneCityTextfield.getText().isEmpty() ||
1432                         editProjectPaneStateTextfield.getText().isEmpty() ||
1433                         editProjectPaneTLIDTextfield.getText().isEmpty() ||
1434                         editProjectPaneTLDTDTextfield.getText().isEmpty() ||
1435                         editProjectPaneTLUCTextfield.getText().isEmpty() ||
1436                         editProjectPaneTLRTextfield.getText().isEmpty() ||
1437                         editProjectPaneLSWTLTDTextfield.getText().isEmpty() ||
1438                         editProjectPaneACPLTextfield.getText().isEmpty() ||
1439                         editProjectPaneAPPLFextfield.getText().isEmpty()) {
1440                         //^checks if any textfields are left empty
1441                             emptyTextFields =  true;
1442                             System.out.println("ERROR: Empty text fields");
1443                 }
1444 
1445                 if(emptyTextFields==true) {
```

```
1446                    JOptionPane.showMessageDialog(new JFrame(),
1447                            "All textboxes must be filled before revision",
1448                            "ERROR",
1449                            JOptionPane.WARNING_MESSAGE);
1450                    //^runs a warning option pane, if textfield(s) are empty
1451                }
1452
1453                if(emptyTextFields==false) { //runs if all textfields are full
1454                    try {
1455                        Integer.parseInt(editProjectPaneTLIDTextfield.getText());
1456                        Integer.parseInt(editProjectPaneTLDTDTextfield.getText());
1457                        Integer.parseInt(editProjectPaneTLUCTextfield.getText());
1458                        Integer.parseInt(editProjectPaneTLRTextfield.getText());
1459                        Integer.parseInt(editProjectPaneLSWTLTDTextfield.getText());
1460                        Double.parseDouble(editProjectPaneACPLTextfield.getText());
1461                        Double.parseDouble(editProjectPaneAPPLFextfield.getText());
1462                        //^check if certain textfields contain numbers
1463                    }
1464                    catch(NumberFormatException e1) {
1465                    //^runs if certain textfields do not contain numbers
1466                        JOptionPane.showMessageDialog(new JFrame(),
1467                                "Please insert numbers where needed",
1468                                "ERROR",
1469                                JOptionPane.WARNING_MESSAGE);
1470                        noNumberTextFields = true;
1471                        //^runs a warning option pane, if textfield(s) dont have numbers
1472                    }
1473                }
1474
1475                if(noNumberTextFields==false && emptyTextFields==false) {
1476                //^runs if there have been no errors found
1477                    StringTokenizer st = new StringTokenizer(lblProjectTitle.getText());
1478                    String nameBuilder="";
1479                    while(st.hasMoreTokens()) nameBuilder+=st.nextToken();
1480                    //^builds the project's name from the project title label
1481                    File projectDataFile = new File(nameBuilder+"Data.txt");
1482                    FileWriter dataWriter = null;
1483                        try {dataWriter = new FileWriter(projectDataFile);}
1484                        catch (IOException e1) {e1.printStackTrace();}
1485                    PrintWriter dpw = new PrintWriter(dataWriter);
1486                    //^makes a File and PrintWriter for the project's data file
1487
1488                    String newProjectPaneProjectNameTextfield =
    editProjectPaneProjectNameTextfield.getText();
1489                        if(!newProjectPaneProjectNameTextfield.equals(checker.originalProject
    PaneProjectNameTextfield)) {
1490                            DataEvent temp = new
    DataEvent(account[accountIndexUsed].getName(), lblProjectTitle.getText(),
1491                                    "Project Name",checker.originalProjectPaneACPLTextfield,
1492                                    newProjectPaneProjectNameTextfield);
1493                            EventScripter.addEvent(temp);
1494                        }
1495                    String newProjectPaneAdressTextfield =
    editProjectPaneAdressTextfield.getText();
1496                    if(!newProjectPaneAdressTextfield.equals(checker.originalProjectPaneAdres
    sTextfield)) {
1497                        DataEvent temp = new DataEvent(account[accountIndexUsed].getName(),
```

```
    lblProjectTitle.getText(),
1498                                "Project Address",checker.originalProjectPaneACPLTextfield,
1499                                newProjectPaneAdressTextfield);
1500                        EventScripter.addEvent(temp);
1501                    }
1502                    String newProjectPaneCityTextfield =
    editProjectPaneCityTextfield.getText();
1503                    if(!newProjectPaneCityTextfield.equals(checker.originalProjectPaneCityTex
    tfield)) {
1504                        DataEvent temp = new DataEvent(account[accountIndexUsed].getName(),
    lblProjectTitle.getText(),
1505                                "Project City",checker.originalProjectPaneCityTextfield,
1506                                newProjectPaneCityTextfield);
1507                        EventScripter.addEvent(temp);
1508                    }
1509                    String newProjectPaneStateTextfield =
    editProjectPaneStateTextfield.getText();
1510                    if(!newProjectPaneStateTextfield.equals(checker.originalProjectPaneStateT
    extfield)) {
1511                        DataEvent temp = new DataEvent(account[accountIndexUsed].getName(),
    lblProjectTitle.getText(),
1512                                "Project State",checker.originalProjectPaneStateTextfield,
1513                                newProjectPaneStateTextfield);
1514                        EventScripter.addEvent(temp);
1515                    }
1516                    String newProjectPaneTLIDTextfield =
    editProjectPaneTLIDTextfield.getText();
1517                    if(!newProjectPaneTLIDTextfield.equals(checker.originalProjectPaneTLIDTex
    tfield)) {
1518                        DataEvent temp = new DataEvent(account[accountIndexUsed].getName(),
    lblProjectTitle.getText(),
1519                                "Total Lots in
    Development",checker.originalProjectPaneTLIDTextfield,
1520                                newProjectPaneTLIDTextfield);
1521                        EventScripter.addEvent(temp);
1522                    }
1523                    String newProjectPaneTLDTDTextfield =
    editProjectPaneTLDTDTextfield.getText();
1524                    if(!newProjectPaneTLDTDTextfield.equals(checker.originalProjectPaneTLDTDT
    extfield)) {
1525                        DataEvent temp = new DataEvent(account[accountIndexUsed].getName(),
    lblProjectTitle.getText(),
1526                                "Total Lots Developed to
    Date",checker.originalProjectPaneTLDTDTextfield,
1527                                newProjectPaneTLDTDTextfield);
1528                        EventScripter.addEvent(temp);
1529                    }
1530                    String newProjectPaneTLUCTextfield =
    editProjectPaneTLUCTextfield.getText();
1531                    if(!newProjectPaneTLUCTextfield.equals(checker.originalProjectPaneTLUCTex
    tfield)) {
1532                        DataEvent temp = new DataEvent(account[accountIndexUsed].getName(),
    lblProjectTitle.getText(),
1533                                "Total Lots Under
    Construction",checker.originalProjectPaneTLUCTextfield,
1534                                newProjectPaneTLUCTextfield);
1535                        EventScripter.addEvent(temp);
```

```
1536                    }
1537                    String newProjectPaneTLRTextfield =
    editProjectPaneTLRTextfield.getText();
1538                    if(!newProjectPaneTLRTextfield.equals(checker.originalProjectPaneTLRTextf
    ield)) {
1539                        DataEvent temp = new DataEvent(account[accountIndexUsed].getName(),
    lblProjectTitle.getText(),
1540                                "Total Lots
    Remaining",checker.originalProjectPaneTLRTextfield,
1541                                newProjectPaneTLRTextfield);
1542                        EventScripter.addEvent(temp);
1543                    }
1544                    String newProjectPaneLSWTLTDTextfield =
    editProjectPaneLSWTLTDTextfield.getText();
1545                    if(!newProjectPaneLSWTLTDTextfield.equals(checker.originalProjectPaneLSWT
    LTDTextfield)) {
1546                        DataEvent temp = new DataEvent(account[accountIndexUsed].getName(),
    lblProjectTitle.getText(),
1547                                "Total Lots Sold Within the Last 30
    Days",checker.originalProjectPaneLSWTLTDTextfield,
1548                                newProjectPaneLSWTLTDTextfield);
1549                        EventScripter.addEvent(temp);
1550                    }
1551                    String newProjectPaneACPLTextfield =
    editProjectPaneACPLTextfield.getText();
1552                    if(!newProjectPaneACPLTextfield.equals(checker.originalProjectPaneACPLTex
    tfield)) {
1553                        DataEvent temp = new DataEvent(account[accountIndexUsed].getName(),
    lblProjectTitle.getText(),
1554                                "Average Cost Per
    Lot",checker.originalProjectPaneACPLTextfield,
1555                                newProjectPaneACPLTextfield);
1556                        EventScripter.addEvent(temp);
1557                    }
1558                    String newProjectPaneAPPLFextfield =
    editProjectPaneAPPLFextfield.getText();
1559                    if(!newProjectPaneAPPLFextfield.equals(checker.originalProjectPaneAPPLFex
    tfield)) {
1560                        DataEvent temp = new DataEvent(account[accountIndexUsed].getName(),
    lblProjectTitle.getText(),
1561                                "Average Cost Per
    Lot",checker.originalProjectPaneAPPLFextfield,
1562                                newProjectPaneAPPLFextfield);
1563                        EventScripter.addEvent(temp);
1564                    }
1565
1566                    dpw.println(editProjectPaneProjectNameTextfield.getText());
1567                    dpw.println(editProjectPaneAdressTextfield.getText());
1568                    dpw.println(editProjectPaneCityTextfield.getText());
1569                    dpw.println(editProjectPaneStateTextfield.getText());
1570                    dpw.println(editProjectPaneTLIDTextfield.getText());
1571                    dpw.println(editProjectPaneTLDTDTextfield.getText());
1572                    dpw.println(editProjectPaneTLUCTextfield.getText());
1573                    dpw.println(editProjectPaneTLRTextfield.getText());
1574                    dpw.println(editProjectPaneLSWTLTDTextfield.getText());
1575                    dpw.println(editProjectPaneACPLTextfield.getText());
1576                    dpw.println(editProjectPaneAPPLFextfield.getText());
```

```
1577                    //^prints the new data values, replacing the older ones
1578                    dpw.close();
1579
1580                    editProjectPane.setVisible(false);
1581                    projectPane.setVisible(true);
1582                    //^takes the client back to the respective project's detail pane
1583                    displayData.projectData();
1584                    //^reruns the projectData() method to refresh the detail labels
1585                }
1586            }
1587        });
1588
1589        //resets the new project pane
1590        btnReset.addActionListener(new ActionListener() {
1591            public void actionPerformed(ActionEvent arg0) {
1592                Object[] warningOptions={"Return","Reset new project"};
1593                int resetWarning = JOptionPane.showOptionDialog //generates a JOptionPane
    warning the client
1594                        (null, "WARNING\nAre you sure you want to reset your new project?",
1595                                "Warning",JOptionPane.DEFAULT_OPTION,
    JOptionPane.WARNING_MESSAGE,
1596                                null, warningOptions, warningOptions[0]);
1597                if(resetWarning==1) { //runs if "reset" was nevertheless selected
1598                    clear.newProjectPage(); //resets the new project page
1599                }
1600        }});
1601
1602        //login verification system
1603            //submit button
1604                btnSubmit.addActionListener(new ActionListener() {
1605                    public void actionPerformed(ActionEvent arg0){
1606                        login.runLogin();
1607                    }});
1608            //enter button
1609                passwordField.addKeyListener(new KeyAdapter() {
1610                    public void keyPressed(KeyEvent evt){
1611                        if(evt.getKeyCode()==KeyEvent.VK_ENTER) {
1612                            login.runLogin();
1613                        }
1614                    }});
1615
1616
1617        //logout confirmation
1618        btnLogout.addActionListener(new ActionListener() {
1619            public void actionPerformed(ActionEvent arg0) {
1620                int logoutConformation = JOptionPane.showConfirmDialog
1621                        (null, "Are you sure?","Conformation", JOptionPane.YES_NO_OPTION);
1622                if(logoutConformation==0) { //runs if client selects "YES"
1623                    loginPane.setVisible(true);
1624                    mainMenuPane.setVisible(false); //takes client back to the login pane
1625                    clear.loginPage(); //clears the login page
1626                    clear.newProjectPage(); //clear the new project pane
1627                }
1628            }
1629        });
1630
1631        //Takes client to the new project pane
```

```
1632        btnNewProject.addActionListener(new ActionListener() {
1633            public void actionPerformed(ActionEvent e) {
1634                newProjectPane.setVisible(true);
1635                mainMenuPane.setVisible(false); //takes user to the new pane
1636
1637                //renames check boxes to account names
1638                int accountIndexHasBeenMet=0;
1639                for(int x=0;x<account.length;x++) {
1640                    if(x==accountIndexUsed) accountIndexHasBeenMet=1; //checks for the
    client's index
1641                    if(x!=accountIndexUsed){ //skips over the client's index
1642                        if(accountIndexHasBeenMet==0)
    chckbxAccount[x].setText(account[x].getName());
1643                        if(accountIndexHasBeenMet==1)
    chckbxAccount[x-1].setText(account[x].getName());
1644                        //^titles checkboxes in the correct index, based on the if the
    client's index has been met
1645                    }
1646                }
1647            }
1648        });
1649
1650        //takes client to the respective project page
1651        for(int x=0;x<listOfProjectButtons.length;x++){
1652            listOfProjectButtons[x].addActionListener(new ActionListener() {
1653                public void actionPerformed(ActionEvent e) {
1654                    mainMenuPane.setVisible(false);
1655                    projectPane.setVisible(true); //takes user to the respective project
1656                    lblProjectTitle.setText(e.getActionCommand()); //writes the project title
    label
1657                    tabbedPane.setSelectedIndex(0);
1658                    displayData.projectData(); //displays the project's data
1659                    taskTable.generateData(); //generates the task table
1660                    commentBoard.generateCommentBoard(); //generates the comment board
1661                    problemTable.generateData(); //generates the problem table
1662                    setTitle(e.getActionCommand()+" - Real Estate Tracker"); //renames window
    title
1663                }
1664            });
1665        }
1666
1667
1668        //saving a new project
1669        btnSave.addActionListener(new ActionListener() {
1670            public void actionPerformed(ActionEvent e) {
1671                boolean emptyTextFields = false;
1672                boolean noNumberTextFields = false; //sets all error booleans to false
1673
1674                if(projectNameTextfield.getText().isEmpty() ||
1675                    adressTextfield.getText().isEmpty() ||
1676                    cityTextfield.getText().isEmpty() ||
1677                    stateTextfield.getText().isEmpty() ||
1678                    totalLotsInDevTextfield.getText().isEmpty() ||
1679                    lblTotalDevelopedTextfield.getText().isEmpty() ||
1680                    totalLotsUnderConstructionTextfield.getText().isEmpty() ||
1681                    totalLotsRemainingTextfield.getText().isEmpty() ||
1682                    lotsSoldWithinTextfield.getText().isEmpty() ||
```

```
1683                    averageCostPerTextfield.getText().isEmpty() ||
1684                    averagePricePerTextfield.getText().isEmpty()) {
1685                    //^checks if any of the textfields are left empty
1686                        emptyTextFields =  true; //triggers error boolean
1687                        System.out.println("ERROR: Empty text fields");
1688                }
1689
1690                if(emptyTextFields==true) { //runs if textfield(s) left empty
1691                    JOptionPane.showMessageDialog(new JFrame(),
1692                            "All textboxes must be filled before submission",
1693                            "ERROR",
1694                            JOptionPane.WARNING_MESSAGE);
1695                    //^displays an error option pane advising the client to fill all
    textfields
1696                }
1697
1698                if(emptyTextFields==false) { //runs if all textfields contain data
1699                    try {
1700                        Integer.parseInt(totalLotsInDevTextfield.getText());
1701                        Integer.parseInt(lblTotalDevelopedTextfield.getText());
1702                        Integer.parseInt(totalLotsUnderConstructionTextfield.getText());
1703                        Integer.parseInt(totalLotsRemainingTextfield.getText());
1704                        Integer.parseInt(lotsSoldWithinTextfield.getText());
1705                        Double.parseDouble(averageCostPerTextfield.getText());
1706                        Double.parseDouble(averagePricePerTextfield.getText());
1707                        //^checks if certain textfields are numbers
1708                    }
1709                    catch(NumberFormatException e1) {
1710                        JOptionPane.showMessageDialog(new JFrame(),
1711                                "Please insert numbers where needed",
1712                                "ERROR",
1713                                JOptionPane.WARNING_MESSAGE);
1714                    //^displays an error option pane advising the client to fill in numbers
1715                        noNumberTextFields = true; //triggers erorr boolean
1716                    }
1717                }
1718                if(noNumberTextFields==false && emptyTextFields==false) { //runs if no error
    booleans
1719                    //creating file's name
1720                    StringTokenizer st = new StringTokenizer(projectNameTextfield.getText(),"
    ");
1721                    String fileName = "";
1722                    while(st.hasMoreTokens()) fileName+=st.nextToken(); //builds the project
    name
1723
1724                    //data file
1725                    File dataFile = new File(fileName+"Data.txt"); //creates the project's
    data file
1726                    FileWriter dataWriter = null;
1727                        try {dataWriter = new FileWriter(dataFile);}
1728                        catch (IOException e1) {e1.printStackTrace();}
1729                    PrintWriter dpw = new PrintWriter(dataWriter); //makes a File and
    PrintWriter
1730                    dpw.println(projectNameTextfield.getText());
1731                    dpw.println(adressTextfield.getText());
1732                    dpw.println(cityTextfield.getText());
1733                    dpw.println(stateTextfield.getText());
```

```
1734                    dpw.println(totalLotsInDevTextfield.getText());
1735                    dpw.println(lblTotalDevelopedTextfield.getText());
1736                    dpw.println(totalLotsUnderConstructionTextfield.getText());
1737                    dpw.println(totalLotsRemainingTextfield.getText());
1738                    dpw.println(lotsSoldWithinTextfield.getText());
1739                    dpw.println(averageCostPerTextfield.getText());
1740                    dpw.println(averagePricePerTextfield.getText());
1741                    //^writes the project's data into the new data file
1742                    dpw.close();
1743
1744                    //account file
1745                    File accountFile = new File(fileName+"Accounts.txt"); //creates the
    project's account file
1746                    FileWriter accountWriter = null;
1747                        try {accountWriter = new FileWriter(accountFile);}
1748                        catch (IOException e1) {e1.printStackTrace();}
1749                    PrintWriter apw = new PrintWriter(accountWriter); //makes a File and
    PrintWriter
1750                    apw.println(account[accountIndexUsed].getName()); //automatically adds
    the client to the file
1751                    for(int x=0;x<chckbxAccount.length;x++)
1752                        if(chckbxAccount[x].isSelected())
1753                            apw.println(chckbxAccount[x].getText());
1754                    //^if checkbox is selected, then the account is added to the project
    account file
1755                    apw.close();
1756
1757                    //writes new project to the list of projects file
1758                    try{
1759                        RandomAccessFileEditor.addProject("listOfProjectsRAF.txt", fileName);
1760                    }
1761                    catch (Exception e2) {
1762                        e2.printStackTrace();
1763                    }
1764
1765                    //creates task, comment, and problem files
1766                    File tasksFile = new File(fileName+"Tasks.csv"); //creates the project's
    tasks file
1767                        FileWriter tasksFileWriter = null;
1768                            try {tasksFileWriter = new FileWriter(tasksFile, true);}
1769                            catch (IOException e1) {e1.printStackTrace();}
1770                        PrintWriter tasksPW = new PrintWriter(tasksFileWriter); //makes a
    File and PrintWriter
1771                        tasksPW.print("Task,Contractor,Projected Completion");
1772                        //^prints headers despite using a tableModel (used for exports;
    requested by the client)
1773                        tasksPW.close();
1774                    File commentsFile = new File(fileName+"Comments.txt"); //creates the
    project's comment file
1775                        FileWriter commentsFileWriter = null;
1776                            try {commentsFileWriter = new FileWriter(commentsFile, true);}
1777                            catch (IOException e1) {e1.printStackTrace();}
1778                        PrintWriter commentsPW = new PrintWriter(commentsFileWriter); //makes
    a File and PrintWriter
1779                        commentsPW.print(""); //ensures project comments file is ready for
    future comments
1780                        commentsPW.close();
```

```
1781                    File problemsFile = new File(fileName+"Problems.csv"); //creates the project's problems file
1782                        FileWriter problemsFileWriter = null;
1783                            try {problemsFileWriter = new FileWriter(problemsFile, true);}
1784                            catch (IOException e1) {e1.printStackTrace();}
1785                        PrintWriter problemsPW = new PrintWriter(problemsFileWriter); //makes a File and PrintWriter
1786                        problemsPW.print("Title,Description,Priority");
1787                        //^ensures project problems file is ready for future problems
1788                        problemsPW.close();
1789
1790                //GUI processes
1791                    mainMenuPane.setVisible(true);
1792                    newProjectPane.setVisible(false); //returns user to the main menu
1793                    clear.newProjectPage(); //clears the new project pane
1794                    projectListMaker.createProjectList();
1795                    //^reruns the project list algorithm so that the new project appears as a button
1796                }
1797
1798            }
1799        });
1800
1801        btnEdit.addActionListener(new ActionListener() {
1802            public void actionPerformed(ActionEvent arg0) {
1803                displayData.editProjectData();
1804
1805                checker.captureOriginalDataValues();
1806            }
1807        });
1808
1809        //adds the new task to the task table
1810        btnAdd.addActionListener(new ActionListener() {
1811            public void actionPerformed(ActionEvent arg0) {
1812
1813                String typeOfTask = tasksComboBox.getSelectedItem().toString();
1814                String contractor = contractorTextField.getText();
1815                String date = projCompMMTextField.getText() + "/" + projCompDDTextField.getText() + "/" +
1816                        projCompYYYYTextField.getText();
1817
1818                boolean emptyTextFields = false;
1819                boolean noNumberTextFields = false; //sets all error booleans to false
1820
1821                if(projCompYYYYTextField.getText().isEmpty() ||
1822                    projCompDDTextField.getText().isEmpty() ||
1823                    projCompMMTextField.getText().isEmpty() ||
1824                    contractorTextField.getText().isEmpty() ||
1825                    tasksComboBox.getSelectedItem().equals(" ")) {
1826                    //^runs if textfield(s) are left empty
1827                        emptyTextFields =  true;
1828                        System.out.println("ERROR: Empty text fields");
1829                }
1830
1831                if(emptyTextFields==true) {
1832                //^runs if textfield(s) are left empty
1833                    JOptionPane.showMessageDialog(new JFrame(),
```

```
1834                            "All textboxes must be filled before submission",
1835                            "ERROR",
1836                            JOptionPane.WARNING_MESSAGE);
1837                    //^A warning option pane notifies the user of the blank textfields
1838                }
1839
1840                if(emptyTextFields==false) {
1841                //^runs if all textfields are filled
1842                    try {
1843                        Integer.parseInt(projCompYYYYTextField.getText());
1844                        Integer.parseInt(projCompDDTextField.getText());
1845                        Integer.parseInt(projCompMMTextField.getText());
1846                        //^tests if certain textfields are numbers
1847                    }
1848                    catch(NumberFormatException e1) {
1849                    //^runs if those certian textfields are not numbers
1850                        JOptionPane.showMessageDialog(new JFrame(),
1851                                "Please insert a valid date",
1852                                "ERROR",
1853                                JOptionPane.WARNING_MESSAGE);
1854                        noNumberTextFields = true;
1855                        //^runs a warning option pane notifying the client of the mistake
1856                    }
1857                }
1858                if(noNumberTextFields==false && emptyTextFields==false) {
1859                //^runs if no errors have been found
1860                    taskTable.addData(); //adds the new task
1861                    clear.newTasks(); //clears the textfields on the tasks pane
1862
1863                    TaskEvent temp = new TaskEvent(account[accountIndexUsed].getName(),
    lblProjectTitle.getText(),
1864                            typeOfTask, contractor, date);
1865
1866                    EventScripter.addEvent(temp);
1867                }
1868            }
1869        });
1870
1871        //exports the task csv file by opening the file in notepad
1872        btnExport.addActionListener(new ActionListener() {
1873            public void actionPerformed(ActionEvent arg0) {
1874                StringTokenizer st = new StringTokenizer(lblProjectTitle.getText());
1875                String nameBuilder="";
1876                while(st.hasMoreTokens()) nameBuilder+=st.nextToken();
1877                    try {Runtime.getRuntime().exec("notepad "+nameBuilder+"Tasks.csv");}
1878                    catch (IOException e) {e.printStackTrace();}
1879            }
1880        });
1881
1882        //saves a new comment made
1883        btnSubmit_1.addActionListener(new ActionListener() {
1884            public void actionPerformed(ActionEvent arg0) {
1885
1886                String comment = commentTextArea.getText();
1887
1888                boolean emptyTextFields = false; //sets error boolean to false
1889
```

```
1890                if(commentTextArea.getText().isEmpty()) {
1891                        emptyTextFields =  true;
1892                        System.out.println("ERROR: Empty text fields");
1893                }
1894                if(emptyTextFields==true) { //runs if the comment area is left blank
1895                    JOptionPane.showMessageDialog(new JFrame(),
1896                            "You must write a comment before you submit one!",
1897                            "ERROR",
1898                            JOptionPane.WARNING_MESSAGE);
1899                    //^runs a warning option pane informing the client of the problem
1900                }
1901                if(emptyTextFields==false) { //runs if there are no errors
1902                    commentBoard.addComment();
1903
1904                    CommentEvent temp = new CommentEvent(account[accountIndexUsed].getName(),
    lblProjectTitle.getText(),comment);
1905
1906                    EventScripter.addEvent(temp);
1907                }
1908            }
1909        });
1910
1911        //adds a new problem
1912        btnAddProblem.addActionListener(new ActionListener() {
1913            public void actionPerformed(ActionEvent arg0) {
1914
1915                String title = titleTextField.getText();
1916                String description = descriptionTextField.getText();
1917                String priority = priorityComboBox.getSelectedItem().toString();
1918
1919                boolean emptyTextFields = false;
1920
1921                if(titleTextField.getText().isEmpty() ||
    descriptionTextField.getText().isEmpty()) {
1922                //^runs if textfield(s) are left empty
1923                    emptyTextFields =  true;
1924                    System.out.println("ERROR: Empty text fields");
1925                }
1926
1927                if(emptyTextFields==true) {
1928                //^runs if textfield(s) are left empty
1929                    JOptionPane.showMessageDialog(new JFrame(),
1930                            "All textboxes must be filled before submission",
1931                            "ERROR",
1932                            JOptionPane.WARNING_MESSAGE);
1933                    //^A warning option pane notifies the user of the blank textfields
1934                }
1935
1936                if(emptyTextFields==false) {
1937                //^runs if no errors have been found
1938                    problemTable.addData(); //adds the new task
1939                    clear.newProblems(); //clears the textfields on the tasks pane
1940
1941                    ProblemEvent temp = new ProblemEvent(account[accountIndexUsed].getName(),
    lblProjectTitle.getText(),
1942                                                        title, description, priority, 0);
1943
```

```
1944                    EventScripter.addEvent(temp);
1945                }
1946
1947            }
1948        });
1949
1950        //Creates a recommended problem to solve
1951        btnRecommendAProblem.addActionListener(new ActionListener() {
1952            public void actionPerformed(ActionEvent e) {
1953                //reads the problems csv file
1954                StringTokenizer st = new StringTokenizer(lblProjectTitle.getText()," ");
1955                String fileName="";
1956                while(st.hasMoreTokens()) fileName+=st.nextToken();
1957                File csvFile = new File(fileName+"Problems.csv");
1958                Scanner reader=null;
1959                    try {reader = new Scanner(csvFile);}
1960                    catch (FileNotFoundException e1) {e1.printStackTrace();}
1961
1962                //creates the data ArrayList
1963                ArrayList<String> dataList = new ArrayList<String>();
1964
1965                //reads through the header values (they have already been set and only...
1966                //...exist in the text file for exporting reasons)
1967                reader.nextLine();
1968
1969                //reads the actual task data
1970                while(reader.hasNextLine()) dataList.add(reader.nextLine());
1971
1972                //converts from an ArrayList to a normal array
1973                String[] dataArray = new String[dataList.size()];
1974                for(int x=0;x<dataArray.length;x++) dataArray[x] = dataList.get(x);
1975
1976                //creates an array of Problem instances
1977                PQueue tempProblemQueue = new PQueue(dataList.size()*3);
1978                //the capacity is three times larger than the list size for extra...
1979                //...robustness at the sake of computational speed
1980
1981                //adds the data to the Problem Queue
1982                for(int x=0;x<dataArray.length;x++){
1983                    String[] rowData = new String[3];
1984                    int rowIndex = 0;
1985                    StringTokenizer st2 = new StringTokenizer(dataArray[x], ",");
1986                    String title = st2.nextToken();
1987                    String description = st2.nextToken();
1988                    int priority = Integer.parseInt(st2.nextToken());
1989                    tempProblemQueue.enqueue(new Problem(title, description, priority));
1990                }
1991                reader.close();
1992
1993                //checks for empty problem queue
1994                if(tempProblemQueue.isEmpty()) {
1995                    JOptionPane.showMessageDialog(new JFrame(),
1996                            "There are no problems to solve",
1997                            "ERROR",
1998                            JOptionPane.WARNING_MESSAGE);
1999                }
2000                else {
```

```
2001                    //recognizes the recommended problem
2002                    Problem recommendation = tempProblemQueue.front();
2003                    String output = "";
2004                    output += "SOLVE THIS PROBLEM FIRST\n\n";
2005                    output += "Title: "+recommendation.getTitle()+"\n";
2006                    output += "Description: "+recommendation.getDescription()+"\n";
2007                    output += " ";
2008
2009                    Object[] selectionOptions={"Solve Problem","Dismiss"};
2010                    int resetWarning = JOptionPane.showOptionDialog //generates a JOptionPane
    warning the client
2011                                (null, output,
2012                                        "Recommended Problem to
    Solve",JOptionPane.DEFAULT_OPTION, JOptionPane.WARNING_MESSAGE,
2013                                        null, selectionOptions, selectionOptions[0]);
2014                    if(resetWarning==0) { //runs if "Solve Problem" was selected
2015                        Problem tempProblem = problemTable.removeProblem();
2016                        problemTable.generateData(); //removes the problem
2017
2018                        ProblemEvent temp = new
    ProblemEvent(account[accountIndexUsed].getName(), lblProjectTitle.getText(),
2019                                tempProblem.getTitle(), tempProblem.getDescription(),
    ""+tempProblem.getPriority(), 1);
2020                        //problem event is instantiated with a "1" parameter at the end to
    signal this is a ~removed~ problem
2021
2022                        EventScripter.addEvent(temp);
2023                    }
2024                }
2025            }
2026        });
2027
2028
2029        //allows the client to CTRL+TAB through project tabs
2030        ArrayList<Integer> pressedKeys = new ArrayList<Integer>();
2031        tabbedPane.addKeyListener(new KeyAdapter() {
2032            public void keyPressed(KeyEvent e) {
2033                pressedKeys.add(e.getKeyCode());
2034                if(pressedKeys.size()>1) {
2035                    int selectionIndex = tabbedPane.getSelectedIndex();
2036                    switch(selectionIndex) {
2037                        case 0: tabbedPane.setSelectedIndex(1);
2038                        case 1: tabbedPane.setSelectedIndex(2);
2039                        case 2: tabbedPane.setSelectedIndex(0);
2040                    }
2041                }
2042            }
2043            public void keyReleased(KeyEvent e) {
2044                pressedKeys.remove(e.getKeyCode());
2045            }
2046            public void keyTyped(KeyEvent e) {/*Not overrided*/}
2047        });
2048
2049        //Allows the client to view the event script
2050        btnEventLog.addActionListener(new ActionListener() {
2051            public void actionPerformed(ActionEvent arg0) {
2052
```

```
2053                Object[] options = {"Unsorted/Chronological",
2054                                    "Sorted by Project",
2055                                    "Sorted by Operation"};
2056                int n = JOptionPane.showOptionDialog(new JFrame(),
2057                    "Select your event log sorting type",
2058                    "Event Log",
2059                    JOptionPane.YES_NO_CANCEL_OPTION,
2060                    JOptionPane.QUESTION_MESSAGE,
2061                    null,
2062                    options,
2063                    options[0]);
2064
2065                JFrame eventFrame = new JFrame();
2066                    //eventFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
2067                    eventFrame.setTitle("Event Log");
2068                    eventFrame.setSize(500,500);
2069                    eventFrame.setLocationRelativeTo(null);
2070
2071                JPanel eventPanel = new JPanel();
2072                GridLayout layout = new GridLayout();
2073                eventPanel.setLayout(layout);
2074
2075                String[][] data = null;
2076                switch(n) {
2077                case 0:
2078                    data =
    EventScripter.getUnsortedEventTable(account[accountIndexUsed].getName());
2079                    break;
2080                case 1:
2081                    data =
    EventScripter.getProjectSortedEventTable(account[accountIndexUsed].getName());
2082                    break;
2083                case 2:
2084                    data =
    EventScripter.getOperationSortedEventTable(account[accountIndexUsed].getName());
2085                    break;
2086                }
2087                String[] columnName = {"Project","Operation","Details"};
2088                JTable eventTable = new JTable(data, columnName);
2089                JScrollPane scrollpane = new JScrollPane(eventTable);
2090                eventPanel.add(scrollpane);
2091
2092
2093                eventFrame.add(eventPanel);
2094                eventFrame.setVisible(true);
2095            }
2096        });
2097
2098
2099    }
2100 }
```

```java

public class Account {

    //variables
    private int index;
    private String username;
    private String password;
    private String name;

    //constructors
    public Account(int i, String u, String p, String n) {
        index = i;
        username = u;
        password = p;
        name = n;
    }
    public Account() {
        this(0, "", "", "");
    }
    public void setAccount(Account a) {
        index = a.getIndex();
        username = a.getUsername();
        password = a.getPassword();
        name = a.getName();
    }

    //setters
    public void setIndex(int i){
        index = i;
    }
    public void setUsername(String u) {
        username = u;
    }
    public void setPassword(String p) {
        password = p;
    }
    public void setName(String n) {
        name = n;
    }

    //getters
    public int getIndex(){
        return index;
    }
    public String getUsername() {
        return username;
    }
    public String getPassword() {
        return password;
    }
    public String getName() {
        return name;
    }
}

```

```java
import java.util.ArrayList;

public abstract class Event {

    protected static final String[] operationOptions =
 {"Undeclaired","Data","Task","Comment","Problem"};

    protected String client;
    protected String projectName;
    protected String operation;

    public Event() {
        client = "NA";
        projectName = "NA";
        operation = operationOptions[0];
    }

    public Event(String input, String input2) {
        client = input;
        projectName = input2;
        operation = operationOptions[0];
    }

    public Event(String input, String input2, int input3) {
        client = input;
        projectName = input2;
        operation = operationOptions[input3];
    }

    public static String[] getOperationOptions() {
        return operationOptions;
    }

    public String getClient() {
        return client;
    }

    public void setClient(String client) {
        this.client = client;
    }

    public String getProjectName() {
        return projectName;
    }

    public void setProjectName(String projectName) {
        this.projectName = projectName;
    }

    public String getOperation() {
        return operation;
    }

    public void setOperation(String operation) {
        this.operation = operation;
    }

```

```
57      public String toString() {
58          String output = client + " edited " + projectName + "'s " + operation + " page";
59          return output;
60      }
61
62      public String scriptLog() {
63          String output = client + "~" + projectName + "~" + "0" + "~";
64          return output;
65      }
66  }
67
```

CommentEvent.java

```
1
2 public class CommentEvent extends Event{
3
4     private String comment;
5
6     public CommentEvent() {
7         super();
8     }
9
10    public CommentEvent(String input1, String input2, String input3) {
11        super(input1, input2);
12        super.operation = super.operationOptions[3];
13        comment = input3;
14    }
15
16    public String getComment() {
17        return comment;
18    }
19
20    public void setComment(String comment) {
21        this.comment = comment;
22    }
23
24    public String toString() {
25        String output = client + " edited " + projectName + "'s " + operation + " page: \"" +
26                        comment + "\" was commented by " + client;
27        return output;
28    }
29
30    public String scriptLog() {
31        String output = client + "~" + projectName + "~" + "3" + "~" + comment + "~";
32        return output;
33    }
34 }
35
36
```

```java

public class DataEvent extends Event{

    private String dataFieldChanged;
    private String originalDataValue;
    private String newDataValue;

    public DataEvent() {
        super();
    }

    public DataEvent(String input1, String input2, String[] input4) {
        super(input1, input2);
        super.operation = super.operationOptions[1];

        assert input4.length >= 3;

        dataFieldChanged = input4[0];
        originalDataValue = input4[1];
        newDataValue = input4[2];
    }

    public DataEvent(String input1, String input2, String input3, String input4, String input5)
 {
        super(input1, input2);
        super.operation = super.operationOptions[1];

        dataFieldChanged = input3;
        originalDataValue = input4;
        newDataValue = input5;
    }

    public String getDataFieldChanged() {
        return dataFieldChanged;
    }

    public void setDataFieldChanged(String dataFieldChanged) {
        this.dataFieldChanged = dataFieldChanged;
    }

    public String getOriginalDataValue() {
        return originalDataValue;
    }

    public void setOriginalDataValue(String originalDataValue) {
        this.originalDataValue = originalDataValue;
    }

    public String getNewDataValue() {
        return newDataValue;
    }

    public void setNewDataValue(String newDataValue) {
        this.newDataValue = newDataValue;
    }

    public String toString() {
```

```
57          String output = client + " edited " + projectName + "'s " + operation + " page: " +
58                          dataFieldChanged + " was changed from " + originalDataValue + " to " +
  newDataValue;
59          return output;
60      }
61
62      public String scriptLog() {
63          String output = client + "~" + projectName + "~" + "1" + "~" + dataFieldChanged +
64                          "~" + originalDataValue + "~" + newDataValue + "~";
65          return output;
66      }
67  }
68
```

```java

public class TaskEvent extends Event{

    private String taskCategory;
    private String taskContractor;
    private String taskProjectedCompletion;

    public TaskEvent() {
        super();
    }

    public TaskEvent(String input1, String input2, String[] input3) {
        super(input1, input2);
        super.operation = super.operationOptions[2];

        assert input3.length >= 3;

        taskCategory = input3[0];
        taskContractor = input3[1];
        taskProjectedCompletion = input3[2];
    }

    public TaskEvent(String input1, String input2, String input3, String input4, String input5)
 {
        super(input1, input2);
        super.operation = super.operationOptions[2];

        taskCategory = input3;
        taskContractor = input4;
        taskProjectedCompletion = input5;
    }

    public String getTaskCategory() {
        return taskCategory;
    }

    public void setTaskCategory(String taskCategory) {
        this.taskCategory = taskCategory;
    }

    public String getTaskContractor() {
        return taskContractor;
    }

    public void setTaskContractor(String taskContractor) {
        this.taskContractor = taskContractor;
    }

    public String getTaskProjectedCompletion() {
        return taskProjectedCompletion;
    }

    public void setTaskProjectedCompletion(String taskProjectedCompletion) {
        this.taskProjectedCompletion = taskProjectedCompletion;
    }

    public String toString() {
```

```
        String output = client + " edited " + projectName + "'s " + operation + " page: A " +
                        taskCategory + " task was contracted with " + taskContractor + " set
  for " + taskProjectedCompletion;
        return output;
    }

    public String scriptLog() {
        String output = client + "~" + projectName + "~" + "2" + "~" + taskCategory +
                        "~" + taskContractor + "~" + taskProjectedCompletion + "~";
        return output;
    }

}


```

```java

public class ProblemEvent extends Event{

    private String problemTitle;
    private String problemDescription;
    private String problemPriority;
    private int problemAddedOrDeleted;
        //0=added (enqueued)
        //1=deleted (dequeued)

    public ProblemEvent() {
        super();
    }

    public ProblemEvent(String input1, String input2, String[] input3) {
        super(input1, input2);
        super.operation = super.operationOptions[4];

        assert input3.length >= 4;

        problemTitle = input3[0];
        problemDescription = input3[1];
        problemPriority = input3[2];
        int temp = Integer.parseInt(input3[3].trim());
        assert temp==0||temp==1;
        problemAddedOrDeleted = temp;
    }

    public ProblemEvent(String input1, String input2, String input3, String input4, String input5, int input6) {
        super(input1, input2);
        super.operation = super.operationOptions[4];

        problemTitle = input3;
        problemDescription = input4;
        problemPriority = input5;
        assert input6==0||input6==1;
        problemAddedOrDeleted = input6;
    }

    public String getProblemTitle() {
        return problemTitle;
    }

    public void setProblemTitle(String problemTitle) {
        this.problemTitle = problemTitle;
    }

    public String getProblemDescription() {
        return problemDescription;
    }

    public void setProblemDescription(String problemDescription) {
        this.problemDescription = problemDescription;
    }

    public String getProblemPriority() {
```

```
57          return problemPriority;
58      }
59
60      public void setProblemPriority(String problemPriority) {
61          this.problemPriority = problemPriority;
62      }
63
64      public int getProblemAddedOrDeleted() {
65          return problemAddedOrDeleted;
66      }
67
68      public void setProblemAddedOrDeleted(int problemAddedOrDeleted) {
69          assert problemAddedOrDeleted==0||problemAddedOrDeleted==1;
70          this.problemAddedOrDeleted = problemAddedOrDeleted;
71      }
72
73      public String toString() {
74          String output = client + " edited " + projectName + "'s " + operation + " page: ";
75          if(problemAddedOrDeleted==0) {
76              output += problemTitle + " (" + problemDescription + ") was added as a problem with
  priority " + problemPriority;
77          }
78          if(problemAddedOrDeleted==1) {
79              output += problemTitle + " (" + problemDescription + ") was removed as a problem
  with priority " + problemPriority;
80          }
81
82          return output;
83      }
84
85      public String scriptLog() {
86          String output = client + "~" + projectName + "~" + "4" + "~" + problemTitle +
87                          "~" + problemDescription + "~" + problemPriority +"~" +
  problemAddedOrDeleted + "~";
88          return output;
89      }
90
91  }
92
93
```

```java
import java.io.File;

public class EventScripter {
    private static Stack<Event> eventStack = new Stack<Event>();
    private static File eventLog = new File("eventLog.txt");
    private static FileWriter fileWriter;
    private static PrintWriter printWriter;
    private static Scanner scanner;

    public static void buildStackFromEventLogFile() {

        if(!eventStack.isEmpty()) eventStack.clear();

        try {
            scanner = new Scanner(eventLog);
        }
        catch (FileNotFoundException e) {
            e.printStackTrace();
        }
        while(scanner.hasNextLine()) {
            //reads next line
            String[] tempTokens = scanner.nextLine().split("~");

            //creates an array with guaranteed size >5
            String[] tokens = new String[7];
            for(int x=0;x<tempTokens.length;x++) tokens[x] = tempTokens[x];

            //adds according to the event type
            switch(Integer.parseInt(tokens[2].trim())) {
            case 1:
                DataEvent tempData = new DataEvent(tokens[0],tokens[1],tokens[3],
                        tokens[4],tokens[5]);
                eventStack.push(tempData);
                //System.out.println("New data event added to stack --- " + tempData);
                break;
            case 2:
                TaskEvent tempTask = new TaskEvent(tokens[0],tokens[1],tokens[3],
                        tokens[4],tokens[5]);
                eventStack.push(tempTask);
                //System.out.println("New task event added to stack --- " + tempTask);
                break;
            case 3:
                CommentEvent tempComment = new CommentEvent(tokens[0],tokens[1],tokens[3]);
                eventStack.push(tempComment);
                //System.out.println("New comment event added to stack --- " + tempComment);
                break;
            case 4:
                ProblemEvent tempProblem = new ProblemEvent(tokens[0],tokens[1],tokens[3],
                        tokens[4],tokens[5],Integer.parseInt(tokens[6].trim()));
                eventStack.push(tempProblem);
                //System.out.println("New problem event added to stack --- " + tempProblem);
                break;
            }
        }
        scanner.close();
    }

```

```
66      public static void buildLogFileFromStack() {
67
68          //clears the contents of the existing events log file
69          FileWriter fwOb = null;
70          PrintWriter pwOb = null;
71          try {
72              fwOb = new FileWriter(eventLog, false);
73              pwOb = new PrintWriter(fwOb, false);
74          }
75          catch (IOException e1) {
76              e1.printStackTrace();
77          }
78          pwOb.flush();
79          pwOb.close();
80          try {
81              fwOb.close();
82          }
83          catch (IOException e1) {
84              e1.printStackTrace();
85          }
86
87          //adds the events from the Stack
88          try {
89              //creates the file writer and print writer
90              fileWriter = new FileWriter(eventLog, true);
91              printWriter = new PrintWriter(fileWriter);
92          }
93          catch (IOException e) {
94              e.printStackTrace();
95          }
96
97          //flips the ordering so the following lines of code don't write to the text field
    backwards
98          Stack<Event> temp = new Stack<Event>();
99          while(!eventStack.isEmpty()) {
100             temp.push(eventStack.pop());
101         }
102
103         //adds the file's scriptLog to the event log
104         System.out.println("DEBUG TEMP: "+temp);
105         while(!temp.isEmpty()) {
106             printWriter.print(temp.pop().scriptLog());
107             printWriter.print("\r\n");
108         }
109         printWriter.close();
110
111         //rebuilds the stack becuase in the process of creating the next text file, the stack
    was emptied
112         buildStackFromEventLogFile();
113     }
114
115     public static String addEvent(Event newEvent) {
116         //adds to the Stack
117         eventStack.push(newEvent);
118
119         //adds to the event log file
120         try {
```

```
121             //creates the file writer and print writer
122             fileWriter = new FileWriter(eventLog, true);
123             printWriter = new PrintWriter(fileWriter);
124         }
125         catch (IOException e) {
126             e.printStackTrace();
127         }
128         //adds the file's scriptLog to the event log
129         printWriter.print(newEvent.scriptLog());
130         printWriter.print("\r\n");
131         System.out.println(newEvent.toString());
132 
133         printWriter.close();
134 
135         return newEvent.toString();
136     }
137 
138     public static Stack<Event> removeEvent(Event removeEvent) {
139         buildStackFromEventLogFile();
140         Stack<Event> tempStack = new Stack<Event>();
141         while(!eventStack.isEmpty()) {
142             Event tempEvent = eventStack.pop();
143             //note the script logs (and not the objects themselves) are compared because the
    vales should not be compared...
144             //...not the reference points of the actual reference variables
145             if(!tempEvent.scriptLog().equals(removeEvent.scriptLog())) {
146                 tempStack.push(tempEvent);
147             }
148         }
149         while(!tempStack.isEmpty()) {
150             eventStack.push(tempStack.pop());
151         }
152 
153         buildLogFileFromStack();
154 
155         return eventStack;
156     }
157 
158     public static Stack<Event> getEventStack() {
159         return eventStack;
160     }
161 
162     public static void setEventStack(Stack<Event> eventStack) {
163         EventScripter.eventStack = eventStack;
164     }
165 
166     public static String[][] getUnsortedEventTable(String name){
167 
168         buildStackFromEventLogFile();
169 
170         //constructs a properly sized 2D array
171         String data[][] = new String[eventStack.size()][3];
172 
173         //copies the existing Stack to a temp variable
174         Stack<Event> tempStack = (Stack<Event>) eventStack.clone();
175 
176         int c=0;
```

```
177         while(!tempStack.isEmpty()) {
178             Event tempEvent = tempStack.pop();
179             if(tempEvent.getClient().equals(name)) {
180 
181                 data[c][0] = tempEvent.getProjectName();
182 
183                 //removes the unnessary content from the toString that complicates...
184                 //...the already mentioned details by removing everything before the colon
185                 String[] tempTokens = tempEvent.toString().split(":");
186                 String details = tempTokens[1];
187 
188                 //distinguishes which operation was used and properly...
189                 //...writes the subsequent data
190                 if(tempEvent.getOperation().trim().equals("Data")) {
191                     data[c][1] = "Data";
192                     data[c][2] = details;
193                     c++;
194                 }
195                 if(tempEvent.getOperation().trim().equals("Task")) {
196                     data[c][1] = "Task";
197                     data[c][2] = details;
198                     c++;
199                 }
200                 if(tempEvent.getOperation().trim().equals("Comment")) {
201                     data[c][1] = "Comment";
202                     data[c][2] = details;
203                     c++;
204                 }
205                 if(tempEvent.getOperation().trim().equals("Problem")) {
206                     data[c][1] = "Problem";
207                     data[c][2] = details;
208                     c++;
209                 }
210             }
211         }
212 
213         //System prints for debugging
214         for(int x=0;x<data.length;x++) {
215             for(int y=0;y<data[x].length;y++) {
216                 System.out.print(data[x][y]);
217                 System.out.print("\t");
218             }
219             System.out.print("\n");
220         }
221 
222         return data;
223     }
224 
225     public static String[][] getProjectSortedEventTable(String name){
226 
227         buildStackFromEventLogFile();
228 
229         //creates a properly sized 2D array
230         String data[][] = new String[eventStack.size()][3];
231 
232         //copies the existing stack to a temp variable
233         Stack<Event> tempStack = (Stack<Event>) eventStack.clone();
```

```
234
235         //adds every element in the stack to the new ArrayList
236         ArrayList<Event> eventList = new ArrayList<Event>();
237         while(!tempStack.isEmpty()) {
238             Event tempEvent = tempStack.pop();
239             if(tempEvent.getClient().equals(name)) {
240                 eventList.add(tempEvent);
241             }
242         }
243
244         //Uses RandomAccessFileEditor to generate a list of projects
245         String[] projectArr = null;
246         try {projectArr = RandomAccessFileEditor.getProjects("listOfProjectsRAF.txt");}
247         catch (Exception e) {e.printStackTrace();}
248
249         /*Creates an array of ArrayLists to store events to. Each ArrayList contains
250         all the events from one project. Thus, the array contains all the ArrayLists
251         of all the different projects the client is a part of. The loop manually
252         instantiates each ArrayList to avoid null pointer errors*/
253         ArrayList<Event>[] sortedEventList = new ArrayList[projectArr.length];
254         for(int x=0;x<sortedEventList.length;x++) {
255             sortedEventList[x] = new ArrayList<Event>();
256         }
257
258         for(int x=0;x<eventList.size();x++) {
259             for(int y=0;y<projectArr.length;y++) {
260                 StringTokenizer st = new StringTokenizer(eventList.get(x).getProjectName());
261                 String projectName = "";
262                 while(st.hasMoreTokens()) projectName += st.nextToken();
263                 //^modifies the Project Name's title to contain no spaces
264                 if(projectName.equals(projectArr[y])) {
265                     sortedEventList[y].add(eventList.get(x));
266                 }
267                 //^if an event matches a project, it's added to its respective ArrayList
268             }
269         }
270
271         ArrayList<Event> finishedSortedEventList = new ArrayList<Event>();
272         //^The final ArrayList to contain all the merged events, now in proper order
273         for(int x=0;x<sortedEventList.length;x++) {
274             if(!sortedEventList[x].isEmpty()) {
275             //^checks to see if the project even has any events
276                 for(int y=0;y<sortedEventList[x].size();y++) {
277                     finishedSortedEventList.add(sortedEventList[x].get(y)); //adds event in
    order
278                 }
279             }
280         }
281
282         for(int z=0;z<finishedSortedEventList.size();z++) {
283             Event tempEvent = finishedSortedEventList.get(z);
284             data[z][0] = tempEvent.getProjectName();
285
286             String[] tempTokens = tempEvent.toString().split(":");
287             String details = tempTokens[1];
288
289             if(tempEvent.getOperation().trim().equals("Data")) {
```

```
290                 data[z][1] = "Data";
291                 data[z][2] = details;
292             }
293             if(tempEvent.getOperation().trim().equals("Task")) {
294                 data[z][1] = "Task";
295                 data[z][2] = details;
296             }
297             if(tempEvent.getOperation().trim().equals("Comment")) {
298                 data[z][1] = "Comment";
299                 data[z][2] = details;
300             }
301             if(tempEvent.getOperation().trim().equals("Problem")) {
302                 data[z][1] = "Problem";
303                 data[z][2] = details;
304             }
305         }
306 
307         //System prints for debugging
308         for(int x=0;x<data.length;x++) {
309             for(int y=0;y<data[x].length;y++) {
310                 System.out.print(data[x][y]);
311                 System.out.print("\t");
312             }
313             System.out.print("\n");
314         }
315 
316         return data;
317     }
318 
319     public static String[][] getOperationSortedEventTable(String name){
320 
321 
322         buildStackFromEventLogFile();
323 
324         String data[][] = new String[eventStack.size()][3];
325 
326         Stack<Event> tempStack = (Stack<Event>) eventStack.clone();
327 
328         ArrayList<Event> eventList = new ArrayList<Event>();
329         while(!tempStack.isEmpty()) {
330             Event tempEvent = tempStack.pop();
331             if(tempEvent.getClient().equals(name)) {
332                 eventList.add(tempEvent);
333             }
334         }
335 
336         //lists the operations in a String array
337         String[] operationArr = {"Undeclaired","Data","Task","Comment","Problem"};;
338 
339         /*Creates an array of ArrayLists to store events to. Each ArrayList contains
340         all the events from one type of operation. Thus, the array contains all the
341         ArrayLists of all the different operations the client is a part of. The loop
342         manually instantiates each ArrayList to avoid null pointer errors*/
343         ArrayList<Event>[] sortedEventList = new ArrayList[operationArr.length];
344         for(int x=0;x<sortedEventList.length;x++) {
345             sortedEventList[x] = new ArrayList<Event>();
346         }
```

```
347
348         for(int x=0;x<eventList.size();x++) {
349             for(int y=0;y<operationArr.length;y++) {
350                 if(eventList.get(x).getOperation().equals(operationArr[y])) {
351                     sortedEventList[y].add(eventList.get(x));
352                 }
353                 //^if an event matches an operation, it's added to its respective ArrayList
354             }
355         }
356
357         ArrayList<Event> finishedSortedEventList = new ArrayList<Event>();
358         //^The final ArrayList to contain all the merged events, now in proper order
359         for(int x=0;x<sortedEventList.length;x++) {
360             if(!sortedEventList[x].isEmpty()) {
361                 //^checks to see if the project even has any events
362                 for(int y=0;y<sortedEventList[x].size();y++) {
363                     finishedSortedEventList.add(sortedEventList[x].get(y)); //adds event in
    order
364                 }
365             }
366         }
367
368         for(int z=0;z<finishedSortedEventList.size();z++) {
369             Event tempEvent = finishedSortedEventList.get(z);
370             data[z][0] = tempEvent.getProjectName();
371
372             String[] tempTokens = tempEvent.toString().split(":");
373             String details = tempTokens[1];
374
375             if(tempEvent.getOperation().trim().equals("Data")) {
376                 data[z][1] = "Data";
377                 data[z][2] = details;
378             }
379             if(tempEvent.getOperation().trim().equals("Task")) {
380                 data[z][1] = "Task";
381                 data[z][2] = details;
382             }
383             if(tempEvent.getOperation().trim().equals("Comment")) {
384                 data[z][1] = "Comment";
385                 data[z][2] = details;
386             }
387             if(tempEvent.getOperation().trim().equals("Problem")) {
388                 data[z][1] = "Problem";
389                 data[z][2] = details;
390             }
391         }
392
393         //System prints for debugging
394         for(int x=0;x<data.length;x++) {
395             for(int y=0;y<data[x].length;y++) {
396                 System.out.print(data[x][y]);
397                 System.out.print("\t");
398             }
399             System.out.print("\n");
400         }
401
402         return data;
```

```
403     }
404
405 }
406
```

DataTable.java

```java
public interface DataTable {
    public void generateData();
    public void addData();
}
```

```java
import java.security.MessageDigest;

public class PasswordHasher {

    public static String generateHash(String input) {

        //Creates string builder to add chars to
        StringBuilder newHash = new StringBuilder();

        try {
            //Creates instance of the Secure Hash Algorithm 1
            MessageDigest sha1 = MessageDigest.getInstance("SHA-1");

            //Computes bytes from the input string using the SHA-1 hash function
            byte[] hashedBytes = sha1.digest(input.getBytes());

            //Translates hashed bytes into append'able chars
            char[] possibleChars =
  {'0','1','2','3','4','5','6','7','8','9','a','b','c','d','e','f'};
            for (int x=0;x<hashedBytes.length;++x) {
                byte b = hashedBytes[x];
                newHash.append(possibleChars[(b & 0xf0) >> 4]);
                newHash.append(possibleChars[b & 0x0f]);
            }
        }
        catch (NoSuchAlgorithmException e) {
            //catches possible exception
            System.out.println("ERROR IN GENERATING HASH FOR STRING: \" "+input+" \" ");
            e.printStackTrace();
        }

        return newHash.toString();
    }
}
```

```java
public class Problem implements Comparable<Problem>{
    private int priority;
    private String title;
    private String description;

    public Problem(){
        priority = 1;
        title = "NA";
        description = "";
    }

    public Problem(String s, String s2, int n){
        priority = n;
        title = s;
        description = s2;
    }

    public Problem(String s, int n){
        priority = n;
        title = s;
        description = "";
    }

    public String toString(){
        return "TITLE: " + title + " ... PRIORITY: " + priority;
    }

    public void setPriority(int n){
        priority = n;}

    public int getPriority(){
        return priority;}

    public void setTitle(String n){
        title = n;}

    public String getTitle(){
        return title;}

    public void setDescription(String n){
        description = n;}

    public String getDescription(){
        return description;}


    public int compareTo(Problem anotherProblem){
        return this.getPriority() - anotherProblem.getPriority();
    }
}
```

```java
public class PQueue{

    private Problem[] heap;
    private int heapSize;
    private int capacity;

    public PQueue(int capacity){
        this.capacity = capacity + 1;
        heap = new Problem[this.capacity];
        heapSize = 0;
    }

    public void clear(){
        heap = new Problem[capacity];
        heapSize = 0;
    }

    public boolean isEmpty(){
        return heapSize == 0;
    }

    public boolean isFull(){
        return heapSize == capacity - 1;
    }

    public int size(){
        return heapSize;
    }

    public void enqueue(Problem passedProblem){
        heap[++heapSize] = passedProblem;
        int pos = heapSize;
        while(pos != 1 && passedProblem.getPriority() > heap[pos/2].getPriority()){
            heap[pos] = heap[pos/2];
            pos /=2;
        }
        heap[pos] = passedProblem;
    }

    public Problem dequeue(){
        int parent, child;
        Problem item, temp;
        if (isEmpty()){
            System.out.println("Problem heap is empty");
            return null;
        }

        item = heap[1];
        temp = heap[heapSize--];

        parent = 1;
        child = 2;
        while (child <= heapSize){
            if(child < heapSize && heap[child].getPriority() < heap[child + 1].getPriority()) {
                child++;
            }
            if(temp.getPriority() >= heap[child].getPriority()) {
```

```
58                  break;
59              }
60
61              heap[parent] = heap[child];
62              parent = child;
63              child *= 2;
64          }
65          heap[parent] = temp;
66
67          return item;
68      }
69
70      public Problem front(){
71          return heap[1];
72      }
73
74      public Problem rear(){
75          return heap[heapSize];
76      }
77
78      public String toString(){
79          String output = null;
80          for(int x=0;x<heap.length;x++) {
81              output+=heap[x].toString()+"\n";
82          }
83          return output;
84      }
85  }
```

```java
import java.io.FileNotFoundException;

public  class RandomAccessFileEditor {

    private String nonStaticFileDirectory;

    public RandomAccessFileEditor(String nonStaticFileDirectory) {
        this.nonStaticFileDirectory = nonStaticFileDirectory;
    }

    public static String readData(String filepath, int position) throws Exception{
        //makes RAF and seeks to correct position
        RandomAccessFile file = new RandomAccessFile(filepath, "rw");
        file.seek(position);

        //collects the first character to see if its a " " space for later testing
        int readInt = file.read();
        String output = "";

        //RECURSIVELY collects the other characters
        file.seek(position);
        if(readInt!=32) output += (char)file.read() + readData(filepath, position+1);

        return output;
    }

    public static String[] getProjects(String filepath) throws Exception {
        //makes the array list to add project names to
        ArrayList<String> projectsList = new ArrayList<String>();

        //finds total seek size
        int totalSeek = Integer.parseInt(readData(filepath, 0));

        //hunts through the document to add projects to the array
        for(int x=100;x<=totalSeek;x+=100) {
            projectsList.add(readData(filepath, x));
        }

        for(int x=0;x<projectsList.size();x++) System.out.println(projectsList.get(x));

        //converts array list to traditional array; returns array
        String[] projectArray = new String[projectsList.size()];
        for(int x=0;x<projectsList.size();x++) projectArray[x] = projectsList.get(x);

        return projectArray;
    }

    public static void addProject(String filepath, String projectTitle) throws Exception {

        RandomAccessFile file = new RandomAccessFile(filepath, "rw");

        //read the size of the RAF
        int totalSeek = Integer.parseInt(readData(filepath, 0));
        int newTotal = totalSeek+100;
        String newTotalString = null;
        if(totalSeek>=000 && totalSeek<1000) newTotalString = "00"+newTotal;
        else if(totalSeek>=1000 && totalSeek<10000) newTotalString = "0"+newTotal;
```

```
61          else if(totalSeek>=10000 && totalSeek<100000) newTotalString = ""+newTotal;
62
63          //go back to the beginning and replace the new total size
64          file.seek(0);
65          file.write(newTotalString.getBytes("UTF-8"));
66
67          //go to the end of the file and add the new project
68          file.seek(newTotal);
69          String projectTitleToAdd = projectTitle+" ";
70          file.write(projectTitleToAdd.getBytes("UTF-8"));
71
72          file.close();
73
74          System.out.println(projectTitle + " added to file");
75      }
76
77      public static String readCharacter(String filepath, int position) {
78
79          RandomAccessFile file = null;
80
81          try
82              {file = new RandomAccessFile(filepath, "rw");}
83          catch (FileNotFoundException e2)
84              {e2.printStackTrace();}
85
86          try
87              {file.seek(position);}
88          catch (IOException e1)
89              {e1.printStackTrace();}
90
91          try
92              {return ""+(char)file.read();}
93          catch (IOException e)
94              {e.printStackTrace();}
95
96          return "error";
97      }
98
99      public static String readASCII(String filepath, int position) {
100
101         RandomAccessFile file = null;
102
103         try
104             {file = new RandomAccessFile(filepath, "rw");}
105         catch (FileNotFoundException e2)
106             {e2.printStackTrace();}
107
108         try
109             {file.seek(position);}
110         catch (IOException e1)
111             {e1.printStackTrace();}
112
113         try
114             {return ""+file.read();}
115         catch (IOException e)
116             {e.printStackTrace();}
117
```

```
118         return "error";
119     }
120 
121 
122     public String readData(int position) throws Exception {
123         return readData(nonStaticFileDirectory, position);
124     }
125 
126     public String[] getProjects() throws Exception {
127         return getProjects(nonStaticFileDirectory);
128     }
129 
130     public void addProject(String projectTitle) throws Exception {
131         addProject(nonStaticFileDirectory, projectTitle);
132     }
133 
134     public String readCharacter(int position) {
135         return readCharacter(nonStaticFileDirectory, position);
136     }
137 
138     public String readASCII(int position) {
139         return readCharacter(nonStaticFileDirectory, position);
140     }
141 }
142 
```