

```

1 /**
2  * Created by Daniel on 3/6/2016.
3 */
4 package trunk.View;
5
6 import javafx.application.Application;
7 import javafx.event.ActionEvent;
8 import javafx.event.EventHandler;
9 import javafx.geometry.Insets;
10 import javafx.geometry.Pos;
11 import javafx.scene.Scene;
12 import javafx.scene.control.*;
13 import javafx.scene.control.Alert.AlertType;
14 import javafx.scene.layout.*;
15 import javafx.scene.paint.Color;
16 import javafx.scene.text.Font;
17 import javafx.scene.text.FontWeight;
18 import javafx.scene.text.Text;
19 import javafx.stage.FileChooser;
20 import javafx.stage.Stage;
21 import javafx.util.converter.DateTimeStringConverter;
22 import trunk.Control.UpdateEQ;
23 import trunk.Model.Portfolio;
24 import trunk.Model.UndoAction;
25
26 import javax.crypto.Cipher;
27 import javax.crypto.KeyGenerator;
28 import javax.crypto.SecretKey;
29 import java.io.*;
30 import java.util.ArrayList;
31 import java.util.Optional;
32
33 //import trunk.Model.UndoAction;
34
35 //import svn/trunk.Model.Portfolio;
36
37 public class FPTS extends Application {
38     private static ArrayList<Portfolio> portfolios = new
39         ArrayList<Portfolio>();
40     public Stage stage;
41     int pindex = 0;
42     public static String id;
43
44     public static void main(String[] args) {
45         //Deleting portfolio by command line

```

```

45         if(args.length!= 0) {
46             int i = 0;
47             while(i < args.length) {
48                 if(args[i].equals("-delete")) {
49                     ArrayList<Portfolio> port = null;
50                     try {
51                         FileInputStream fileIn = new
52                             FileInputStream("myfile");
53                             ObjectInputStream in = new
54                             ObjectInputStream(fileIn);
55                             port = (ArrayList<Portfolio>) in.
56                             readObject();
57                             in.close();
58                             fileIn.close();
59                     } catch (ClassNotFoundException e) {
60                         e.printStackTrace();
61                     } catch (FileNotFoundException e) {
62                         e.printStackTrace();
63                     } catch (IOException e) {
64                         e.printStackTrace();
65                     }
66                     for (Portfolio p : port) {
67                         if (p.loginID.equals(args[i+1])) {
68                             port.remove(p);
69                             try {
70                                 FileOutputStream fos = new
71                                     FileOutputStream("myfile");
72                                     ObjectOutputStream oos =
73                                     new ObjectOutputStream(fos);
74                                     oos.writeObject(port);
75                                     oos.close();
76                                     fos.close();
77                             } catch (IOException ioe) {
78                                 ioe.printStackTrace();
79                             }
80                         }
81                     launch(args);
82                 }
83             }
84         /*
```

```

85      * Check if string is a double
86      */
87      public static boolean isDouble(String numString) {
88          try {
89              Double.parseDouble(numString);
90          } catch (Exception ex) {
91              return false;
92          }
93          return true;
94      }
95
96      /*
97      * Check if string is an int
98      */
99      public static boolean isInt(String numString) {
100         try {
101             Integer.parseInt(numString);
102         } catch (Exception ex) {
103             return false;
104         }
105         return true;
106     }
107
108     /*
109     * Check if string is a date
110     */
111     public static boolean isDate(String date) {
112         DateTimeStringConverter format = new
113             DateTimeStringConverter("MM/dd/YYYY");
114         try{
115             format.fromString(date);
116         } catch (Exception ex){
117             return false;
118         }
119         return true;
120     }
121     /*
122     * Code for adding a portfolio
123     */
124     public void addPortfolio(String username, String
password) {
125         Portfolio portfolio = new Portfolio("", "");
126         portfolio.loginID = username;
127         portfolio.password = password;

```

```

128         ArrayList<Portfolio> port2 = null;
129         try {
130             FileInputStream fileIn = new FileInputStream(
131                 "myfile");
132             ObjectInputStream in = new ObjectInputStream(
133                 fileIn);
134             port2 = (ArrayList<Portfolio>)in.readObject()
135             ;
136             in.close();
137             fileIn.close();
138         } catch (ClassNotFoundException ee) {
139             ee.printStackTrace();
140         } catch (FileNotFoundException ee) {
141             ee.printStackTrace();
142         } catch (IOException ee) {
143             ee.printStackTrace();
144         }
145         portfolios.add(pindex, portfolio);
146         pindex++;
147         try {
148             FileOutputStream fos = new FileOutputStream(""
149                 "myfile");
150             ObjectOutputStream oos = new
151                 ObjectOutputStream(fos);
152             oos.writeObject(portfolios);
153             oos.close();
154             fos.close();
155         } catch (IOException ioe) {
156             ioe.printStackTrace();
157         }
158         /*
159          *  Code for basic skeleton of border pane
160          */
161         public BorderPane borderPane(Stage primaryStage,
162             Portfolio p, Text t){
163             EquitiesView equitiesView = new EquitiesView();
164             CAView caView = new CAView();
165             simulationView simulationView = new
166                 simulationView();
167             transactionView transactionView = new

```

```

165 transactionView();
166     WatchlistView watchlistView = new WatchlistView()
167 ;
168     BorderPane border = new BorderPane();
169     primaryStage.setTitle(p.loginID + "'s Portfolio")
170 ;
171     GridPane grid = new GridPane();
172     grid.setAlignment(Pos.CENTER);
173     grid.setHgap(10);
174     grid.setVgap(10);
175     grid.setPadding(new Insets(5, 10, 5, 10));
176
177     Text title = t;
178     title.setFont(Font.font("Calibri", FontWeight.NORMAL, 20));
179     grid.add(title, 0, 0, 2, 1);
180
181     Button watchList = new Button("Watchlist");
182     Button equities = new Button("Equities");
183     Button history = new Button("Transaction History");
184
185     Button cas = new Button("Cash Accounts");
186     Button logout = new Button("Logout");
187     Button sim = new Button("Simulation");
188     Button importP = new Button("Import Portfolio");
189     Button exportP = new Button("Export Portfolio");
190     Button undo = new Button("Undo");
191     Button redo = new Button("Redo");
192
193     Button update = new Button("Update");
194
195     undo.setOnAction(new EventHandler<ActionEvent>()
196     { // Sadaf
197         @Override
198         public void handle(ActionEvent event) {
199             //code here
200
201             UndoAction test = new UndoAction(p);
202             test.handleCommand(primaryStage, FPTS.this,
203             ,equitiesView, caView);
204
205         }
206     });

```

```

204
205      redo.setOnAction(new EventHandler<ActionEvent>()
206      {
207          @Override
208          public void handle(ActionEvent event) {
209              UndoAction test = new UndoAction(p);
210              test.redo(primaryStage,FPTS.this,
211              equitiesView,caView);
212          }
213      });
214
215      logout.setOnAction(new EventHandler<ActionEvent>(
216      ) {
217          @Override
218          public void handle(ActionEvent e) {
219              UndoAction test = new UndoAction(p);
220
221              Alert alert = new Alert(AlertType.
222              CONFIRMATION);
223              alert.setTitle("Confirmation Dialog");
224              alert.setHeaderText("Do you want to save
225              changes to your portfolio?");
226
227              ButtonType save = new ButtonType("Save
228              changes");
229              ButtonType nosave = new ButtonType("Quit
230              without saving");
231              ButtonType cancel = new ButtonType("Cancel");
232
233              alert.getButtonTypes().addAll(save,
234              nosave, cancel);
235              Optional<ButtonType> result = alert.
236              showAndWait();
237
238              if (result.get() == save) {
239                  test.allclear();
240                  caView.saveCA(p);
241                  equitiesView.saveEquities(p);
242                  transactionView.saveHistory(p);
243                  watchlistView.saveWatchlist(p);
244                  p.saveMAs(p);
245
246                  Scene scene = login(primaryStage);
247                  stage.setScene(scene);
248              }

```

```

239             else if(result.get() == nosave) {
240                 test.allclear();
241                 Scene scene = login(primaryStage);
242                 stage.setScene(scene);
243             }
244             else{
245                 return;
246             }
247         }
248     );
249
250     importP.setOnAction(new EventHandler<ActionEvent>
251     () {
252         @Override
253         public void handle(ActionEvent event) {
254             FileChooser fileChooser = new FileChooser
255             ();
256             fileChooser.setTitle("Open Resource File"
257             );
258             File file = fileChooser.showOpenDialog(
259             stage);
260             if(file != null) {
261                 p.importP(file.toString(),p);
262             }
263         }
264     );
265
266     exportP.setOnAction(new EventHandler<ActionEvent>
267     () {
268         @Override
269         public void handle(ActionEvent event) {
270             final Stage dialog = new Stage();
271             dialog.initOwner(primaryStage);
272             GridPane grid2 = new GridPane();
273             grid2.setAlignment(Pos.CENTER);
274             grid2.setHgap(10);
275             grid2.setVgap(10);
276             grid2.setPadding(new Insets(5, 10, 5, 10)
277         );
278
279         TextField file = new TextField();
280         grid2.add(new Label("Filename: "),0,1);
281         grid2.add(file,1,1);
282
283         Scene dialogScene = new Scene(grid2, 400,

```

```
277    400);  
278            dialog.setScene(dialogScene);  
279            dialog.show();  
280            Button cancel = new Button("Cancel");  
281            grid2.add(cancel,1,7);  
282            Button submit = new Button("Submit");  
283            grid2.add(submit,0,7);  
284            cancel.setOnAction(new EventHandler<  
ActionEvent>() {  
285                @Override  
286                public void handle(ActionEvent e) {  
287                    dialog.close();  
288                }  
289            });  
290            submit.setOnAction(new EventHandler<  
ActionEvent>() {  
291                @Override  
292                public void handle(ActionEvent event)  
293                {  
294                    Portfolio pInfo = new Portfolio(  
295                    "", "");  
296                    pInfo.exportP(file.getText(), p);  
297                    dialog.close();  
298                }  
299            });  
300  
301            VBox topContainer = new VBox(2); //Creates a  
container to hold all Menu Objects.  
302            HBox buttonBar = new HBox(5); //Creates our tool  
-bar to hold the buttons.  
303  
304            Label space = new Label(" ");  
305            Label space2 = new Label(" ");  
306            Label spacer = new Label(" ");  
307            HBox.setHgrow(spacer, Priority.ALWAYS);  
308            spacer.setMaxWidth(Double.MAX_VALUE);  
309            Label spacer2 = new Label(" ");  
310            HBox.setHgrow(spacer2, Priority.ALWAYS);  
311            spacer2.setMaxWidth(Double.MAX_VALUE);  
312  
313            topContainer.getChildren().add(buttonBar);  
314            border.setTop(buttonBar);  
315
```

```

316         watchList.setOnAction(new EventHandler<
317             ActionEvent>() {
318                 @Override
319                 public void handle(ActionEvent event) {
320                     Scene scene = watchlistView.ViewWatchlist(
321                         primaryStage,p,FPTS.this);
322                     stage.setScene(scene);
323                 }
324             });
325             equities.setOnAction(new EventHandler<ActionEvent>()
326             () {
327                 @Override
328                 public void handle(ActionEvent event) {
329                     Scene scene = equitiesView.ViewEquities(
330                         primaryStage,p,FPTS.this);
331                     stage.setScene(scene);
332                 }
333             });
334             history.setOnAction(new EventHandler<ActionEvent>()
335             () {
336                 @Override
337                 public void handle(ActionEvent event) {
338                     Scene scene = transactionView.
339                         Transactions(primaryStage,p,FPTS.this);
340                     stage.setScene(scene);
341                 }
342             });
343             cas.setOnAction(new EventHandler<ActionEvent>() {
344                 @Override
345                 public void handle(ActionEvent event) {
346                     Scene scene = caView.CashAccounts(
347                         primaryStage,p,FPTS.this);
348                     stage.setScene(scene);
349                 }
350             });
351             sim.setOnAction(new EventHandler<ActionEvent>() {
352                 @Override
353                 public void handle(ActionEvent event) {
354                     Scene scene = simulationView.simulation(
355                         primaryStage, p, FPTS.this);
356                     stage.setScene(scene);
357                 }
358             });
359             update.setOnAction(new EventHandler<ActionEvent>(
360             ) {

```

```

352             @Override
353             public void handle(ActionEvent event) {
354                 try{
355                     UpdateEQ updateEQ = new UpdateEQ();
356                     updateEQ.updateEquities(p, p.
357                         equityList2);
358                     }catch(Exception ex){
359                         ex.getMessage();
360                     }
361                 });
362
363                 buttonBar.getChildren().addAll(space, importP,
364                     exportP, spacer2, watchList, equities, history, cas,
365                     sim, update, spacer, undo, redo, logout,
366                     space2);
367
368                 border.setCenter(grid);
369                 return border;
370             }
371             @Override
372             public void start(Stage primaryStage) {
373                 stage = primaryStage;
374
375                 Scene login = login(primaryStage);
376
377                 primaryStage.setScene(login);
378                 primaryStage.show();
379             }
380
381             /*
382             * GUI for login page
383             */
384             public Scene login(final Stage primaryStage) {
385                 primaryStage.setTitle("FPTS");
386                 GridPane grid = new GridPane();
387                 grid.setAlignment(Pos.CENTER);
388                 grid.setHgap(10);
389                 grid.setVgap(10);
390                 grid.setPadding(new Insets(5, 10, 5, 10));
391
392                 Text title = new Text("Financial Portfolio
393                 Tracking System");
394                 title.setFont(Font.font("Calibri", FontWeight.

```

```

392 NORMAL, 20));
393         grid.add(title,0,0,2,1);
394
395     Label user = new Label("Username:");
396     grid.add(user,0,1);
397     final TextField username = new TextField();
398     grid.add(username,1,1);
399
400     Label pass = new Label("Password:");
401     grid.add(pass,0,2);
402     PasswordField password = new PasswordField();
403     grid.add(password,1,2);
404
405     Button login = new Button("Login");
406     grid.add(login,0,4);
407
408     Button register = new Button("Register");
409     grid.add(register,1,4);
410
411     ArrayList<Portfolio> port = null;
412     try {
413         FileInputStream fileIn = new FileInputStream(
414             "myfile");
415         ObjectInputStream in = new ObjectInputStream(
416             fileIn);
417         port = (ArrayList<Portfolio>) in.readObject()
418         ;
419         in.close();
420         fileIn.close();
421     } catch (ClassNotFoundException e) {
422         e.printStackTrace();
423     } catch (FileNotFoundException e) {
424         e.printStackTrace();
425     } catch (IOException e) {
426         e.printStackTrace();
427     }
428     final ArrayList<Portfolio> ports = port;
429
430     final Text error = new Text();
431     grid.add(error, 1, 6);
432     login.setOnAction(new EventHandler<ActionEvent>()
{
433         @Override
434         public void handle(ActionEvent e) {
435             if(username.getText().trim().isEmpty() ||
```

```

432     password.getText().trim().isEmpty()) {
433             error.setFill(Color.FIREBRICK);
434             error.setText("Fill in all fields");
435         }
436     else {
437         if(ports != null) {
438             for (Portfolio p : ports) {
439                 if (p.loginID.equals(username
440 .getText().trim()) && p.password.equals(password.getText(
441 .trim()))) {
442                     Scene scene = portfolio(
443                         primaryStage, p);
444                     stage.setScene(scene);
445                 }
446             }
447         }
448     }
449 }
450
451     register.setOnAction(new EventHandler<ActionEvent> {
452         @Override
453         public void handle(ActionEvent event) {
454             Scene scene = register(primaryStage);
455             stage.setScene(scene);
456         }
457     });
458
459     Scene scene = new Scene(grid, 1200, 700);
460     return scene;
461 }
462
463
464 //GUI for user registration page
465 public Scene register(final Stage primaryStage) {
466     Portfolio portfolio = new Portfolio("", "");
467
468     primaryStage.setTitle("Registration");
469     GridPane grid = new GridPane();
470     grid.setAlignment(Pos.CENTER);
471     grid.setHgap(10);

```

```

472         grid.setVgap(10);
473         grid.setPadding(new Insets(5, 10, 5, 10));
474
475         Text title = new Text("Enter your Information");
476         title.setFont(Font.font("Calibri", FontWeight.
477             NORMAL, 20));
478         grid.add(title, 0, 0, 2, 1);
479
480         Label user = new Label("Username:");
481         grid.add(user, 0, 1);
482         final TextField username = new TextField();
483         grid.add(username, 1, 1);
484
485         Label pass = new Label("Password:");
486         grid.add(pass, 0, 2);
487         PasswordField password = new PasswordField();
488         grid.add(password, 1, 2);
489
490         final ToggleGroup group = new ToggleGroup();
491
492         RadioButton rb1 = new RadioButton("Yes");
493         rb1.setToggleGroup(group);
494
495         RadioButton rb2 = new RadioButton("No");
496         rb2.setToggleGroup(group);
497
498         Label imp = new Label("Import Portfolio?");
499         grid.add(imp, 0, 3);
500         grid.add(rb1, 0, 4);
501         grid.add(rb2, 1, 4);
502
503         final Text error = new Text();
504         grid.add(error, 1, 6);
505         Button register = new Button("Register");
506         grid.add(register, 0, 5);
507         register.setOnAction(new EventHandler<ActionEvent>
508             () {
509
510             @Override
511             public void handle(ActionEvent e) {
512                 final String name = username.getText().
513                     trim();
514                 if(username.getText().trim().isEmpty() ||
515                     password.getText().trim().isEmpty())
516                     error.setFill(Color.FIREBRICK);
517             }
518         });

```

```

513                     error.setText("Fill in all fields");
514
515                 }
516             else {
517                 boolean usertaken = false;
518                 ArrayList<Portfolio> port = null;
519                 try {
520                     FileInputStream fileIn = new
521                         FileInputStream("myfile");
522                         ObjectInputStream in = new
523                             ObjectInputStream(fileIn);
524                             port = (ArrayList<Portfolio>) in.
525                                 readObject();
526                                 in.close();
527                                 fileIn.close();
528             } catch (ClassNotFoundException ee) {
529                 ee.printStackTrace();
530             } catch (FileNotFoundException ee) {
531                 ee.printStackTrace();
532             } catch (IOException ee) {
533                 ee.printStackTrace();
534             }
535             if(port != null) {
536                 for (Portfolio p : port) {
537                     if (p.loginID.equals(username
538                         .getText().trim())) {
539                         error.setFill(Color.
540                             FIREBRICK);
541                         error.setText("Username
542                             is already taken");
543                         usertaken = true;
544                     }
545                 }
546             try{
547                 File file = new File("
548                     encryptedAccounts");
549                     FileWriter fwriter = new
550                         FileWriter(file.getAbsoluteFile());

```

```

549                                BufferedWriter bwriter = new
550                                BufferedWriter(fwriter);
551
552                                KeyGenerator keygenerator =
553                                KeyGenerator.getInstance("DES");
554                                SecretKey myDesKey =
555                                keygenerator.generateKey();
556
557                                Cipher desCipher;
558                                desCipher = Cipher.
559                                getInstance("DES");
560
561                                for (Portfolio p : port) {
562                                byte[] user = p.loginID.
563                                getBytes("UTF8");
564                                byte[] pass = p.password.
565                                getBytes("UTF8");
566
567                                desCipher.init(Cipher.
568                                ENCRYPT_MODE, myDesKey);
569
570                                desCipher.doFinal(user);
571
572                                desCipher.doFinal(pass);
573
574                                String username = new
575                                String(userEncrypted);
576                                String password = new
577                                String(passEncrypted);
578
579                                bwriter.write(username);
580                                bwriter.write(password);
581
582                                }
583                                bwriter.close();
584
585                                }catch(Exception ex)
586                                {
587                                System.out.println("Exception
588                                ");
589                                }
590
591                                if(rb1.isSelected()){
592                                ArrayList<Portfolio> port2 =
593                                null;
594
595                                try {
596                                FileInputStream fileIn =
597                                new FileInputStream("myfile");
598
599                                ObjectInputStream in =

```

```

579 new ObjectInputStream(fileIn);
580                                         port2 = (ArrayList<
581                                         Portfolio>) in.readObject();
582                                         in.close();
583                                         fileIn.close();
583                                         } catch (
584                                         ClassNotFoundException e1) {
584                                         e1.printStackTrace();
585                                         } catch (
586                                         FileNotFoundException e1) {
586                                         e1.printStackTrace();
587                                         } catch (IOException e1) {
587                                         e1.printStackTrace();
588                                         }
589                                         }
590                                         final ArrayList<Portfolio>
590                                         ports = port2;
591                                         Portfolio p = null;
592                                         for(Portfolio portfolio1 :
593                                         ports) {
593                                         if(portfolio1.loginID.
594                                         equals(name)) {
594                                         p = portfolio1;
595                                         break;
596                                         }
597                                         }
598                                         }
599                                         FileChooser fileChooser = new
599                                         FileChooser();
600                                         fileChooser.setTitle("Open
Resource File");
601                                         File file = fileChooser.
602                                         showOpenDialog(stage);
603                                         if(file != null) {
603                                         EquitiesView equitiesView
604                                         = new EquitiesView();
604                                         CAView caView = new
605                                         transactionView
606                                         transactionView = new transactionView();
606                                         WatchlistView
607                                         watchlistView = new WatchlistView();
607                                         p.importP(file.toString())
608                                         ,p);
608                                         caView.saveCA(p);
609                                         
```

```

610                               equitiesView.saveEquities
611                               (p);
612                               transactionView.
613                               saveHistory(p);
614                               watchlistView.
615                               saveWatchlist(p);
616                               p.saveMAs(p);
617                               }
618                               Scene scene = login(
619                               primaryStage);
620                               stage.setScene(scene);
621                               }
622
623                               }
624                               }
625                               }
626                               });
627
628                               Button cancel = new Button("Cancel");
629                               grid.add(cancel,1,5);
630                               cancel.setOnAction(new EventHandler<ActionEvent>(
631                               {
632                                   @Override
633                                   public void handle(ActionEvent e) {
634                                       Scene scene = login(primaryStage);
635                                       stage.setScene(scene);
636                                   }
637                               });
638
639                               Scene scene = new Scene(grid, 1200, 700);
640                               return scene;
641                           }
642
643
644                           //GUI for portfolio
645                           public Scene portfolio(final Stage primaryStage,
646                           Portfolio p){
646                               Text title = new Text("Successfully logged into "
+ p.loginID + "'s Portfolio");

```

```
647         BorderPane border = borderPane(primaryStage, p,
648                                         title);
649         Scene scene = new Scene(border, 1200, 700);
650         return scene;
651     }
652
653
654
```

```

1 package trunk.View;
2
3 import javafx.collections.FXCollections;
4 import javafx.collections.ObservableList;
5 import javafx.event.ActionEvent;
6 import javafx.event.EventHandler;
7 import javafx.geometry.Insets;
8 import javafx.geometry.Orientation;
9 import javafx.geometry.Pos;
10 import javafx.scene.Scene;
11 import javafx.scene.control.*;
12 import javafx.scene.control.cell.PropertyValueFactory;
13 import javafx.scene.layout.BorderPane;
14 import javafx.scene.layout.GridPane;
15 import javafx.scene.layout.VBox;
16 import javafx.scene.paint.Color;
17 import javafx.scene.text.Text;
18 import javafx.stage.FileChooser;
19 import javafx.stage.Stage;
20 import trunk.Model.CashAccount;
21 import trunk.Model.Portfolio;
22 import trunk.Model.UndoAction;
23
24 import java.io.*;
25 import java.util.ArrayDeque;
26 import java.util.ArrayList;
27 import java.util.Optional;
28
29 /**
30 * Created by Daniel on 3/25/2016.
31 */
32
33 public class CAView {
34     /*
35      * Code for saving cash accounts
36     */
37     public void saveCA(Portfolio p) {
38         ArrayList<Portfolio> port = null;
39         try {
40             FileInputStream fileIn = new FileInputStream("myfile");
41             ObjectInputStream in = new ObjectInputStream(
42                 fileIn);
43             port = (ArrayList<Portfolio>) in.readObject();
44             in.close();

```

```

44             fileIn.close();
45
46         } catch (ClassNotFoundException ee) {
47             ee.printStackTrace();
48         } catch (FileNotFoundException ee) {
49             ee.printStackTrace();
50         } catch (IOException ee) {
51             ee.printStackTrace();
52         }
53     if(port != null) {
54         port.forEach(iter -> {
55             iter.cashAccList = p.cashAccList;
56         });
57     }
58
59     try
60     {
61         FileOutputStream fileOut =
62             new FileOutputStream("myfile");
63         ObjectOutputStream out = new
64             ObjectOutputStream(fileOut);
65         out.writeObject(port);
66         out.close();
67         fileOut.close();
68     } catch(IOException i)
69     {
70         i.printStackTrace();
71     }
72     /*
73      * GUI for Cash Accounts page
74      */
75     public Scene CashAccounts(final Stage primaryStage,
76     Portfolio p, FPTS FPTS) {
77         Text title = new Text(p.loginID + "'s Cash
78         Accounts");
79         BorderPane border = FPTS.borderPane(primaryStage,
80         p, title);
81
82         VBox left = new VBox();
83
84        ToolBar toolBar2 = new ToolBar();
85         toolBar2.setOrientation(Orientation.VERTICAL);
86
87         left.getChildren().add(toolBar2);

```

```
85         left.setSpacing(10);
86
87     final VBox vbox = new VBox();
88     vbox.setSpacing(5);
89     vbox.setPadding(new Insets(10, 0, 0, 10));
90
91     final ObservableList<CashAccount> cashAccounts =
92         FXCollections.observableArrayList(p.cashAccList);
93     TableView table = new TableView();
94     table.setEditable(true);
95
96     vbox.getChildren().addAll(title, table);
97
98     TableColumn name = new TableColumn("Account Name");
99     TableColumn amount = new TableColumn("Balance");
100    TableColumn date = new TableColumn("Date Created");
101
102    name.setCellValueFactory(
103        new PropertyValueFactory<CashAccount,
104        String>("name")
105    );
106    amount.setCellValueFactory(
107        new PropertyValueFactory<CashAccount,
108        Double>("balance")
109    );
110    date.setCellValueFactory(
111        new PropertyValueFactory<CashAccount,
112        String>("date")
113    );
114    Label spacer = new Label(" ");
115    Button add = new Button("Add Account");
116    Button remove = new Button("Remove Account");
117    Button deposit = new Button("Deposit Funds");
118    Button withdrawal = new Button("Withdraw Funds");
119    Button transfer = new Button("Transfer Funds");
120    Button exportCA = new Button("Export Account");
121    Button importCA = new Button("Import Account");
122
123    toolBar2.getItems().addAll(spacer, add, remove,
124    deposit, withdrawal, transfer, exportCA, importCA);
125
126    transfer.setOnAction(new EventHandler<ActionEvent> {
127        >() {
```

```

122             @Override
123             public void handle(ActionEvent event) {
124                 final Stage dialog = new Stage();
125                 dialog.initOwner(primaryStage);
126                 GridPane grid2 = new GridPane();
127                 grid2.setAlignment(Pos.CENTER);
128                 grid2.setHgap(10);
129                 grid2.setVgap(10);
130                 grid2.setPadding(new Insets(5, 10, 5, 10)
131             );
132
133             TextField to = new TextField();
134             grid2.add(new Label("Account to Transfer
135             to: "), 0, 1);
136
137             TextField from = new TextField();
138             grid2.add(new Label("Account to Transfer
139             from: "), 0, 2);
140
141             TextField num = new TextField();
142             grid2.add(new Label("Amount to Transfer
143             : "), 0, 3);
144
145             Scene dialogScene = new Scene(grid2, 400,
146             400);
147
148             dialog.setScene(dialogScene);
149             dialog.show();
150             Button cancel = new Button("Cancel");
151             grid2.add(cancel, 1, 7);
152             Button submit = new Button("Submit");
153             grid2.add(submit, 0, 7);
154             cancel.setOnAction(new EventHandler<
155             ActionEvent>() {
156
157                 @Override
158                 public void handle(ActionEvent e) {
159                     dialog.close();
160                 }
161             });
162             submit.setOnAction(new EventHandler<
163             ActionEvent>() {
164
165                 @Override
166                 public void handle(ActionEvent event)
167             }

```

```

159                         CashAccount c1 = new CashAccount(
      "", "", 0, "", null);
160                         CashAccount c2 = new CashAccount(
      "", "", 0, "", null);
161                         for(CashAccount c : p.cashAccList
162 ) {
163                         if(c.getName().equals(to.
      getText().trim())) {
164                             c1 = c;
165                             break;
166                         }
167                         for(CashAccount c : p.cashAccList
168 ) {
169                         if(c.getName().equals(from.
      getText().trim())) {
170                             c2 = c;
171                             break;
172                         }
173                         final Text error = new Text();
174                         grid2.add(error,1,5);
175                         if(Double.parseDouble(num.getText
      ()) > c2.balance) {
176                             error.setFill(Color.FIREBRICK
      );
177                             error.setText("Cannot
      transfer more than the account's balance.");
178                         }
179                         else if(Double.parseDouble(num.
      getText()) < 0) {
180                             error.setFill(Color.FIREBRICK
      );
181                             error.setText("Enter a valid
      amount of money");
182                         }
183                         else {
184                             CashAccount c = new
      CashAccount("", "", 0, "", null);
185                             c.transfer(c1, c2, Double.
      parseDouble(num.getText()),p);
186                             UndoAction transferCA = new
      UndoAction("Transfer", "CashAccount", c1, c2, Double.
      parseDouble(num.getText()),null, "");

```

```

188                     transferCA.addToQueue(
189                         transferCA);
190
191                     dialog.close();
192                     Scene scene = CashAccounts(
193                         primaryStage, p, FPTS);
194
195                     FPTS.stage.setScene(scene);
196
197                     }
198
199                     remove.setOnAction(new EventHandler<ActionEvent>(
200                         ) {
201
202                             @Override
203                             public void handle(ActionEvent event) {
204
205                                 ArrayList<String> choices = new ArrayList<
206                                     >();
207
208                                 for(CashAccount c : p.cashAccList) {
209                                     choices.add(c.name);
210
211                                 }
212
213                                 ChoiceDialog<String> dialog = new
214                                     ChoiceDialog<>("None", choices);
215
216                                 dialog.setContentText("Choose the cash
217                                     account you wish to remove:");
218
219                                 Optional<String> result = dialog.
220                                     showAndWait();
221
222                                 if (result.isPresent()) {
223                                     CashAccount c = new CashAccount("", ""
224                                         , 0, "", null);
225
226                                     for (CashAccount n : p.cashAccList) {
227                                         if (n.getName().equals(result.get()
228                                             )) {
229
230                                             c = n;
231                                         }
232                                         c.deleteCashAccount(result.get(), p);
233
234                                         UndoAction add = new UndoAction(""
235                                             Remove", "CashAccount", c, null, 0, null, "");

```

```

223                     add.redo.clear();
224                     add.addToQueue(add);
225
226                     Scene scene = CashAccounts(
227                         primaryStage, p, FPTS);
228                         FPTS.stage.setScene(scene);
229                     }
230                 });
231
232             //Action for importing cash account
233             importCA.setOnAction(new EventHandler<ActionEvent>()
234             {
235                 @Override
236                 public void handle(ActionEvent event) {
237                     FileChooser fileChooser = new FileChooser();
238                     fileChooser.setTitle("Open Resource File");
239                     File file = fileChooser.showOpenDialog(
240                         FPTS.stage);
241                     if(file != null){
242                         ArrayList<CashAccount> existingAccts
243                         = new ArrayList<CashAccount>();
244                         CashAccount c = new CashAccount("", "", "", 0, "", p);
245                         existingAccts = c.importCA(file.toString());
246                         UndoAction add = new UndoAction("Import", "CashAccount", null, null, 0, null, file.toString());
247                         add.redo.clear();
248                         add.addToQueue(add);
249
250                         for(CashAccount exAcct :
251                             existingAccts){
252                             Text title = new Text("Account "
253                             + exAcct.name + " already exists");
254                             final Stage dialog = new Stage();
255                             dialog.initOwner(primaryStage);
256                             GridPane grid2 = new GridPane();
257                             grid2.setAlignment(Pos.CENTER);
258                             grid2.setHgap(10);
259                             grid2.setVgap(10);
260                             grid2.setPadding(new Insets(5, 10));
261                         }
262                     }
263                 }
264             });
265         }
266     }
267 }
```



```

285                     dialog.close();
286                     dialog.close();
287                 }
288             });
289         cancel.setOnAction(new
290             EventHandler<ActionEvent>() {
291                 @Override
292                 public void handle(
293                     ActionEvent event) {
294                     dialog.close();
295                 }
296             });
297         Scene dialogScene = new Scene(
298             grid2, 300, 200);
299         dialog.setScene(dialogScene);
300         dialog.show();
301     }
302 }
303 }
304 });
305
306 //Action for depositing funds to a cash account
307 deposit.setOnAction(new EventHandler<ActionEvent>
308 () {
309     @Override
310     public void handle(ActionEvent event) {
311         ArrayList<String> choices = new ArrayList
312             <>();
313         for(CashAccount e : p.cashAccList) {
314             if(!e.name.equals("")) {
315                 choices.add(e.name);
316             }
317         }
318         ChoiceDialog<String> dialog = new
319             ChoiceDialog<>("None", choices);
320         dialog.setContentText("Choose the cash
321             account to deposit to:");
322         Optional<String> result = dialog.
323             showAndWait();
324         if (result.isPresent()) {

```

```

321                     final Stage dialog2 = new Stage();
322                     dialog2.initOwner(primaryStage);
323                     GridPane grid2 = new GridPane();
324                     grid2.setAlignment(Pos.CENTER);
325                     grid2.setHgap(10);
326                     grid2.setVgap(10);
327                     grid2.setPadding(new Insets(5, 10, 5,
328                                         10));
328
329                     VBox dialogVbox = new VBox(20);
330                     dialogVbox.getChildren().add(new Text
331                         ("This is a Dialog"));
332                     TextField shares = new TextField();
333
334                     grid2.add(new Label("Deposit to " +
335                         result.get() + ": "), 0, 1);
336                     grid2.add(shares, 1, 1);
337                     Scene dialogScene = new Scene(grid2,
338                         400, 400);
339                     Button cancel = new Button("Cancel");
340                     grid2.add(cancel, 1, 3);
341                     Button submit = new Button("Submit");
342                     grid2.add(submit, 0, 3);
343                     cancel.setOnAction(new EventHandler<
344                         ActionEvent>() {
345                         @Override
346                         public void handle(ActionEvent e)
347                             {dialog2.close(); }
348                         });
349                     final Text error = new Text();
350                     grid2.add(error, 1, 8);
351                     submit.setOnAction(new EventHandler<
352                         ActionEvent>() { // this one
353                         @Override
354                         public void handle(ActionEvent
355                             event) {
356                             if(shares.getText().isEmpty())
357                             {
358                                 error.setFill(Color.
359                                     FIREBRICK);
360                                 error.setText("Fill in
361                                     all fields");
362                             }
363                             else if(!FPTS.isInt(shares.
364                                 getText())))

```

```

354                     {
355                         error.setFill(Color.
356                         FIREBRICK);
357                         error.setText("Enter
358                         information in correct format");
359                     }
360                     else if(Integer.parseInt(
361                         shares.getText()) < 1) {
362                         error.setFill(Color.
363                         FIREBRICK);
364                         error.setText("Enter
365                         valid number");
366                     }
367                     else{
368                         int numShares = Integer.
369                         parseInt(shares.getText());
370                         for(CashAccount e : p.
371                             cashAccList){
372                             if(e.name.equals(
373                                 result.get())))
374                             e.balance +=
375                                 numShares;
376                         }
377                         dialog2.close();
378                         Scene scene =
379                         CashAccounts(primaryStage,p, FPTS);
380                         FPTS.stage.setScene(scene
381                         );
382                     }
383                     }
384                     //Action for withdrawing funds from a cash
385                     account
386                     withdrawal.setOnAction(new EventHandler<
387                         ActionEvent>() {

```

```

386             @Override
387             public void handle(ActionEvent event) {
388                 ArrayList<String> choices = new ArrayList
389                 <>();
390                 for(CashAccount e : p.cashAccList) {
391                     if(!e.name.equals("")) {
392                         choices.add(e.name);
393                     }
394                 ChoiceDialog<String> dialog = new
395                 ChoiceDialog<>("None", choices);
396                 dialog.setContentText("Choose the cash
397                 account to withdraw from:");
398                 Optional<String> result = dialog.
399                 showAndWait();
400                 if (result.isPresent()) {
401                     final Stage dialog2 = new Stage();
402                     dialog2.initOwner(primaryStage);
403                     GridPane grid2 = new GridPane();
404                     grid2.setAlignment(Pos.CENTER);
405                     grid2.setHgap(10);
406                     grid2.setVgap(10);
407                     grid2.setPadding(new Insets(5, 10, 5,
408                     10));
409                     VBox dialogVbox = new VBox(20);
410                     dialogVbox.getChildren().add(new Text
411                     ("This is a Dialog"));
412                     TextField amount = new TextField();
413
414                     grid2.add(new Label("Withdraw from "
415                     + result.get() + ":"),0,1);
416                     grid2.add(amount,1,1);
417                     Scene dialogScene = new Scene(grid2,
418                     400, 400);
419                     Button cancel = new Button("Cancel");
420                     grid2.add(cancel,1,3);
421                     Button submit = new Button("Submit");
422                     grid2.add(submit,0,3);
423                     cancel.setOnAction(new EventHandler<
424                     ActionEvent>() {
425                         @Override
426                         public void handle(ActionEvent e)
427                         {dialog2.close();}}
```

```

421                     });
422                     final Text error = new Text();
423                     grid2.add(error, 1, 8);
424                     submit.setOnAction(new EventHandler<
425                         ActionEvent>() { // this one
426                             @Override
427                             public void handle(ActionEvent
428                                 event) {
429                                     if(amount.getText().isEmpty())
430                                         ) {
431                                             error.setFill(Color.
432                                                 FIREBRICK);
433                                             error.setText("Fill in
434                                                 all fields");
435                                         }
436                                         else if(!FPTS.toInt(amount.
437                                             getText())))
438                                         {
439                                             error.setFill(Color.
440                                                 FIREBRICK);
441                                             error.setText("Enter
442                                                 valid number");
443                                         }
444                                         else{
445                                             int amountInt = Integer.
446                                             parseInt(amount.getText());
447                                             for(CashAccount e : p.
448                                                 cashAccList){
449                                                 result.get());
450                                                 if(e.name.equals(
451                                                     amountInt) {
452                                                     amountInt;
453                                                     e.balance >
454                                                         e.balance ==
455                                                         dialog2.close
456                                         );
457                                         Scene scene =

```

```

448     CashAccounts(primaryStage, p, FPTS);
449                                         FPTS.stage.
450             setScene(scene);
451         } else{
452             error.setFill
453             (Color.FIREBRICK);
454             error.setText
455             ("Cannot overdraw account");
456         }
457     }
458 );
459
460         dialog2.setScene(dialogScene);
461         dialog2.show();
462     }
463     }
464 );
465
466 //Action for exporting a cash account
467 exportCA.setOnAction(new EventHandler<ActionEvent
>() {
468     @Override
469     public void handle(ActionEvent event) {
470         final Stage dialog = new Stage();
471         dialog.initOwner(primaryStage);
472         GridPane grid2 = new GridPane();
473         grid2.setAlignment(Pos.CENTER);
474         grid2.setHgap(10);
475         grid2.setVgap(10);
476         grid2.setPadding(new Insets(5, 10, 5, 10)
);
477
478         TextField file = new TextField();
479         grid2.add(new Label("Filename: "), 0, 1);
480         grid2.add(file, 1, 1);
481
482         Scene dialogScene = new Scene(grid2, 400,
400);
483         dialog.setScene(dialogScene);
484         dialog.show();
485         Button cancel = new Button("Cancel");
486         grid2.add(cancel, 1, 7);

```

```

487             Button submit = new Button("Submit");
488             grid2.add(submit,0,7);
489             cancel.setOnAction(new EventHandler<
490                 ActionEvent>() {
491                     @Override
492                     public void handle(ActionEvent e) {
493                         dialog.close();
494                     }
495                     });
496             submit.setOnAction(new EventHandler<
497                 ActionEvent>() {
498                     @Override
499                     public void handle(ActionEvent event)
500                     {
501                         CashAccount c = new CashAccount(
502                             "", "", 0, "", null);
503                         c.exportCA(file.getText(), p);
504                         dialog.close();
505                     }
506                     });
507             });
508             add.setOnAction(new EventHandler<ActionEvent>() {
509                 @Override
510                 public void handle(ActionEvent event) {
511                     final Stage dialog = new Stage();
512                     dialog.initOwner(primaryStage);
513                     GridPane grid2 = new GridPane();
514                     grid2.setAlignment(Pos.CENTER);
515                     grid2.setHgap(10);
516                     grid2.setVgap(10);
517                     grid2.setPadding(new Insets(5, 10, 5, 10));
518                     Label name = new Label("Name:");
519                     grid2.add(name, 0, 1);
520                     final TextField CName = new TextField();
521                     grid2.add(CName, 1, 1);
522                     Label amount = new Label("Balance:");
523                     grid2.add(amount, 0, 2);
524                     TextField Iamount = new TextField();
525                     grid2.add(Iamount, 1, 2);
526             });

```

```

527             Label date = new Label("Date:");
528             grid2.add(date, 0, 3);
529             TextField CAdate = new TextField();
530             grid2.add(CAdate, 1, 3);
531
532             final Text error = new Text();
533             grid2.add(error, 1, 6);
534             Button submit = new Button("Submit");
535             grid2.add(submit, 1, 4);
536             submit.setOnAction(new EventHandler<
537                 ActionEvent>() {
538                     @Override
539                     public void handle(ActionEvent event)
540                     {
541                         boolean same = false;
542                         for(CashAccount c : p.cashAccList
543                             ) {
544                             if(c.getName().equals(CAname.
545                                 getText())) {
546                                 same = true;
547                                 break;
548                             }
549                         }
550                         if(Double.parseDouble(Iamount.
551                             getText()) > 0) {
552                             if (!same) {
553                                 CashAccount c = new
554                                     CashAccount("", "", 0, "", p);
555                                     c.addCashAccount(CAname.
556                                         getText(), Double.parseDouble(Iamount.getText()), CAdate.
557                                         getText(), p);
558                                     CashAccount temp = null;
559                                     for (CashAccount n : p.
560                                         cashAccList) {
561                                         if (n.getName().
562                                             equals(CAname.getText())) {
563                                             temp = n;
564                                         }
565                                         }
566                                         UndoAction add = new
567                                         UndoAction("Add", "CashAccount", temp,null,0,null,"");
568                                         add.redo.clear();
569                                         add.addToQueue(add);
570                                         Scene scene =

```

```

560 CashAccounts(primaryStage, p, FPTS);
561
562                                     FPTS.stage.setScene(scene
563                                         );
564 } else if (same) {
565     dialog.close();
566     Text title = new Text(""
567         Account already exists");
568     final Stage dialog = new
569         Stage();
570     dialog.initOwner(
571         primaryStage);
572     GridPane grid2 = new
573         GridPane();
574     grid2.setAlignment(Pos.
575         CENTER);
576     grid2.setHgap(10);
577     grid2.setVgap(10);
578     grid2.setPadding(new
579         Insets(5, 10, 5, 10));
580     grid2.add(title, 0, 0);
581
582     Button ignore = new
583         Button("Overwrite");
584     ignore.setOnAction(new
585         EventHandler<ActionEvent>() {
586             @Override
587             public void handle(
588                 ActionEvent event) {
589                     CashAccount c =
590                     new CashAccount("", "", 0, "", p);
591                     c.addCashAccount(
592                         Cname.getText(), Double.parseDouble(Iamount.getText()),
593                         CAdate.getText(), p);
594
595                     Scene scene =

```

```

589 CashAccounts(primaryStage, p, FPTS);
590
591                                     FPTS.stage.
592             setScene(scene);
593             dialog.close();
594         }
595     });
596     add.setOnAction(new
597         EventHandler<ActionEvent>() {
598             @Override
599             public void handle(
600                 ActionEvent event) {
601                 CashAccount c =
602                     new CashAccount("", "", 0, "", null);
603                     c.updateBalance(
604                         CAname.getText(), Double.parseDouble(Iamount.getText()),
605                         true, p);
606
607                 Scene scene =
608                     CashAccounts(primaryStage, p, FPTS);
609
610                 FPTS.stage.
611                     setScene(scene);
612                     dialog.close();
613                     dialog.close();
614                     }
615                     );
616                     }
617             } else {
618                 error.setFill(Color.FIREBRICK
619             );
620                 error.setText("Enter a valid
621 amount of money");

```

```
620 }  
621 }  
622 } );  
623  
624 Scene dialogScene = new Scene(grid2, 300,  
625 200);  
626 dialog.setScene(dialogScene);  
627 dialog.show();  
628 } );  
629  
630  
631 table.setItems(cashAccounts);  
632 table.getColumns().addAll(name, amount, date);  
633 name.prefWidthProperty().bind(table.widthProperty()  
634 .multiply(0.3));  
635 amount.prefWidthProperty().bind(table.  
widthProperty().multiply(0.1));  
636 date.prefWidthProperty().bind(table.widthProperty()  
637 .multiply(0.075));  
638  
639 border.setCenter(vbox);  
640 border.setLeft(left);  
641 Scene scene = new Scene(border, 1200, 700);  
642 return scene;  
643 }
```

```

1 package trunk.View;
2
3 import javafx.collections.FXCollections;
4 import javafx.collections.ObservableList;
5 import javafx.collections.transformation.FilteredList;
6 import javafx.event.ActionEvent;
7 import javafx.event.EventHandler;
8 import javafx.geometry.Insets;
9 import javafx.geometry.Orientation;
10 import javafx.geometry.Pos;
11 import javafx.scene.Scene;
12 import javafx.scene.control.*;
13 import javafx.scene.control.cell.PropertyValueFactory;
14 import javafx.scene.layout.*;
15 import javafx.scene.paint.Color;
16 import javafx.scene.text.Font;
17 import javafx.scene.text.Text;
18 import javafx.stage.FileChooser;
19 import javafx.stage.Stage;
20 import javafx.util.converter.DateTimeStringConverter;
21 import trunk.Control.HistoryInvoker;
22 import trunk.Control.UpdateEQ;
23 import trunk.Model.*;
24
25 import java.io.*;
26 import java.text.DateFormat;
27 import java.text.SimpleDateFormat;
28 import java.util.ArrayList;
29 import java.util.Date;
30 import java.util.Optional;
31
32 /**
33 * Created by Daniel on 3/25/2016.
34 */
35 public class EquitiesView {
36     /*
37         * Code for saving equities
38     */
39     public void saveEquities(Portfolio p) {
40         ArrayList<Portfolio> port = null;
41         try {
42             FileInputStream fileIn = new FileInputStream("myfile");
43             ObjectInputStream in = new ObjectInputStream(

```

File - C:\Users\hack master\Desktop\school\Semester 4\SWEN-262 (FPTS)\svn\trunk\View\EquitiesView.java

```
44         port = (ArrayList<Portfolio>) in.readObject();
45         in.close();
46         fileIn.close();
47     } catch (ClassNotFoundException ee) {
48         ee.printStackTrace();
49     } catch (FileNotFoundException ee) {
50         ee.printStackTrace();
51     } catch (IOException ee) {
52         ee.printStackTrace();
53     }
54     port.forEach(iter -> {
55         iter.equityList1 = p.equityList1;
56         iter.equityList2 = p.equityList2;
57     });
58
59     try {
60         FileOutputStream fileOut =
61             new FileOutputStream("myfile");
62         ObjectOutputStream out = new
63             ObjectOutputStream(fileOut);
64         out.writeObject(port);
65         out.close();
66         fileOut.close();
67     }catch(IOException i)
68     {
69         i.printStackTrace();
70     }
71 }
72
73 /*
74 * Code for filtering equities
75 */
76 public void filter(FilteredList<Equity> filteredData,
77     FilteredList<Equity> filteredData2, TextField tickerFilter
78 , TextField nameFilter, TextField maFilter, String
79     newValue) {
80     filteredData.setPredicate( equity -> {
81         if (newValue == null || newValue.isEmpty()) {
82             return true;
83         }
84         boolean exists = false;
85         if(equity.getMarketAverage().isEmpty() &&
86             maFilter.getText().equals("")) {
87             exists = true;
88         }
89         else if(nameFilter.getText().toLowerCase().contains(equity.getName().toLowerCase())) {
90             exists = true;
91         }
92         else if(tickerFilter.getText().contains(equity.getTicker().toLowerCase())) {
93             exists = true;
94         }
95         else if(maFilter.getText().contains(equity.getMarketAverage().toString().toLowerCase())))
96             exists = true;
97         else
98             exists = false;
99     }
100 }
```

```

84         }
85
86         for(String s : equity.getMarketAverage()) {
87             if(s.toLowerCase().contains(maFilter.
88                 getText().toLowerCase())))
89                 exists = true;
90             }
91             if(equity.getTickerSymbol().toLowerCase() .
92                 contains(tickerFilter.getText().toLowerCase())
93                     && equity.getName().toLowerCase() .
94                     contains(nameFilter.getText().toLowerCase())
95                         && exists
96                     )
97             return true;
98         }
99     }
100    );
101    filteredData2.setPredicate(equity -> {
102        if (newValue == null || newValue.isEmpty()) {
103            return true;
104        }
105        boolean exists = false;
106        if(equity.getMarketAverage().isEmpty() &&
107            maFilter.getText().equals("")){
108                exists = true;
109            }
110            for(String s : equity.getMarketAverage()) {
111                if(s.toLowerCase().contains(maFilter.
112                    getText().toLowerCase())))
113                    exists = true;
114                }
115                if(equity.getTickerSymbol().toLowerCase() .
116                    contains(tickerFilter.getText().toLowerCase())
117                        && equity.getName().toLowerCase() .
118                        contains(nameFilter.getText().toLowerCase())
119                            && exists
120                        )
121                return true;
122            }
123        else{

```

```

122                     return false;
123                 }
124             });
125         }
126
127     /*
128      * GUI for Equities page
129     */
130
130     public Scene ViewEquities(Stage primaryStage,
131     Portfolio p, FPTS FPTS) {
132
133         Text title = new Text(p.loginID + "'s Equities");
134         BorderPane border = FPTS.borderPane(primaryStage,
135         p, title);
136
137         VBox left = new VBox(5);
138
139         ToolBar toolBar2 = new ToolBar();
140         toolBar2.setOrientation(Orientation.VERTICAL);
141         Label Search = new Label("Search Equities");
142         left.setVgrow(Search, Priority.ALWAYS);
143         Search.setMaxHeight(Double.MAX_VALUE);
144         left.getChildren().addAll(toolBar2, Search);
145         left.setSpacing(10);
146
147         final Label ownedE = new Label(p.loginID + "'s
148         Equities");
149         ownedE.setFont(new Font("Arial", 20));
150         final Label availableE = new Label("Available
151         Equities");
152         availableE.setFont(new Font("Arial", 20));
153
154         TableView tableOwnedE = new TableView();
155         tableOwnedE.setEditable(true);
156
156         TableColumn symbols = new TableColumn("Ticker
157         Symbol");
158         TableColumn names = new TableColumn("Names");
159         TableColumn shares = new TableColumn("Shares");
160         TableColumn salePrice = new TableColumn("Sale
161         Price");
162         TableColumn acquisitionDate = new TableColumn("Acquisition Date");
163         TableColumn MarketAverage = new TableColumn("Market Average");
164         TableColumn totalValue = new TableColumn("Total
165         Value");
166
167

```

```

158         TableColumn symbols2 = new TableColumn("Ticker
159             Symbol) ;
160         TableColumn names2 = new TableColumn("Names") ;
161         TableColumn salePrice2 = new TableColumn("Sale
162             Price") ;
163         TableColumn MarketAverage2 = new TableColumn("Market
164             Average") ;
165
166         Label spacer = new Label(" ") ;
167
168         final VBox vbox = new VBox();
169         vbox.setSpacing(5);
170         vbox.setPadding(new Insets(10, 0, 0, 10));
171         vbox.getChildren().addAll(ownedE, tableOwnedE,
172             availableE, tableAvailableE);
173         border.setLeft(left);
174         final ObservableList<Equity> equities1 =
175             FXCollections.observableArrayList(p.equityList1);
176         final ObservableList<Equity> equities2 =
177             FXCollections.observableArrayList(p.equityList2);
178
179         symbols.setCellValueFactory(
180             new PropertyValueFactory<Equity, String>("tickerSymbol")
181         );
182         names.setCellValueFactory(
183             new PropertyValueFactory<Equity, String>("name")
184         );
185         shares.setCellValueFactory(
186             new PropertyValueFactory<Equity, Integer>(
187                 "shares")
188         );
189         salePrice.setCellValueFactory(
190             new PropertyValueFactory<Equity, Double>("acquisitionPrice")
191         );
192         acquisitionDate.setCellValueFactory(
193             new PropertyValueFactory<Equity, Date>("acquisitionDate")
194         );
195         MarketAverage.setCellValueFactory(

```

```

192             new PropertyValueFactory<Equity, String>("MA")
193         );
194         totalValue.setCellValueFactory(
195             new PropertyValueFactory<Equity, Double>(
196                 "total"
197             );
198
199         symbols2.setCellValueFactory(
200             new PropertyValueFactory<Equity, String>("tickerSymbol")
201         );
202         names2.setCellValueFactory(
203             new PropertyValueFactory<Equity, String>("name")
204         );
205         salePrice2.setCellValueFactory(
206             new PropertyValueFactory<Equity, Double>("acquisitionPrice")
207         );
208         MarketAverage2.setCellValueFactory(
209             new PropertyValueFactory<Equity, String>("MA")
210         );
211
212
213         tableOwnedE.setItems(equities1);
214         tableOwnedE.getColumns().addAll(symbols, names,
215             shares, salePrice, acquisitionDate, MarketAverage,
216             totalValue);
217         symbols.prefWidthProperty().bind(tableOwnedE.
218             widthProperty().multiply(0.1));
219         names.prefWidthProperty().bind(tableOwnedE.
220             widthProperty().multiply(0.35));
221         shares.prefWidthProperty().bind(tableOwnedE.
222             widthProperty().multiply(0.05));
223         salePrice.prefWidthProperty().bind(tableOwnedE.
224             widthProperty().multiply(0.075));
225         acquisitionDate.prefWidthProperty().bind(
226             tableOwnedE.widthProperty().multiply(0.1));
227         MarketAverage.prefWidthProperty().bind(
228             tableOwnedE.widthProperty().multiply(0.15));
229         totalValue.prefWidthProperty().bind(tableOwnedE.
230             widthProperty().multiply(0.125));

```

```

222
223         tableAvailableE.setItems(equities2);
224         tableAvailableE.getColumns().addAll(symbols2,
225             names2, salePrice2, MarketAverage2);
226         symbols2.prefWidthProperty().bind(tableAvailableE
227             .widthProperty().multiply(0.1));
228         names2.prefWidthProperty().bind(tableAvailableE.
229             widthProperty().multiply(0.35));
230         salePrice2.prefWidthProperty().bind(
231             tableAvailableE.widthProperty().multiply(0.075));
232         MarketAverage2.prefWidthProperty().bind(
233             tableAvailableE.widthProperty().multiply(0.125));
234
235         TableView mas = new TableView();
236         final ObservableList<Equity> market =
237             FXCollections.observableArrayList(p.marketAverages);
238
239         TableColumn salePrice3 = new TableColumn("Sale
240             Price");
241         TableColumn MarketAverage3 = new TableColumn("Market Average");
242         TableColumn shares3 = new TableColumn("Shares");
243
244         salePrice3.setCellValueFactory(
245             new PropertyValueFactory<Equity, Double>("acquisitionPrice")
246         );
247         MarketAverage3.setCellValueFactory(
248             new PropertyValueFactory<Equity, String>("MA")
249         );
250         shares3.setCellValueFactory(
251             new PropertyValueFactory<Equity, Double>("shares")
252         );
253
254         mas.setItems(market);
255         mas.getColumns().addAll(MarketAverage3, salePrice3
256             , shares3);
257
258         Button add = new Button("Add Equity");
259         Button remove = new Button("Remove Equity");
260         Button importEq = new Button("Import Equity");
261         Button exportEq = new Button("Export Equity");

```

```

255
256         VBox.setVgrow(Search, Priority.ALWAYS);
257         Search.setMaxHeight(Double.MAX_VALUE);
258
259         Button addShares = new Button("Add Shares");
260         Button sellShares = new Button("Sell Shares");
261         Button addSharestoMA = new Button("Add Shares to
262             MA");
263
264         Button goback = new Button("View Portfolio");
265
266         //Action for importing equities
267         importEq.setOnAction(new EventHandler<ActionEvent>()
268             >() {
269                 @Override
270                 public void handle(ActionEvent event) {
271                     FileChooser fileChooser = new FileChooser();
272                     fileChooser.setTitle("Open Resource File");
273                     File file = fileChooser.showOpenDialog(
274                         FPTS.stage);
275                     if(file != null){
276                         Equity e = new Equity("", "", 0, 0, "", null);
277                         e.importEQ(file.toString(), p);
278                         UndoAction add = new UndoAction("Import",
279                             "Equity", null, null, 0, null, file.toString());
280                         add.redo.clear();
281                         add.addToQueue(add);
282                         try{
283                             UpdateEQ updateEQ = new UpdateEQ();
284                             updateEQ.updateEquities(p, p.
285                                 equityList2);
286                             }catch(Exception ex){
287                                 ex.getMessage();
288                             }
289                         Scene scene = ViewEquities(
290                             primaryStage, p, FPTS);
291                         FPTS.stage.setScene(scene);
292                     }
293                 });

```

```

290         //Action for exporting equities
291         exportEq.setOnAction(new EventHandler<ActionEvent>()
292             >() {
293                 @Override
294                 public void handle(ActionEvent event) {
295                     final Stage dialog = new Stage();
296                     dialog.initOwner(primaryStage);
297                     GridPane grid2 = new GridPane();
298                     grid2.setAlignment(Pos.CENTER);
299                     grid2.setHgap(10);
300                     grid2.setVgap(10);
301                     grid2.setPadding(new Insets(5, 10, 5, 10));
302
303                     TextField file = new TextField();
304                     grid2.add(new Label("Filename: "), 0, 1);
305                     grid2.add(file, 1, 1);
306
307                     Scene dialogScene = new Scene(grid2, 400,
308                         400);
309                     dialog.setScene(dialogScene);
310                     dialog.show();
311                     Button cancel = new Button("Cancel");
312                     grid2.add(cancel, 1, 7);
313                     Button submit = new Button("Submit");
314                     grid2.add(submit, 0, 7);
315                     cancel.setOnAction(new EventHandler<
316                         ActionEvent>() {
317                             @Override
318                             public void handle(ActionEvent e) {
319                                 dialog.close();
320                             }
321                         });
322                     submit.setOnAction(new EventHandler<
323                         ActionEvent>() {
324                             @Override
325                             public void handle(ActionEvent event)
326                             {
327                                 Equity e = new Equity("", "", 0, 0,
328                                     "", null);
329                                 e.exportEQ(file.getText(), p);
330                                 dialog.close();
331                             }
332                         });
333                 }
334             });

```

```

328         });
329
330         //Action for going back to portfolio
331         goback.setOnAction(new EventHandler<ActionEvent>(
332             {
333                 @Override
334                 public void handle(ActionEvent e) {
335                     Scene scene = FPTS.portfolio(primaryStage
336                     ,p);
337                     FPTS.stage.setScene(scene);
338                 }
339             });
340
341         //Action for adding equities
342         add.setOnAction(new EventHandler<ActionEvent>() {
343             @Override
344             public void handle(ActionEvent event) {
345                 final Stage dialog = new Stage();
346                 dialog.initOwner(primaryStage);
347                 GridPane grid2 = new GridPane();
348                 grid2.setAlignment(Pos.CENTER);
349                 grid2.setHgap(10);
350                 grid2.setVgap(10);
351                 grid2.setPadding(new Insets(5, 10, 5, 10));
352
353                 VBox dialogVbox = new VBox(20);
354                 dialogVbox.getChildren().add(new Text("This is a Dialog"));
355                 TextField tickerSymbol = new TextField();
356                 grid2.add(new Label("Ticker Symbol: "),0,
357                     1);
358                 grid2.add(tickerSymbol,1,1);
359                 TextField names = new TextField();
360                 grid2.add(new Label("Names: "),0,2);
361                 grid2.add(names,1,2);
362
363                 TextField numShares = new TextField();
364                 grid2.add(new Label("Number of Shares: ")
365                     ,0,3);
366                 grid2.add(numShares,1,3);
367                 TextField salePrice = new TextField();
368                 grid2.add(new Label("Sale Price: "),0,4);
369                 grid2.add(salePrice,1,4);

```

```

367             TextField acquisitionDate = new TextField
368             ();
369             grid2.add(new Label("Aquisition Date (MM/
370 dd/YYYY) : "), 0, 5);
371             grid2.add(acquisitionDate, 1, 5);
372             VBox SIlist = new VBox(2);
373             HBox buttonList = new HBox(30);
374             GridPane sectorIndex = new GridPane();
375             Button addSectorIndex = new Button("add")
376             ;
377             Button removeSectorIndex = new Button(""
378 remove");
379             buttonList.getChildren().addAll(
380             addSectorIndex, removeSectorIndex);
381             sectorIndex.add(buttonList, 0, 0);
382             sectorIndex.add(SIlist, 0, 1);
383             addSectorIndex.setOnAction(new
384             EventHandler<ActionEvent>() {
385                 @Override
386                 public void handle(ActionEvent event)
387                 {
388                     TextField newField = new
389                     TextField();
390                     SIlist.getChildren().add(newField
391                     );
392                     removeSectorIndex.setOnAction(new
393             EventHandler<ActionEvent>() {
394                 @Override
395                 public void handle(ActionEvent event)
396                 {
397                     SIlist.getChildren().remove(0, 1);
398                     });
399                     grid2.add(new Label("Sector/Index: "), 0, 6
400 );
401                     grid2.add(sectorIndex, 1, 6);
402                     Scene dialogScene = new Scene(grid2, 400,
403                     400);
404                     dialog.setScene(dialogScene);
405                     dialog.show();
406                     Button cancel = new Button("Cancel");
407                     grid2.add(cancel, 1, 7);
408                     Button submit = new Button("Submit");

```

```

398                     grid2.add(submit, 0, 7);
399                     cancel.setOnAction(new EventHandler<
400                         ActionEvent>() {
401                             @Override
402                             public void handle(ActionEvent e) {
403                                 dialog.close();
404                             }
405                             final Text error = new Text();
406                             grid2.add(error, 1, 8);
407                             submit.setOnAction(new EventHandler<
408                             ActionEvent>() { // this one
409                                 @Override
410                                 public void handle(ActionEvent event)
411                                 {
412                                     if(tickerSymbol.getText().isEmpty()
413                                         () || numShares.getText().isEmpty() || acquisitionDate.
414                                         getText().isEmpty() ||
415                                         salePrice.getText().
416                                         isEmpty() || sectorIndex.getChildren().isEmpty())
417                                         {
418                                             error.setFill(Color.FIREBRICK
419                                         );
420                                         error.setText("Fill in all
421                                         fields");
422                                         }
423                                         else if(!FPTS.isDouble(salePrice.
424                                         getText()) || !FPTS.toInt(numShares.getText()) || !FPTS.
425                                         isDate(acquisitionDate.getText()))
426                                         {
427                                             error.setFill(Color.FIREBRICK
428                                         );
429                                         error.setText("Enter
430                                         information in correct format");
431                                         }
432                                         else if(Integer.parseInt(
433                                         numShares.getText()) < 1){
434                                             error.setFill(Color.FIREBRICK
435                                         );
436                                         error.setText("Enter valid
437                                         number of shares");
438                                         }
439                                         else{
440                                             ArrayList SIarray = new
441                                         ArrayList();

```

File - C:\Users\hack master\Desktop\school\Semester 4\SWEN-262 (FPTS)\svn\trunk\View\EquitiesView.java

```

427                         SIlisit.getChildren().forEach(
428             (SI) -> {
429                 SIarray.add(((TextField)
430                             SI).getText());
431                 });
432                 Equity e = new Equity(
433                     tickerSymbol.getText(), names.getText(),
434                     Integer.parseInt(numShares.getText()),
435                     Double.parseDouble(salePrice.getText()),
436                     acquisitionDate.
437                     getText(), SIarray.
438                     addEquity(tickerSymbol.
439                     getText(), names.getText(),
440                     Integer.parseInt(numShares.
441                     getText()),
442                     Double.parseDouble(
443                     salePrice.getText()), acquisitionDate.getText(),
444                     SIarray,
445                     false, p);
446
447             UndoAction add = new
448             UndoAction("Add", "Equity", null, null, 0, e, "");
449             add.redo.clear();
450             add.addToQueue(add);
451
452         dialog.close();
453         Scene scene = ViewEquities(
454             primaryStage, p, FPTS);
455         FPTS.stage.setScene(scene);
456
457     }
458
459     }
460
461     }
462
463     }
464
465     }
466
467     }
468
469     }
470
471     }
472
473     }
474
475     }
476
477     }
478
479     }
480
481     }
482
483     }
484
485     }
486
487     }
488
489     }
490
491     }
492
493     }
494
495     }
496
497     }
498
499     }
500
501     }
502
503     }
504
505     }
506
507     }
508
509     }
510
511     }
512
513     }
514
515     }
516
517     }
518
519     }
520
521     }
522
523     }
524
525     }
526
527     }
528
529     }
530
531     }
532
533     }
534
535     }
536
537     }
538
539     }
540
541     }
542
543     }
544
545     }
546
547     }
548
549     }
550
551     }
552
553     }
554
555     }
556
557     }
558
559     }
560
561     }
562
563     }
564
565     }
566
567     }
568
569     }
570
571     }
572
573     }
574
575     }
576
577     }
578
579     }
580
581     }
582
583     }
584
585     }
586
587     }
588
589     }
590
591     }
592
593     }
594
595     }
596
597     }
598
599     }
599
600     }
601
602     }
603
604     }
605
606     }
607
608     }
609
609     }
610
611     }
612
613     }
614
615     }
616
617     }
618
619     }
619
620     }
621
622     }
623
624     }
625
626     }
627
628     }
629
629     }
630
631     }
632
633     }
634
635     }
636
637     }
638
639     }
639
640     }
641
642     }
643
644     }
645
646     }
647
648     }
649
649     }
650
651     }
652
653     }
654
655     }
656
657     }
658
659     }
659
660     }
661
662     }
663
664     }
665
666     }
667
667     }
668
669     }
669
670     }
671
672     }
673
674     }
675
676     }
677
678     }
679
679     }
680
681     }
682
683     }
684
685     }
686
687     }
688
689     }
689
690     }
691
692     }
693
694     }
695
696     }
697
697     }
698
699     }
699
700     }
701
702     }
703
704     }
705
706     }
707
708     }
709
709     }
710
711     }
712
713     }
714
715     }
716
717     }
718
719     }
719
720     }
721
722     }
723
724     }
725
726     }
727
727     }
728
729     }
729
730     }
731
732     }
733
734     }
735
736     }
737
737     }
738
739     }
739
740     }
741
742     }
743
744     }
745
745     }
746
747     }
747
748     }
749
749     }
750
751     }
752
753     }
754
755     }
756
757     }
758
759     }
759
760     }
761
762     }
763
764     }
765
766     }
767
767     }
768
769     }
769
770     }
771
772     }
773
774     }
775
776     }
777
777     }
778
779     }
779
780     }
781
782     }
783
784     }
785
786     }
787
787     }
788
789     }
789
790     }
791
792     }
793
794     }
795
796     }
797
797     }
798
799     }
799
800     }
801
802     }
803
804     }
805
806     }
807
808     }
809
809     }
810
811     }
812
813     }
814
815     }
816
817     }
818
819     }
819
820     }
821
822     }
823
824     }
825
826     }
827
827     }
828
829     }
829
830     }
831
832     }
833
834     }
835
836     }
837
837     }
838
839     }
839
840     }
841
842     }
843
844     }
845
846     }
847
847     }
848
849     }
849
850     }
851
852     }
853
854     }
855
856     }
857
858     }
859
859     }
860
861     }
862
863     }
864
865     }
866
867     }
868
869     }
869
870     }
871
872     }
873
874     }
875
876     }
877
878     }
879
879     }
880
881     }
882
883     }
884
885     }
886
887     }
888
889     }
889
890     }
891
892     }
893
894     }
895
896     }
897
898     }
899
900     }
901
902     }
903
904     }
905
906     }
907
908     }
909
909     }
910
911     }
912
913     }
914
915     }
916
917     }
918
919     }
919
920     }
921
922     }
923
924     }
925
926     }
927
927     }
928
929     }
929
930     }
931
932     }
933
934     }
935
936     }
937
937     }
938
939     }
939
940     }
941
942     }
943
944     }
945
945     }
946
947     }
947
948     }
949
950     }
951
952     }
953
954     }
955
956     }
957
958     }
959
959     }
960
961     }
962
963     }
964
965     }
966
967     }
968
969     }
969
970     }
971
972     }
973
974     }
975
976     }
977
977     }
978
979     }
979
980     }
981
982     }
983
984     }
985
986     }
987
987     }
988
989     }
989
990     }
991
992     }
993
994     }
995
996     }
997
997     }
998
999     }
999
1000    }
1001
1002    }
1003
1004    }
1005
1006    }
1007
1008    }
1009
1009    }
1010
1011    }
1012
1013    }
1014
1015    }
1016
1017    }
1018
1019    }
1019
1020    }
1021
1022    }
1023
1024    }
1025
1026    }
1027
1027    }
1028
1029    }
1029
1030    }
1031
1032    }
1033
1034    }
1035
1036    }
1037
1037    }
1038
1039    }
1039
1040    }
1041
1042    }
1043
1044    }
1045
1045    }
1046
1047    }
1047
1048    }
1049
1050    }
1051
1052    }
1053
1053    }
1054
1055    }
1056
1056    }
1057
1058    }
1059
1059    }
1060
1061    }
1062
1063    }
1064
1064    }
1065
1066    }
1067
1067    }
1068
1069    }
1069
1070    }
1071
1072    }
1073
1074    }
1075
1075    }
1076
1077    }
1078
1078    }
1079
1079    }
1080
1080    }
1081
1081    }
1082
1082    }
1083
1083    }
1084
1084    }
1085
1085    }
1086
1086    }
1087
1087    }
1088
1088    }
1089
1089    }
1090
1090    }
1091
1091    }
1092
1092    }
1093
1093    }
1094
1094    }
1095
1095    }
1096
1096    }
1097
1097    }
1098
1098    }
1099
1099    }
1100
1100    }
1101
1101    }
1102
1102    }
1103
1103    }
1104
1104    }
1105
1105    }
1106
1106    }
1107
1107    }
1108
1108    }
1109
1109    }
1110
1110    }
1111
1111    }
1112
1112    }
1113
1113    }
1114
1114    }
1115
1115    }
1116
1116    }
1117
1117    }
1118
1118    }
1119
1119    }
1120
1120    }
1121
1121    }
1122
1122    }
1123
1123    }
1124
1124    }
1125
1125    }
1126
1126    }
1127
1127    }
1128
1128    }
1129
1129    }
1130
1130    }
1131
1131    }
1132
1132    }
1133
1133    }
1134
1134    }
1135
1135    }
1136
1136    }
1137
1137    }
1138
1138    }
1139
1139    }
1140
1140    }
1141
1141    }
1142
1142    }
1143
1143    }
1144
1144    }
1145
1145    }
1146
1146    }
1147
1147    }
1148
1148    }
1149
1149    }
1150
1150    }
1151
1151    }
1152
1152    }
1153
1153    }
1154
1154    }
1155
1155    }
1156
1156    }
1157
1157    }
1158
1158    }
1159
1159    }
1160
1160    }
1161
1161    }
1162
1162    }
1163
1163    }
1164
1164    }
1165
1165    }
1166
1166    }
1167
1167    }
1168
1168    }
1169
1169    }
1170
1170    }
1171
1171    }
1172
1172    }
1173
1173    }
1174
1174    }
1175
1175    }
1176
1176    }
1177
1177    }
1178
1178    }
1179
1179    }
1180
1180    }
1181
1181    }
1182
1182    }
1183
1183    }
1184
1184    }
1185
1185    }
1186
1186    }
1187
1187    }
1188
1188    }
1189
1189    }
1190
1190    }
1191
1191    }
1192
1192    }
1193
1193    }
1194
1194    }
1195
1195    }
1196
1196    }
1197
1197    }
1198
1198    }
1199
1199    }
1200
1200    }
1201
1201    }
1202
1202    }
1203
1203    }
1204
1204    }
1205
1205    }
1206
1206    }
1207
1207    }
1208
1208    }
1209
1209    }
1210
1210    }
1211
1211    }
1212
1212    }
1213
1213    }
1214
1214    }
1215
1215    }
1216
1216    }
1217
1217    }
1218
1218    }
1219
1219    }
1220
1220    }
1221
1221    }
1222
1222    }
1223
1223    }
1224
1224    }
1225
1225    }
1226
1226    }
1227
1227    }
1228
1228    }
1229
1229    }
1230
1230    }
1231
1231    }
1232
1232    }
1233
1233    }
1234
1234    }
1235
1235    }
1236
1236    }
1237
1237    }
1238
1238    }
1239
1239    }
1240
1240    }
1241
1241    }
1242
1242    }
1243
1243    }
1244
1244    }
1245
1245    }
1246
1246    }
1247
1247    }
1248
1248    }
1249
1249    }
1250
1250    }
1251
1251    }
1252
1252    }
1253
1253    }
1254
1254    }
1255
1255    }
1256
1256    }
1257
1257    }
1258
1258    }
1259
1259    }
1260
1260    }
1261
1261    }
1262
1262    }
1263
1263    }
1264
1264    }
1265
1265    }
1266
1266    }
1267
1267    }
1268
1268    }
1269
1269    }
1270
1270    }
1271
1271    }
1272
1272    }
1273
1273    }
1274
1274    }
1275
1275    }
1276
1276    }
1277
1277    }
1278
1278    }
1279
1279    }
1280
1280    }
1281
1281    }
1282
1282    }
1283
1283    }
1284
1284    }
1285
1285    }
1286
1286    }
1287
1287    }
1288
1288    }
1289
1289    }
1290
1290    }
1291
1291    }
1292
1292    }
1293
1293    }
1294
1294    }
1295
1295    }
1296
1296    }
1297
1297    }
1298
1298    }
1299
1299    }
1300
1300    }
1301
1301    }
1302
1302    }
1303
1303    }
1304
1304    }
1305
1305    }
1306
1306    }
1307
1307    }
1308
1308    }
1309
1309    }
1310
1310    }
1311
1311    }
1312
1312    }
1313
1313    }
1314
1314    }
1315
1315    }
1316
1316    }
1317
1317    }
1318
1318    }
1319
1319    }
1320
1320    }
1321
1321    }
1322
1322    }
1323
1323    }
1324
1324    }
1325
1325    }
1326
1326    }
1327
1327    }
1328
1328    }
1329
1329    }
1330
1330    }
1331
1331    }
1332
1332    }
1333
1333    }
1334
1334    }
1335
1335    }
1336
1336    }
1337
1337    }
1338
1338    }
1339
1339    }
1340
1340    }
1341
1341    }
1342
1342    }
1343
1343    }
1344
1344    }
1345
1345    }
1346
1346    }
1347
1347    }
1348
1348    }
1349
1349    }
1350
1350    }
1351
1351    }
1352
1352    }
1353
1353    }
1354
1354    }
1355
1355    }
1356
1356    }
1357
1357    }
1358
1358    }
1359
1359    }
1360
1360    }
1361
1361    }
1362
1362    }
1363
1363    }
1364
1364    }
1365
1365    }
1366
1366    }
1367
1367    }
1368
1368    }
1369
1369    }
1370
1370    }
1371
1371    }
1372
1372    }
1373
1373    }
1374
1374    }
1375
1375    }
1376
1376    }
1377
1377    }
1378
1378    }
1379
1379    }
1380
1380    }
1381
1381    }
1382
1382    }
1383
1383    }
1384
1384    }
1385
1385    }
1386
1386    }
1387
1387    }
1388
1388    }
1389
1389    }
1390
1390    }
1391
1391    }
1392
1392    }
1393
1393    }
1394
1394    }
1395
1395    }
1396
1396    }
1397
1397    }
1398
1398    }
1399
1399    }
1400
1400    }
1401
1401    }
1402
1402    }
1403
1403    }
1404
1404    }
1405
1405    }
1406
1406    }
1407
1407    }
1408
1408    }
1409
1409    }
1410
1410    }
1411
1411    }
1412
1412    }
1413
1413    }
1414
1414    }
1415
1415    }
1416
1416    }
1417
1417    }
1418
1418    }
1419
1419    }
1420
1420    }
1421
1421    }
1422
1422    }
1423
1423    }
1424
1424    }
1425
1425    }
1426
1426    }
1427
1427    }
1428
1428    }
1429
1429    }
1430
1430    }
1431
1431    }
1432
1432    }
1433
1433    }
1434
1434    }
1435
1435    }
1436
1436    }
1437
1437    }
1438
1438    }
1439
1439    }
1440
1440    }
1441
1441    }
1442
1442    }
1443
1443    }
1444
1444    }
1445
1445    }
1446
1446    }
1447
1447    }
1448
1448    }
1449
1449    }
1450
1450    }
1451
1451    }
1452
1452    }
1453
1453    }
1454
1454    }
1455
1455    }
1456
1456    }
1457
1457    }
1458
1458    }
1459
1459    }
1460
1460    }
1461
1461    }
1462
1462    }
1463
1463    }
1464
1464    }
1465
1465    }
1466
1466    }
1467
1467    }
1468
1468    }
1469
1469    }
1470
1470    }
1471
1471    }
1472
1472    }
1473
1473    }
1474
1474    }
1475
1475    }
1476
1476    }
1477
1477    }
1478
1478    }
1479
1479    }
1480
1480    }
1481
1481    }
1482
1482    }
1483
1483    }
1484
1484    }
1485
1485    }
1486
1486    }
1487
1487    }
1488
1488    }
1489
1489    }
1490
1490    }
1491
1491    }
1492
1492    }
1493
1493    }
1494
1494    }
1495
1495    }
1496
1496    }
1497
1497    }
1498
1498    }
1499
1499    }
1500
1500    }
1501
1501    }
1502
1502    }
1503
1503    }
1504
1504    }
1505
1505    }
1506
1506    }
1507
1507    }
1508
1508    }
1509
1509    }
1510
1510    }
1511
1511    }
1512
1512    }
1513
1513    }
1514
1514    }
1515
1515    }
1516
1516    }
1517
1517    }
1518
1518    }
1519
1519    }
1520
1520    }
1521
1521    }
1522
1522    }
1523
1523    }
1524
1524    }
1525
1525    }
1526
1526    }
1527
1527    }
1528
1528    }
1529
1529    }
1530
1530    }
1531
1531    }
1532
1532    }
1533
1533    }
1534
1534    }
1535
1535    }
1536
1536    }
1537
1537    }
1538
1538    }
1539
1539    }
1540
1540    }
1541
1541    }
1542
1542    }
1543
1543    }
1544
1544    }
1545
1545    }
1546
1546    }
1547
1547    }
1548
1548    }
1549
1549    }
1550
1550    }
1551
1551    }
1552
1552    }
1553
1553    }
1554
1554    }
1555
1555    }
1556
1556    }
1557
1557    }
1558
1558    }
1559
1559    }
1560
1560    }
1561
1561    }
1562
1562    }
1563
1563    }
1564
1564    }
1565
1565    }
1566
1566    }
1567
1567    }
1568
1568    }
1569
1569    }
1570
1570    }
1571
1571    }
1572
1572    }
1573
1573    }
1574
1574    }
1575
1575    }
1576
1576    }
1577
1577    }
1578
1578    }
1579
1579    }
1580
1580    }
1581
1581    }
1582
1582    }
1583
1583    }
1584
1584    }
1585
1585    }
1586
1586    }
1587
1587    }
1588
1588    }
1589
1589    }
1590
1590    }
1591
1591    }
1592
1592    }
1593
1593    }
1594
1594    }
1595
1595    }
1596
1596    }
1597
1597    }
1598
1598    }
1599
1599    }
1600
1600    }
1601
1601    }
1602
1602    }
1603
1603    }
1604
1604    }
1605
1605    }
1606
1606    }
1607
1607    }
1608
1608    }
1609
1609    }
1610
1610    }
1611
1611    }
1612
1612    }
1613
1613    }
1614
1614    }
1615
1615    }
1616
1616    }
1617
1617    }
1618
1618    }
1619
1619    }
1620
1620    }
1621
1621    }
1622
1622    }
1623
1623    }
1624
1624    }
1625
1625    }
1626
1626    }
1627
1627    }
1628
1628    }
1629
1629    }
1630
1630    }
1631
1631    }
1632
1632    }
1633
1633    }
1634
1634    }
1635
1635    }
1636
1636    }
1637
1637    }
1638
1638    }
1639
1639    }
1640
1640    }
1641
1641    }
1642
1642    }
1643
1643    }
1644
1644    }
1645
1645    }
1646
1646    }
1647
1647    }
1648
1648    }
1649
1649    }
1650
1650    }
1651
1651    }
1652
1652    }
1653
1653    }
1654
1654    }
1655
1655    }
1656
1656    }
1657
1657    }
1658
1658    }
1659
1659    }
1660
1660    }
1661
1661    }
1662
1662    }
1663
1663    }
1664
1664    }
1665
1665    }
1666
1666    }
1667
1667    }
1668
1668    }
1669
1669    }
1670
1670    }
1671
1671    }
1672
1672    }
1673
1673    }
1674
1674    }
1675
1675    }
1676
1676    }
1677
1677    }
1678
1678    }
1679
1679    }
1680
1680    }
1681
1681    }
1682
1682    }
1683
1683    }
1684
1684    }
1685
1685    }
1686
1686    }
1687
1687    }
1688
1688    }
1689
1689    }
1690
1690    }
1691
1691    }
1692
1692    }
1693
1693    }
1694
1694    }
1695
1695    }
1696
1696    }
1697
1697    }
1698
1698    }
1699
1699    }
1700
1700    }
1701
1701    }
1702
1702    }
1703
1703    }
1704
1704    }
1705
1705    }
1706
1706    }
1707
1707    }
1708
1708    }
1709
1709    }
1710
1710    }
1711
1711    }
1712
1712    }
1713
1713    }
1714
1714    }
1715
1715    }
1716
1716    }
1717
1717    }
1718
1718    }
1719
1719    }
1720
1720    }
1721
1721    }
1722
1722    }
1723
1723    }
1724
1724    }
1725
1725    }
1726
1726    }
1727
1727    }
1728
1728    }
1729
1729    }
1730
1730    }
1731
1731    }
1732
1732    }
1733
1733    }
1734
1734    }
1735
1735    }
1736
1736    }
1737
1737    }
1738
1738    }
1739
1739    }
1740
1740    }
1741
1741    }
1742
1742    }
1743
1743    }
1744
1744    }
1745
1745    }
1746
1746    }
1747
1747    }
1748
1748    }
1749
1749    }
1750
1750    }
1751
1751    }
1752
1752    }
1753
1753    }
1754
1754    }
1755
1755    }
1756
1756    }
1757
1757    }
1758
1758    }
1759
1759    }
1760
1760    }
1761
1761    }
1762
1762    }
1763
1763    }
1764
1764    }
1765
1765    }
1766
1766    }
1767
1767    }
1768
1768    }
1769
1769    }
1770
1770    }
1771
1771    }
1772
1772    }
1773
1773    }
1774
1774    }
1775
1775    }
1776
1776    }
1777
1777    }
1778
1778    }
1779
1779    }
1780
1780    }
1781
1781    }
1782
1782    }
1783
1783    }
1784
1784    }
1785
1785    }
1786
1786    }
1787
1787    }
1788
1788    }
1789
1789    }
1790
1790    }
1791
1791    }
1792
1792    }
1793
1793    }
1794
1794    }
1795
1795    }
1796
1796    }
1797
1797    }
1798
1798    }
1799
1799    }
1800
1800    }
1801
1801    }
1802
1802    }
1803
1803    }
1804
1804    }
1805
1805    }
1806
1806    }
1807
1807    }
1808
1808    }
1809
1809    }
1810
1810    }
1811
1811    }
1812
1812    }
1813
1813    }
1814
1814    }
1815
1815    }
1816
1816    }
1817
1817    }
1818
1818    }
1819
1819    }
1820
1820    }
1821
1821    }
1822
1822    }
1823
1823    }
1824
1824    }
1825
1825    }
1826
1826    }
1827
1827    }
1828
1828    }
1829
1829    }
1830
1830    }
1831
1831    }
1832
1832    }
1833
1833    }
1834
1834    }
1835
1835    }
1836
1836    }
1837
1837    }
1838
1838    }
1839
1839    }
1840
1840    }
1841
1841    }
1842
1842    }
1843
1843    }
1844
1844    }
1845
1845    }
1846
1846    }
1847
1847    }
1848
1848    }
1849
1849    }
1850
1850    }
1851
1851    }
1852
1852    }
1853
1853    }
1854
1854    }
1855
1855    }
1856
1856    }
1857
1857    }
1858
1858    }
1859
1859    }
1860
1860    }
1861
1861    }
1862
1862    }
1863
1863    }
1864
1864    }
1865
1865    }
1866
1866    }
1867
1867    }
1868
1868    }
1869
1869    }
1870
1870    }
1871
1871    }
1872
1872    }
1873
1873    }
1874
1874    }
1875
1875    }
1876
1876    }
1877
1877    }
1878
1878    }
1879
1879    }
1880
1880    }
1881
1881    }
1882
1882    }
188
```

```

458             ArrayList<String> cashAccts = new
459             ArrayList<>();
460             for(Equity e : p.equityList1) {
461                 if(!e.tickerSymbol.equals("")) {
462                     choices.add(e.tickerSymbol);
463                 }
464             for(CashAccount c : p.cashAccList) {
465                 cashAccts.add(c.name);
466             }
467
468             ChoiceDialog<String> dialog = new
469             ChoiceDialog<>("None", choices);
470             dialog.setContentText("Choose the equity
471             you wish to remove:");
472
473             Optional<String> result = dialog.
474             showAndWait();
475             if (result.isPresent()) {
476                 ChoiceDialog<String> addCash = new
477                 ChoiceDialog<>("None", cashAccts);
478                 addCash.setContentText("Choose cash
479                 account to add equity value to:");
480
481                 Optional<String> cashAcct = addCash.
482                 showAndWait();
483
484                 Equity e = new Equity("", "", 0, 0, "",

485                 null);
486                 for (Equity n : p.equityList1) {
487                     if (n.getTickerSymbol().equals(
488                         result.get())) {
489                         e = n;
490                     }
491                 }
492
493
494                 UndoAction add = new UndoAction("

495                 Remove", "Equity", null, null, 0, e, "");
496                 add.redo.clear();
497                 add.addToQueue(add);
498                 e.removeEquity(result.get(), cashAcct
499                 .get(), p);
500             }
501
502             Scene scene = ViewEquities(primaryStage, p
503             , FPTS);

```

```

491                     FPTS.stage.setScene(scene);
492
493                 }
494             });
495
496
497             toolBar2.getItems().addAll(spacer, add, remove,
498             importEq, exportEq, addShares, sellShares,addSharesToMA,
499             mas);
500
501             addShares.setOnAction(new EventHandler<
502             ActionEvent>() {
503                 @Override
504                 public void handle(ActionEvent event) {
505                     ArrayList<String> choices = new ArrayList
506                     <>();
507
508                     for(Equity e : p.equityList2) {
509                         if(!e.tickerSymbol.equals("")) {
510                             choices.add(e.tickerSymbol);
511                         } else{
512                             choices.add(e.marketAverage.get(0
513                         ));
514                         }
515
516                     ChoiceDialog<String> dialog = new
517                     ChoiceDialog<>("None", choices);
518                     dialog.setContentText("Choose the equity
519                     you wish to add shares to:");
520
521
522                     Optional<String> result = dialog.
523                     showAndWait();
524
525                     if (result.isPresent()) {
526                         final Stage dialog2 = new Stage();
527                         dialog2.initOwner(primaryStage);
528                         GridPane grid2 = new GridPane();
529                         grid2.setAlignment(Pos.CENTER);
530                         grid2.setHgap(10);
531                         grid2.setVgap(10);
532                         grid2.setPadding(new Insets(5, 10, 5,
533                         10));
534
535                         VBox dialogVbox = new VBox(20);
536                         dialogVbox.getChildren().add(new Text

```

```

526 ("This is a Dialog"));
527             TextField shares = new TextField();
528
529             grid2.add(new Label("Add Shares to "
530 + result.get() + ":"),0,1);
530             grid2.add(shares,1,1);
531             Scene dialogScene = new Scene(grid2,
532 400, 400);
533             Button cancel = new Button("Cancel");
534             grid2.add(cancel,1,3);
535             Button submit = new Button("Submit");
536             grid2.add(submit,0,3);
537             cancel.setOnAction(new EventHandler<
538 ActionEvent>() {
539             @Override
540             public void handle(ActionEvent e)
541             {dialog2.close(); }
542             });
543             final Text error = new Text();
544             grid2.add(error, 1, 8);
545             submit.setOnAction(new EventHandler<
546 ActionEvent>() { // this one
547             @Override
548             public void handle(ActionEvent
549 event) {
550                 if(shares.getText().isEmpty())
551                 {
552                     error.setFill(Color.
553 FIREBRICK);
554                     error.setText("Fill in
555 all fields");
556                 }
557                 else if(!FPTS.toInt(shares.
558 getText()))
559                 {
560                     error.setFill(Color.
561 FIREBRICK);
562                     error.setText("Enter
563 information in correct format");
564                 }
565                 else if(Integer.parseInt(
566 shares.getText()) < 1)
567                 {
568                     error.setFill(Color.
569 FIREBRICK);
570                     error.setText("Enter
571 a positive integer");
572                 }
573             }
574         });

```

```

556 valid number of shares");
557 }
558 else {
559     int numShares = Integer.
560     parseInt(shares.getText());
561     double tot = 0;
562     for(Equity stuff : p.
563         equityList2) {
564         if (stuff.
565             getTickerSymbol().equals(result.get())) {
566             tot = numShares *
567                 stuff.acquisitionPrice;
568             ArrayList<String>
569             cashAccts = new ArrayList<>();
570             for(CashAccount c
571                 : p.cashAccList) {
572                 cashAccts.add
573                     (c.name);
574             }
575             ChoiceDialog<
576             String> addCash = new ChoiceDialog<>("None", cashAccts);
577             addCash.
578             setContentText("Choose cash account to pay for shares:");
579             Optional<String>
580             cashAcct = addCash.showAndWait();
581             CashAccount c =
582             new CashAccount("", "", 0, "", null);
583             if(cashAcct.
584                 isPresent()) {
585                 for(
586                     CashAccount acct : p.cashAccList) {
587                         if(acct.
588                             name.equals(cashAcct.get())) {
589                             if(
590                                 acct.balance >= tot) {
591                                     c
592                                         .updateBalance(cashAcct.get(), tot, false, p);
593                                     Equity e = new Equity("", "", 0, 0, "", null);
594                                     DateTimeStringConverter format = new
595                                         DateTimeStringConverter("MM/dd/YYYY");
596                                     Date date = new Date();

```

```

581     stuff.acquisitionDate = format.toString(date);
582     e
583     .addEquity(stuff.tickerSymbol,stuff.name,numShares,stuff.
584     acquisitionPrice,
585
586     stuff.acquisitionDate,stuff.getMarketAverage(),
587     false,p);
588     addCash.close();
589     dialog2.close();
590
591     Scene scene = ViewEquities(primaryStage,p, FPTS);
592     FPTS.stage.setScene(scene);
593
594
595     }
596
597     else{
598         error.setFill(Color.FIREBRICK);
599
600         error.setText("Account contains too little funds.");
601
602         }
603
604         addCash.close();
605         dialog2.close();
606         Scene scene =
607         ViewEquities(primaryStage,p, FPTS);
608         FPTS.stage.
609         setScene(scene);
610
611         }
612         else if(stuff.
613         tickerSymbol.equals("")&& stuff.
614

```

```

607 marketAverage.get(0).equals(result.get())) {
608     cashAccts = new ArrayList<>();
609     for (CashAccount c : p.cashAccList) {
610         cashAccts.add(c.name);
611     }
612     ChoiceDialog<
613     String> addCash = new ChoiceDialog<>("None", cashAccts);
614     addCash.setContentText("Choose cash account to pay for shares:");
615     Optional<String> cashAcct = addCash.showAndWait();
616     CashAccount c =
617     new CashAccount("", "", 0, "", null);
618     tot = numShares *
619     stuff.acquisitionPrice;
620     if (cashAcct.isPresent()) {
621         for (CashAccount acct : p.cashAccList) {
622             if (acct.name.equals(cashAcct.get())) {
623                 if (acct.balance >= tot) {
624                     c.updateBalance(cashAcct.get(), tot, false, p);
625                     Equity e = new Equity("", "", 0, 0, "", null);
626                     e.addEquity(stuff.tickerSymbol, stuff.name, numShares, stuff.
627                     acquisitionPrice,
628
629                     stuff.acquisitionDate, stuff.getMarketAverage(),
630                     false, p);
631                     addCash.close();
632                     dialog2.close();
633                     Scene scene = ViewEquities(primaryStage, p, FPTS);
634                     FPTS.stage.setScene(scene);

```

```

629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663

```

}

else{

 error.setFill(Color.FIREBRICK);

 error.setText("Account contains too little funds.");

}

}

}

}

else{

 error.setFill

 (error.setFill

 (Color.FIREBRICK));

 error.setText

 ("Select a cash account");

}

addCash.close();

dialog2.close();

Scene scene =

 ViewEquities(primaryStage,p, FPTS);

FPTS.stage.

 setScene(scene);

}

}

dialog2.close();

Scene scene =

 ViewEquities(primaryStage,p, FPTS);

FPTS.stage.setScene(scene)

);

}

}

) ;

dialog2.setScene(dialogScene);

dialog2.show();

}

}

});

sellShares.setOnAction(**new** EventHandler<

ActionEvent>() {

 @Override

public void handle(ActionEvent event) {

 ArrayList<String> choices = **new** ArrayList

```

663 <> () ;
664             for(Equity e : p.equityList1) {
665                 if (!e.tickerSymbol.equals("")) {
666                     choices.add(e.tickerSymbol);
667                 } else{
668                     choices.add(e.marketAverage.get(0
669                 )
670             }
671             ChoiceDialog<String> dialog = new
672             ChoiceDialog<>("None", choices);
673             dialog.setContentText("Choose the equity
674             you wish to sell shares from:");
675
676             Optional<String> result = dialog.
677             showAndWait();
678             if (result.isPresent()) {
679                 int num1 = 0;
680                 for (Equity stuff : p.equityList1) {
681                     if (stuff.getTickerSymbol() .
682                         equals(result.get())) {
683                         num1 = stuff.shares;
684                     }
685                 }
686             }
687             final int num = num1;
688             final Stage dialog2 = new Stage();
689             dialog2.initOwner(primaryStage);
690             GridPane grid2 = new GridPane();
691             grid2.setAlignment(Pos.CENTER);
692             grid2.setHgap(10);
693             grid2.setVgap(10);
694             grid2.setPadding(new Insets(5, 10, 5,
695             10));
696
697             VBox dialogVbox = new VBox(20);
698             dialogVbox.getChildren().add(new Text
("This is a Dialog"));
699             TextField shares = new TextField();

```

```

699
700                     grid2.add(new Label("Number of Shares
    to Sell: "), 0, 1);
701                     grid2.add(shares, 1, 1);
702                     Scene dialogScene = new Scene(grid2,
    400, 400);
703                     Button cancel = new Button("Cancel");
704                     grid2.add(cancel, 1, 3);
705                     Button submit = new Button("Submit");
706                     grid2.add(submit, 0, 3);
707                     cancel.setOnAction(new EventHandler<
    ActionEvent>() {
708                         @Override
709                         public void handle(ActionEvent e)
    {
710                             dialog2.close();
711                         }
712                     );
713                     final Text error = new Text();
714                     grid2.add(error, 1, 8);
715                     submit.setOnAction(new EventHandler<
    ActionEvent>() { // this one
716                         @Override
717                         public void handle(ActionEvent
    event) {
718                             if (shares.getText().isEmpty(
    )) {
719                                 error.setFill(Color.
    FIREBRICK);
720                                 error.setText("Fill in
    all fields");
721                             } else if (!FPTS.toInt(shares
    .getText())) {
722                                 error.setFill(Color.
    FIREBRICK);
723                                 error.setText("Enter
    information in correct format");
724                             }
725                             int numShares = Integer.
    parseInt(shares.getText());
726                             if (numShares > num ||
    numShares < 0) {
727                                 error.setFill(Color.
    FIREBRICK);
728                                 error.setText("Invalid
    number of shares");
729                             }
730                         }
731                     });
732                     dialog.setScene(dialogScene);
733                     dialog.show();
734                 }
735             );
736         }
737     );
738 
```

```

728 value") ;
729
730     }
731
732     else {
733         Equity e = null;
734         double tot = 0;
735         double amount = 0;
736         for (Equity stuff : p.
737             equityList1) {
738             if (stuff.
739                 getTickerSymbol().equals(result.get())) {
740                 tot = numShares *
741                     stuff.acquisitionPrice;
742                 stuff.shares -=
743                 numShares;
744                 amount = -(

745                 numShares * stuff.acquisitionPrice);
746                 e = stuff;
747             }
748         }
749
750         ArrayList<String>
751         cashAccts = new ArrayList<>();
752         for (CashAccount c : p.
753             cashAccList) {
754             cashAccts.add(c.name)
755             ;
756         }
757         ChoiceDialog<String>
758         addCash = new ChoiceDialog<>("None", cashAccts);
759         addCash.setContentText("

Choose cash account to add shares to:");

760         Optional<String> cashAcct
761         = addCash.showAndWait();

```

```

757                                     CashAccount c = new
758                                     CashAccount ("", "", 0, "", null);
759                                     if (!cashAcct.equals ("None
760                                     "))
761                                     {
762                                     for (CashAccount acct
763                                     : p.cashAccList) {
764                                     if (acct.name.
765                                     equals (cashAcct.get ()) ) {
766                                     if (acct.
767                                     balance >= tot) {
768                                     c.
769                                     updateBalance (cashAcct.get (), tot, true, p);
770                                     dialog2.
771                                     close ();
772                                     Scene
773                                     scene = ViewEquities (primaryStage, p, FPTS);
774                                     FPTS.
775                                     stage.setScene (scene);
776                                     } else{
777                                     error.
778                                     setFill (Color.FIREBRICK);
779                                     error.
780                                     setText ("Account contains too little funds.");
781                                     }
782                                     }
783                                     }
784                                     DateFormat dateFormat =
785                                     new SimpleDateFormat ("yyyy/MM/dd HH:mm:ss");
786                                     Date date = new Date ();
787
788                                     HistoryInvoker test = new
789                                     HistoryInvoker ();
790                                     test.setLog (new
791                                     LogEquityHistory (), p, new History ("Sell Shares", "Equity
792                                     ", e, dateFormat.format (date), amount, -numShares));
793                                     test.logAction ();
794
795                                     dialog2.close ();
796                                     Scene scene =
797                                     ViewEquities (primaryStage, p, FPTS);
798                                     FPTS.stage.setScene (scene
799                                     );

```

```

785
786
787
788
789
790    });
791
792    dialog2.setScene(dialogScene);
793    dialog2.show();
794 }
795 }
796 });
797 addSharesToMA.setOnAction(new EventHandler<
ActionEvent>() {
798     @Override
799     public void handle(ActionEvent event) {
800         ArrayList<String> choices = new ArrayList
<>();
801         for(Equity e : p.marketAverages) {
802             if(!e.tickerSymbol.equals("")) {
803                 choices.add(e.tickerSymbol);
804             } else{
805                 choices.add(e.marketAverage.get(0
));
806             }
807         }
808         ChoiceDialog<String> dialog = new
ChoiceDialog<>("None", choices);
809         dialog.setContentText("Choose the market
average you wish to add shares to:");
810
811         Optional<String> result = dialog.
showAndWait();
812         if (result.isPresent()) {
813             final Stage dialog2 = new Stage();
814             dialog2.initOwner(primaryStage);
815             GridPane grid2 = new GridPane();
816             grid2.setAlignment(Pos.CENTER);
817             grid2.setHgap(10);
818             grid2.setVgap(10);
819             grid2.setPadding(new Insets(5, 10, 5,
10));
820
821             VBox dialogVbox = new VBox(20);
822             dialogVbox.getChildren().add(new Text

```

```

822 ("This is a Dialog"));
823             TextField shares = new TextField();
824
825             grid2.add(new Label("Add Shares to "
826 + result.get() + ":"),0,1);
827             grid2.add(shares,1,1);
828             Scene dialogScene = new Scene(grid2,
829 400, 400);
830             Button cancel = new Button("Cancel");
831             grid2.add(cancel,1,3);
832             Button submit = new Button("Submit");
833             grid2.add(submit,0,3);
834             cancel.setOnAction(new EventHandler<
835 ActionEvent>() {
836             @Override
837             public void handle(ActionEvent e)
838             {dialog2.close(); }
839             });
840             final Text error = new Text();
841             grid2.add(error, 1, 8);
842             submit.setOnAction(new EventHandler<
843 ActionEvent>() { // this one
844             @Override
845             public void handle(ActionEvent
846 event) {
847                 if(shares.getText().isEmpty())
848                 {
849                     error.setFill(Color.
850 FIREBRICK);
851                     error.setText("Fill in
852 all fields");
853                 }
854                 else if(!FPTS.toInt(shares.
855 getText()))
856                 {
857                     error.setFill(Color.
858 FIREBRICK);
859                     error.setText("Enter
860 information in correct format");
861                 }
862                 else if(Integer.parseInt(
863 shares.getText()) < 1)
864                 {
865                     error.setFill(Color.
866 FIREBRICK);
867                     error.setText("Enter
868 a positive integer");
869                 }
870             }
871         });

```

```

852 valid number of shares");
853 }
854 else{
855     int numShares = Integer.
856     parseInt(shares.getText());
857     for(Equity e : p.
858         marketAverages) {
859             if(e.marketAverage.
860                 contains(result.get())) {
861                 e.shares +=
862                 numShares;
863             }
864         }
865     dialog2.close();
866     Scene scene =
867     ViewEquities(primaryStage,p, FPTS);
868     FPTS.stage.setScene(scene
869 );
870     }
871     }
872 }
873 }
874
875 TextField tickerFilter = new TextField();
876 tickerFilter.setPromptText("Enter ticker symbol")
877 ;
878
879 TextField nameFilter = new TextField();
880 nameFilter.setPromptText("Enter name");
881
882 TextField maFilter = new TextField();
883 maFilter.setPromptText("Enter market average");
884
885 FilteredList<Equity> filteredData = new
886 FilteredList<>(equities1);
887 FilteredList<Equity> filteredData2 = new
888 FilteredList<>(equities2);
889
890 tickerFilter.textProperty().addListener((

```

```
887 observable, oldValue, newValue) -> {
888         filter(filteredData,filteredData2,
889     tickerFilter,nameFilter,maFilter,newValue);
890     });
891
892     nameFilter.textProperty().addListener((observable
893 , oldValue, newValue) -> {
894         filter(filteredData,filteredData2,
895     tickerFilter,nameFilter,maFilter,newValue);
896     });
897
898
899
900     // Wrap the FilteredList in a SortedList.
901     FilteredList<Equity> sortedData = new
902     FilteredList<Equity>(filteredData);
903     FilteredList<Equity> sortedData2 = new
904     FilteredList<Equity>(filteredData2);
905
906     // Add sorted (and filtered) data to the table.
907     tableOwnedE.setItems(sortedData);
908     tableAvailableE.setItems(sortedData2);
909
910     VBox bottom = new VBox();
911     HBox searchBoxes = new HBox();
912     searchBoxes.getChildren().addAll(tickerFilter,
913     nameFilter,maFilter);
914
915     bottom.getChildren().addAll(Search,searchBoxes);
916     border.setCenter(vbox);
917     border.setBottom(bottom);
918     Scene scene = new Scene(border, 1200, 750);
919     return scene;
920 }
```

```
1 package trunk.View;
2
3 import javafx.collections.FXCollections;
4 import javafx.collections.ObservableList;
5 import javafx.event.ActionEvent;
6 import javafx.event.EventHandler;
7 import javafx.geometry.Insets;
8 import javafx.geometry.Orientation;
9 import javafx.geometry.Pos;
10 import javafx.scene.Scene;
11 import javafx.scene.control.*;
12 import javafx.scene.control.cell.PropertyValueFactory;
13 import javafx.scene.layout.BorderPane;
14 import javafx.scene.layout.GridPane;
15 import javafx.scene.layout.VBox;
16 import javafx.scene.paint.Color;
17 import javafx.scene.text.Font;
18 import javafx.scene.text.Text;
19 import javafx.stage.Stage;
20 import trunk.Model.Equity;
21 import trunk.Model.Portfolio;
22 import trunk.Model.Watchlist;
23
24 import java.io.*;
25 import java.util.ArrayList;
26 import java.util.Optional;
27
28 /**
29  * Created by Jeff Kotowicz on 4/5/2016.
30  *
31  * Provides a watchlist for equities and Market Averages
32  */
33 public class WatchlistView {
34     /*
35      * Code for saving watchlist
36     */
37     public void saveWatchlist(Portfolio p) {
38         ArrayList<Portfolio> port = null;
39         try {
40             FileInputStream fileIn = new FileInputStream("myfile");
41             ObjectInputStream in = new ObjectInputStream(
42                 fileIn);
43             port = (ArrayList<Portfolio>) in.readObject();
44             in.close();
45         }
46     }
47 }
```

```

44         fileIn.close();
45     } catch (ClassNotFoundException ee) {
46         ee.printStackTrace();
47     } catch (FileNotFoundException ee) {
48         ee.printStackTrace();
49     } catch (IOException ee) {
50         ee.printStackTrace();
51     }
52     port.forEach(iter -> {
53         iter.watchlists = p.watchlists;
54     });
55
56     try
57     {
58         FileOutputStream fileOut =
59             new FileOutputStream("myfile");
60         ObjectOutputStream out = new
61             ObjectOutputStream(fileOut);
62         out.writeObject(port);
63         out.close();
64         fileOut.close();
65     } catch(IOException i)
66     {
67         i.printStackTrace();
68     }
69
70     /*
71      * GUI for Watchlist page
72     */
73     public Scene ViewWatchlist(Stage primaryStage,
74         Portfolio p, FPTS FPTS) {
75         Text title = new Text(p.loginID + "'s Watchlist");
76         BorderPane border = FPTS.borderPane(primaryStage,
77             p, title);
78         VBox left = new VBox(5);
79
80         ToolBar toolBar2 = new ToolBar();
81         toolBar2.setOrientation(Orientation.VERTICAL);
82         left.getChildren().addAll(toolBar2);
83         left.setSpacing(10);
84
85         final Label watchItems = new Label(p.loginID + "'s
86             Watchlist");
87         watchItems.setFont(new Font("Arial", 20));

```

```

85         TableView watchList = new TableView();
86         watchList.setEditable(true);
87         final ObservableList<Watchlist> watch =
88             FXCollections.observableArrayList(p.watchlists);
89
90         TableColumn symbols = new TableColumn("Ticker
91             Symbol");
92         TableColumn lowtrigger = new TableColumn("Low
93             Trigger");
94         TableColumn higtrigger = new TableColumn("High
95             Trigger");
96         TableColumn price = new TableColumn("Per Share
97             Price");
98         TableColumn compare = new TableColumn("Compare");
99
100        TableColumn lowtriggered = new TableColumn("Low
101            Trigger");
102        TableColumn higtriggered = new TableColumn("High
103            Trigger");
104
105        Label spacer = new Label(" ");
106
107        final VBox vbox = new VBox();
108        vbox.setSpacing(5);
109        vbox.setPadding(new Insets(10, 0, 0, 10));
110        vbox.getChildren().addAll(watchItems, watchList);
111        border.setLeft(left);
112
113        symbols.setCellValueFactory(
114            new PropertyValueFactory<Watchlist, String
115            >("tickerSymbol")
116            );
117        lowtrigger.setCellValueFactory(
118            new PropertyValueFactory<Watchlist, Double
119            >("lowtrigger")
120            );
121        higtrigger.setCellValueFactory(
122            new PropertyValueFactory<Watchlist, Double
123            >("higtrigger")
124            );
125        price.setCellValueFactory(
126            new PropertyValueFactory<Watchlist,
127            Double>("pershareprice")
128            );

```

```

119         compare.setCellValueFactory(
120             new PropertyValueFactory<Watchlist,
121             String>("compare")
122         );
123         lowtriggered.setCellValueFactory(
124             new PropertyValueFactory<Watchlist,
125             String>("lowtriggered")
126         );
127         hightriggered.setCellValueFactory(
128             new PropertyValueFactory<Watchlist,
129             String>("hightriggered")
130         );
131
132         //watchList.setItems(equities1);
133         watchList.getColumns().addAll(symbols,
134             hightigger, lowtrigger, price, compare, highttriggered,
135             lowtriggered);
136         watchList.setItems(watch);
137
138         symbols.prefWidthProperty().bind(watchList.
139             widthProperty().multiply(0.1));
140         lowtrigger.prefWidthProperty().bind(watchList.
141             widthProperty().multiply(0.075));
142         hightigger.prefWidthProperty().bind(watchList.
143             widthProperty().multiply(0.15));
144
145         Button add = new Button("Add Item");
146         Button remove = new Button("Remove Item");
147         Button high = new Button("Add High Trigger");
148         Button low = new Button("Add Low Trigger");
149         Button reset = new Button("Reset");
150
151         //Action for adding equities
152         add.setOnAction(new EventHandler<ActionEvent>() {
153             @Override
154             public void handle(ActionEvent event) {
155                 ArrayList<String> choices = new ArrayList<
156                 <>();
157                 double price1 = 0;
158                 for(Equity e : p.equityList2) {

```

```

155                     if(!e.tickerSymbol.equals("")) {
156                         choices.add(e.tickerSymbol);
157                     } else{
158                         choices.add(e.marketAverage.get(0
159                         ));
160                     }
161                     ChoiceDialog<String> dialog = new
162                     ChoiceDialog<>("None", choices);
163                     dialog.setContentText("Choose the equity
164                     you wish to add to the watchlist:");
165                     Optional<String> result = dialog.
166                     showAndWait();
167                     if (result.isPresent()) {
168                         for(Equity e : p.equityList2){
169                             if(e.tickerSymbol.equals(result.
170                             get())) {
171                                 pricel = e.acquisitionPrice;
172                             }
173                             }
174                             final double price = pricel;
175                             final Stage dialog2 = new Stage();
176                             dialog2.initOwner(primaryStage);
177                             GridPane grid2 = new GridPane();
178                             grid2.setAlignment(Pos.CENTER);
179                             grid2.setHgap(10);
180                             grid2.setVgap(10);
181                             grid2.setPadding(new Insets(5, 10, 5,
182                             10));
183
184                             VBox dialogVbox = new VBox(20);
185                             dialogVbox.getChildren().add(new Text
186                             ("This is a Dialog"));
187                             TextField low = new TextField();
188                             TextField high = new TextField();
189
190                             grid2.add(new Label("Low Trigger: "),
191                             0, 1);
192                             grid2.add(low, 1, 1);
193                             grid2.add(new Label("High Trigger: ")
194                             , 0, 2);
195                             grid2.add(high, 1, 2);
196                             Scene dialogScene = new Scene(grid2,
197                             400, 400);

```

```

190                     Button cancel = new Button("Cancel");
191                     grid2.add(cancel, 1, 3);
192                     Button submit = new Button("Submit");
193                     grid2.add(submit, 0, 3);
194                     cancel.setOnAction(new EventHandler<
195                         ActionEvent>() {
196                             @Override
197                             public void handle(ActionEvent e)
198                             {
199                                 dialog2.close();
200                                 }
201                                 );
202                                 final Text error = new Text();
203                                 grid2.add(error, 1, 8);
204                                 submit.setOnAction(new EventHandler<
205                                     ActionEvent>() { // this one
206                                         @Override
207                                         public void handle(ActionEvent
208                                         event) {
209                                             Watchlist e = new Watchlist(
210                                                 result.get(), low.getText(),
211                                                 high.getText(), price,
212                                                 "", ""
213                                                 , "");
214                                             p.watchlists.add(e);
215                                             e.visit(p);
216                                             dialog2.close();
217                                             Scene scene = ViewWatchlist(
218                                                 primaryStage, p, FPTS);
219                                             FPTS.stage.setScene(scene);
220                                         }
221                                         });
222                                         dialog2.setScene(dialogScene);
223                                         dialog2.show();
224                                         }
225                                         );
226                                         //Action for removing equities
227                                         remove.setOnAction(new EventHandler<ActionEvent>(
228                                         ) {
229                                             @Override
230                                             public void handle(ActionEvent event) {
231                                                 ArrayList<String> choices = new ArrayList<

```

```

226 <> () ;
227             for(Watchlist e : p.watchlists) {
228                     choices.add(e.tickerSymbol);
229             }
230             ChoiceDialog<String> dialog = new
231                     ChoiceDialog<>("None", choices);
232                     dialog.setContentText("Choose the
233                     watchlist item you wish to remove:");
234                     Optional<String> result = dialog.
235                     showAndWait();
236                     if (result.isPresent()) {
237                         Watchlist w = null;
238                         for(Watchlist e : p.watchlists) {
239                             if(e.tickerSymbol.equals(result.
240                             get())) {
241                                 w = e;
242                                 }
243                                 p.watchlists.remove(w);
244                                 Scene scene = ViewWatchlist(
245                     primaryStage, p, FPTS);
246                     FPTS.stage.setScene(scene);
247                     }
248                     }
249                     high.setOnAction(new EventHandler<ActionEvent>()
250                     {
251                         @Override
252                         public void handle(ActionEvent event) {
253                             ArrayList<String> choices = new ArrayList
254 <> () ;
255                             for(Watchlist e : p.watchlists) {
256                                 choices.add(e.tickerSymbol);
257                             }
258                             ChoiceDialog<String> dialog = new
259                             ChoiceDialog<>("None", choices);
260                             dialog.setContentText("Choose the
261                             watchlist item you wish to add a high trigger to:");
262                             Optional<String> result = dialog.
263                             showAndWait();
264                             if (result.isPresent()) {
265                                 String lower = "";
266                                 for(Watchlist e : p.watchlists) {

```

```

261                     if(e.tickerSymbol.equals(result.
262                         get())){
263                             lower = e.lowtrigger;
264                         }
265                     final String truelow = lower;
266
267                     final Stage dialog2 = new Stage();
268                     dialog2.initOwner(primaryStage);
269                     GridPane grid2 = new GridPane();
270                     grid2.setAlignment(Pos.CENTER);
271                     grid2.setHgap(10);
272                     grid2.setVgap(10);
273                     grid2.setPadding(new Insets(5, 10, 5,
274                         10));
275
276                     VBox dialogVbox = new VBox(20);
277                     dialogVbox.getChildren().add(new Text
278                         ("This is a Dialog"));
279                     TextField high = new TextField();
280
281                     grid2.add(new Label("High Trigger: ")
282                         , 0, 2);
283                     grid2.add(high, 1, 2);
284                     Scene dialogScene = new Scene(grid2,
285                         400, 400);
286
287                     Button cancel = new Button("Cancel");
288                     grid2.add(cancel, 1, 3);
289                     Button submit = new Button("Submit");
290                     grid2.add(submit, 0, 3);
291                     cancel.setOnAction(new EventHandler<
292                         ActionEvent>() {
293
294                         @Override
295                         public void handle(ActionEvent e)
296                         {
297
298                             dialog2.close();
299                         }
300                     );
301                     final Text error = new Text();
302                     grid2.add(error, 1, 8);
303                     submit.setOnAction(new EventHandler<
304                         ActionEvent>() { // this one
305
306                         @Override
307                         public void handle(ActionEvent
308                             event) {

```

```

297                     if (high.getText().isEmpty())
298                     {
299                         error.setFill(Color.
300                         FIREBRICK);
301                         error.setText("Fill in
302                         all fields");
303                     } else if (!FPTS.isDouble(
304                         high.getText())) {
305                         error.setFill(Color.
306                         FIREBRICK);
307                         error.setText("Enter
308                         information in correct format");
309                     }
310                     else {
311                         for (Watchlist e : p.
312                             watchlists) {
313                             if (!truelow.equals("") )
314                             if (Double.
315                                 parseDouble(high.getText()) < Double.parseDouble(truelow)
316                             ) {
317                                 error.setFill
318                                 (Color.FIREBRICK);
319                                 error.setText
320                                 ("High trigger can't be lower than low trigger");
321                                 break;
322                             }
323                         }
324                     }
325                 }

```

```

326
327             } );
328
329             dialog2.setScene(dialogScene);
330             dialog2.show();
331         }
332     }
333 }
334
335 low.setOnAction(new EventHandler<ActionEvent>() {
336     @Override
337     public void handle(ActionEvent event) {
338         ArrayList<String> choices = new ArrayList
339             <>();
340         for(Watchlist e : p.watchlists){
341             choices.add(e.tickerSymbol);
342         }
343         ChoiceDialog<String> dialog = new
344             ChoiceDialog<>("None", choices);
345         dialog.setContentText("Choose the
346             watchlist item you wish to add a low trigger to:");
347         Optional<String> result = dialog.
348             showAndWait();
349         if (result.isPresent()) {
350             String higher = "";
351             for(Watchlist e : p.watchlists){
352                 if(e.tickerSymbol.equals(result.
353                     get())) {
354                     higher = e.hightrigger;
355                 }
356             }
357             final String truehigh = higher;
358
359             final Stage dialog2 = new Stage();
360             dialog2.initOwner(primaryStage);
361             GridPane grid2 = new GridPane();
362             grid2.setAlignment(Pos.CENTER);
363             grid2.setHgap(10);
364             grid2.setVgap(10);
365             grid2.setPadding(new Insets(5, 10, 5,
366                 10));
367
368             VBox dialogVbox = new VBox(20);
369             dialogVbox.getChildren().add(new Text
("This is a Dialog"));

```

```

364                     TextField low = new TextField();
365                     grid2.add(new Label("Low Trigger: "),
366                         0, 1);
366                     grid2.add(low, 1, 1);
367                     Scene dialogScene = new Scene(grid2,
368                         400, 400);
368                     Button cancel = new Button("Cancel");
369                     grid2.add(cancel, 1, 3);
370                     Button submit = new Button("Submit");
371                     grid2.add(submit, 0, 3);
372                     cancel.setOnAction(new EventHandler<
373                         ActionEvent>() {
374                             @Override
374                             public void handle(ActionEvent e)
375                             {
376                                 dialog2.close();
376                             }
377                             });
378                     final Text error = new Text();
379                     grid2.add(error, 1, 8);
380                     submit.setOnAction(new EventHandler<
381                         ActionEvent>() { // this one
382                             @Override
382                             public void handle(ActionEvent
383                             event) {
383                                 if (low.getText().isEmpty())
384                                 {
384                                     error.setFill(Color.
385                                         FIREBRICK);
385                                     error.setText("Fill in
385                                         all fields");
386                                 } else if (!FPTS.isDouble(low
386                                     .getText())))
387                                 {
387                                     error.setFill(Color.
387                                         FIREBRICK);
388                                     error.setText("Enter
388                                         information in correct format");
389                                 }
390                                 else {
391                                     for (Watchlist e : p.
391                                         watchlists) {
392                                         if (truehigh != "") {
392                                             if (Double.
393                                                 parseDouble(truehigh) < Double.parseDouble(low.getText()))
393                                         }
}

```

```

394                                     error.setFill
395                                         (Color.FIREBRICK);
396                                         error.setText
397                                         ("Low trigger can't be higher than high trigger");
398                                         break;
399                                         }
400                                         }
401                                         if (e.tickerSymbol.
402                                         equals(result.get())) {
403                                         e.lowtrigger =
404                                         low.getText().trim();
405                                         e.visit(p);
406                                         dialog2.close();
407                                         Scene scene =
408                                         FPTS.stage.
409                                         setScene(scene);
410                                         }
411                                         }
412                                         }
413                                         );
414                                         dialog2.setScene(dialogScene);
415                                         dialog2.show();
416                                         }
417                                         }
418                                         }
419                                         );
420                                         reset.setOnAction(new EventHandler<ActionEvent>()
421                                         {
422                                         @Override
423                                         public void handle(ActionEvent event) {
424                                         ArrayList<String> choices = new ArrayList
425                                         <>();
426                                         for(Watchlist e : p.watchlists){
427                                         choices.add(e.tickerSymbol);
428                                         }
429                                         ChoiceDialog<String> dialog = new
430                                         ChoiceDialog<>("None", choices);
431                                         dialog.setContentText("Choose the
432                                         watchlist item you wish to reset:");
433                                         }
434                                         }
435                                         }
436                                         }
437                                         }
438                                         }
439                                         }
440                                         }
441                                         }
442                                         }
443                                         }
444                                         }
445                                         }
446                                         }
447                                         }
448                                         }
449                                         }
450                                         }
451                                         }
452                                         }
453                                         }
454                                         }
455                                         }
456                                         }
457                                         }
458                                         }
459                                         }
460                                         }
461                                         }
462                                         }
463                                         }
464                                         }
465                                         }
466                                         }
467                                         }
468                                         }
469                                         }
470                                         }
471                                         }
472                                         }
473                                         }
474                                         }
475                                         }
476                                         }
477                                         }
478                                         }
479                                         }
480                                         }
481                                         }
482                                         }
483                                         }
484                                         }
485                                         }
486                                         }
487                                         }
488                                         }
489                                         }
490                                         }
491                                         }
492                                         }
493                                         }
494                                         }
495                                         }
496                                         }
497                                         }
498                                         }
499                                         }
500                                         }
501                                         }
502                                         }
503                                         }
504                                         }
505                                         }
506                                         }
507                                         }
508                                         }
509                                         }
510                                         }
511                                         }
512                                         }
513                                         }
514                                         }
515                                         }
516                                         }
517                                         }
518                                         }
519                                         }
520                                         }
521                                         }
522                                         }
523                                         }
524                                         }
525                                         }
526                                         }
527                                         }
528                                         }
529                                         }
530                                         }
531                                         }
532                                         }
533                                         }
534                                         }
535                                         }
536                                         }
537                                         }
538                                         }
539                                         }
540                                         }
541                                         }
542                                         }
543                                         }
544                                         }
545                                         }
546                                         }
547                                         }
548                                         }
549                                         }
550                                         }
551                                         }
552                                         }
553                                         }
554                                         }
555                                         }
556                                         }
557                                         }
558                                         }
559                                         }
560                                         }
561                                         }
562                                         }
563                                         }
564                                         }
565                                         }
566                                         }
567                                         }
568                                         }
569                                         }
570                                         }
571                                         }
572                                         }
573                                         }
574                                         }
575                                         }
576                                         }
577                                         }
578                                         }
579                                         }
580                                         }
581                                         }
582                                         }
583                                         }
584                                         }
585                                         }
586                                         }
587                                         }
588                                         }
589                                         }
590                                         }
591                                         }
592                                         }
593                                         }
594                                         }
595                                         }
596                                         }
597                                         }
598                                         }
599                                         }
599                                         }
600                                         }
601                                         }
602                                         }
603                                         }
604                                         }
605                                         }
606                                         }
607                                         }
608                                         }
609                                         }
609                                         }
610                                         }
611                                         }
612                                         }
613                                         }
614                                         }
615                                         }
616                                         }
617                                         }
618                                         }
619                                         }
619                                         }
620                                         }
621                                         }
622                                         }
623                                         }
624                                         }
625                                         }
626                                         }
627                                         }
628                                         }
629                                         }
629                                         }
630                                         }
631                                         }
632                                         }
633                                         }
634                                         }
635                                         }
636                                         }
637                                         }
638                                         }
639                                         }
639                                         }
640                                         }
641                                         }
642                                         }
643                                         }
644                                         }
645                                         }
646                                         }
647                                         }
648                                         }
649                                         }
649                                         }
650                                         }
651                                         }
652                                         }
653                                         }
654                                         }
655                                         }
656                                         }
657                                         }
658                                         }
659                                         }
659                                         }
660                                         }
661                                         }
662                                         }
663                                         }
664                                         }
665                                         }
666                                         }
667                                         }
668                                         }
669                                         }
669                                         }
670                                         }
671                                         }
672                                         }
673                                         }
674                                         }
675                                         }
676                                         }
677                                         }
678                                         }
679                                         }
679                                         }
680                                         }
681                                         }
682                                         }
683                                         }
684                                         }
685                                         }
686                                         }
687                                         }
688                                         }
689                                         }
689                                         }
690                                         }
691                                         }
692                                         }
693                                         }
694                                         }
695                                         }
696                                         }
697                                         }
698                                         }
699                                         }
699                                         }
700                                         }
701                                         }
702                                         }
703                                         }
704                                         }
705                                         }
706                                         }
707                                         }
708                                         }
709                                         }
709                                         }
710                                         }
711                                         }
712                                         }
713                                         }
714                                         }
715                                         }
716                                         }
717                                         }
718                                         }
719                                         }
719                                         }
720                                         }
721                                         }
722                                         }
723                                         }
724                                         }
725                                         }
726                                         }
727                                         }
728                                         }
729                                         }
729                                         }
730                                         }
731                                         }
732                                         }
733                                         }
734                                         }
735                                         }
736                                         }
737                                         }
738                                         }
739                                         }
739                                         }
740                                         }
741                                         }
742                                         }
743                                         }
744                                         }
745                                         }
746                                         }
747                                         }
748                                         }
749                                         }
749                                         }
750                                         }
751                                         }
752                                         }
753                                         }
754                                         }
755                                         }
756                                         }
757                                         }
758                                         }
759                                         }
759                                         }
760                                         }
761                                         }
762                                         }
763                                         }
764                                         }
765                                         }
766                                         }
767                                         }
768                                         }
769                                         }
769                                         }
770                                         }
771                                         }
772                                         }
773                                         }
774                                         }
775                                         }
776                                         }
777                                         }
778                                         }
779                                         }
779                                         }
780                                         }
781                                         }
782                                         }
783                                         }
784                                         }
785                                         }
786                                         }
787                                         }
788                                         }
789                                         }
789                                         }
790                                         }
791                                         }
792                                         }
793                                         }
794                                         }
795                                         }
796                                         }
797                                         }
798                                         }
799                                         }
799                                         }
800                                         }
801                                         }
802                                         }
803                                         }
804                                         }
805                                         }
806                                         }
807                                         }
808                                         }
809                                         }
809                                         }
810                                         }
811                                         }
812                                         }
813                                         }
814                                         }
815                                         }
816                                         }
817                                         }
818                                         }
819                                         }
819                                         }
820                                         }
821                                         }
822                                         }
823                                         }
824                                         }
825                                         }
826                                         }
827                                         }
828                                         }
829                                         }
829                                         }
830                                         }
831                                         }
832                                         }
833                                         }
834                                         }
835                                         }
836                                         }
837                                         }
838                                         }
839                                         }
839                                         }
840                                         }
841                                         }
842                                         }
843                                         }
844                                         }
845                                         }
846                                         }
847                                         }
848                                         }
849                                         }
849                                         }
850                                         }
851                                         }
852                                         }
853                                         }
854                                         }
855                                         }
856                                         }
857                                         }
858                                         }
859                                         }
859                                         }
860                                         }
861                                         }
862                                         }
863                                         }
864                                         }
865                                         }
866                                         }
867                                         }
868                                         }
869                                         }
869                                         }
870                                         }
871                                         }
872                                         }
873                                         }
874                                         }
875                                         }
876                                         }
877                                         }
878                                         }
879                                         }
879                                         }
880                                         }
881                                         }
882                                         }
883                                         }
884                                         }
885                                         }
886                                         }
887                                         }
888                                         }
889                                         }
889                                         }
890                                         }
891                                         }
892                                         }
893                                         }
894                                         }
895                                         }
896                                         }
897                                         }
898                                         }
899                                         }
899                                         }
900                                         }
901                                         }
902                                         }
903                                         }
904                                         }
905                                         }
906                                         }
907                                         }
908                                         }
909                                         }
909                                         }
910                                         }
911                                         }
912                                         }
913                                         }
914                                         }
915                                         }
916                                         }
917                                         }
918                                         }
919                                         }
919                                         }
920                                         }
921                                         }
922                                         }
923                                         }
924                                         }
925                                         }
926                                         }
927                                         }
928                                         }
929                                         }
929                                         }
930                                         }
931                                         }
932                                         }
933                                         }
934                                         }
935                                         }
936                                         }
937                                         }
938                                         }
939                                         }
939                                         }
940                                         }
941                                         }
942                                         }
943                                         }
944                                         }
945                                         }
946                                         }
947                                         }
948                                         }
949                                         }
949                                         }
950                                         }
951                                         }
952                                         }
953                                         }
954                                         }
955                                         }
956                                         }
957                                         }
958                                         }
959                                         }
959                                         }
960                                         }
961                                         }
962                                         }
963                                         }
964                                         }
965                                         }
966                                         }
967                                         }
968                                         }
969                                         }
969                                         }
970                                         }
971                                         }
972                                         }
973                                         }
974                                         }
975                                         }
976                                         }
977                                         }
978                                         }
979                                         }
979                                         }
980                                         }
981                                         }
982                                         }
983                                         }
984                                         }
985                                         }
986                                         }
987                                         }
988                                         }
989                                         }
989                                         }
990                                         }
991                                         }
992                                         }
993                                         }
994                                         }
995                                         }
996                                         }
997                                         }
998                                         }
999                                         }
999                                         }
1000                                         }
1001                                         }
1002                                         }
1003                                         }
1004                                         }
1005                                         }
1006                                         }
1007                                         }
1008                                         }
1008                                         }
1009                                         }
1010                                         }
1011                                         }
1012                                         }
1013                                         }
1014                                         }
1015                                         }
1016                                         }
1017                                         }
1018                                         }
1019                                         }
1019                                         }
1020                                         }
1021                                         }
1022                                         }
1023                                         }
1024                                         }
1025                                         }
1026                                         }
1027                                         }
1028                                         }
1029                                         }
1029                                         }
1030                                         }
1031                                         }
1032                                         }
1033                                         }
1034                                         }
1035                                         }
1036                                         }
1037                                         }
1038                                         }
1039                                         }
1039                                         }
1040                                         }
1041                                         }
1042                                         }
1043                                         }
1044                                         }
1045                                         }
1046                                         }
1047                                         }
1048                                         }
1049                                         }
1049                                         }
1050                                         }
1051                                         }
1052                                         }
1053                                         }
1054                                         }
1055                                         }
1056                                         }
1057                                         }
1058                                         }
1059                                         }
1059                                         }
1060                                         }
1061                                         }
1062                                         }
1063                                         }
1064                                         }
1065                                         }
1066                                         }
1067                                         }
1068                                         }
1069                                         }
1069                                         }
1070                                         }
1071                                         }
1072                                         }
1073                                         }
1074                                         }
1075                                         }
1076                                         }
1077                                         }
1078                                         }
1079                                         }
1079                                         }
1080                                         }
1081                                         }
1082                                         }
1083                                         }
1084                                         }
1085                                         }
1086                                         }
1087                                         }
1088                                         }
1088                                         }
1089                                         }
1089                                         }
1090                                         }
1091                                         }
1092                                         }
1093                                         }
1094                                         }
1095                                         }
1095                                         }
1096                                         }
1097                                         }
1098                                         }
1099                                         }
1099                                         }
1100                                         }
1101                                         }
1102                                         }
1103                                         }
1104                                         }
1105                                         }
1106                                         }
1107                                         }
1108                                         }
1109                                         }
1109                                         }
1110                                         }
1111                                         }
1112                                         }
1113                                         }
1114                                         }
1115                                         }
1116                                         }
1117                                         }
1118                                         }
1119                                         }
1119                                         }
1120                                         }
1121                                         }
1122                                         }
1123                                         }
1124                                         }
1125                                         }
1126                                         }
1127                                         }
1128                                         }
1129                                         }
1129                                         }
1130                                         }
1131                                         }
1132                                         }
1133                                         }
1134                                         }
1135                                         }
1136                                         }
1137                                         }
1138                                         }
1139                                         }
1139                                         }
1140                                         }
1141                                         }
1142                                         }
1143                                         }
1144                                         }
1145                                         }
1146                                         }
1147                                         }
1148                                         }
1149                                         }
1149                                         }
1150                                         }
1151                                         }
1152                                         }
1153                                         }
1154                                         }
1155                                         }
1156                                         }
1157                                         }
1158                                         }
1159                                         }
1159                                         }
1160                                         }
1161                                         }
1162                                         }
1163                                         }
1164                                         }
1165                                         }
1166                                         }
1167                                         }
1168                                         }
1169                                         }
1169                                         }
1170                                         }
1171                                         }
1172                                         }
1173                                         }
1174                                         }
1175                                         }
1176                                         }
1177                                         }
1178                                         }
1179                                         }
1179                                         }
1180                                         }
1181                                         }
1182                                         }
1183                                         }
1184                                         }
1185                                         }
1186                                         }
1187                                         }
1188                                         }
1188                                         }
1189                                         }
1189                                         }
1190                                         }
1191                                         }
1192                                         }
1193                                         }
1194                                         }
1195                                         }
1196                                         }
1197                                         }
1198                                         }
1198                                         }
1199                                         }
1199                                         }
1200                                         }
1201                                         }
1202                                         }
1203                                         }
1204                                         }
1205                                         }
1206                                         }
1207                                         }
1208                                         }
1209                                         }
1209                                         }
1210                                         }
1211                                         }
1212                                         }
1213                                         }
1214                                         }
1215                                         }
1216                                         }
1217                                         }
1218                                         }
1219                                         }
1219                                         }
1220                                         }
1221                                         }
1222                                         }
1223                                         }
1224                                         }
1225                                         }
1226                                         }
1227                                         }
1228                                         }
1229                                         }
1229                                         }
1230                                         }
1231                                         }
1232                                         }
1233                                         }
1234                                         }
1235                                         }
1236                                         }
1237                                         }
1238                                         }
1239                                         }
1239                                         }
1240                                         }
1241                                         }
1242                                         }
1243                                         }
1244                                         }
1245                                         }
1246                                         }
1247                                         }
1248                                         }
1249                                         }
1249                                         }
1250                                         }
1251                                         }
1252                                         }
1253                                         }
1254                                         }
1255                                         }
1256                                         }
1257                                         }
1258                                         }
1259                                         }
1259                                         }
1260                                         }
1261                                         }
1262                                         }
1263                                         }
1264                                         }
1265                                         }
1266                                         }
1267                                         }
1268                                         }
1269                                         }
1269                                         }
1270                                         }
1271                                         }
1272                                         }
1273                                         }
1274                                         }
1275                                         }
1276                                         }
1277                                         }
1278                                         }
1279                                         }
1279                                         }
1280                                         }
1281                                         }
1282                                         }
1283                                         }
1284                                         }
1285                                         }
1286                                         }
1287                                         }
1288                                         }
1288                                         }
1289                                         }
1289                                         }
1290                                         }
1291                                         }
1292                                         }
1293                                         }
1294                                         }
1295                                         }
1296                                         }
1297                                         }
1298                                         }
1298                                         }
1299                                         }
1299                                         }
1300                                         }
1301                                         }
1302                                         }
1303                                         }
1304                                         }
1305                                         }
1306                                         }
1307                                         }
1308                                         }
1309                                         }
1309                                         }
1310                                         }
1311                                         }
1312                                         }
1313                                         }
1314                                         }
1315                                         }
1316                                         }
1317                                         }
1318                                         }
1319                                         }
1319                                         }
1320                                         }
1321                                         }
1322                                         }
1323                                         }
1324                                         }
1325                                         }
1326                                         }
1327                                         }
1328                                         }
1329                                         }
1329                                         }
1330                                         }
1331                                         }
1332                                         }
1333                                         }
1334                                         }
1335                                         }
1336                                         }
1337                                         }
1338                                         }
1339                                         }
1339                                         }
1340                                         }
1341                                         }
1342                                         }
1343                                         }
1344                                         }
1345                                         }
1346                                         }
1347                                         }
1348                                         }
1349                                         }
1349                                         }
1350                                         }
1351                                         }
1352                                         }
1353                                         }
1354                                         }
1355                                         }
1356                                         }
1357                                         }
1358                                         }
1359                                         }
1359                                         }
1360                                         }
1361                                         }
1362                                         }
1363                                         }
1364                                         }
1365                                         }
1366                                         }
1367                                         }
1368                                         }
1369                                         }
1369                                         }
1370                                         }
1371                                         }
1372                                         }
1373                                         }
1374                                         }
1375                                         }
1376                                         }
1377                                         }
1378                                         }
1379                                         }
1379                                         }
1380                                         }
1381                                         }
1382                                         }
1383                                         }
1384                                         }
1385                                         }
1386                                         }
1387                                         }
1388                                         }
1388                                         }
1389                                         }
1389                                         }
1390                                         }
1391                                         }
1392                                         }
1393                                         }
1394                                         }
1395                                         }
1396                                         }
1397                                         }
1398                                         }
1398                                         }
1399                                         }
1399                                         }
1400                                         }
1401                                         }
1402                                         }
1403                                         }
1404                                         }
1405                                         }
1406                                         }
1407                                         }
1408                                         }
1409                                         }
1409                                         }
1410                                         }
1411                                         }
1412                                         }
1413                                         }
1414                                         }
1415                                         }
1416                                         }
1417                                         }
1418                                         }
1419                                         }
1419                                         }
1420                                         }
1421                                         }
1422                                         }
1423                                         }
1424                                         }
1425                                         }
1426                                         }
1427                                         }
1428                                         }
1429                                         }
1429                                         }
1430                                         }
1431                                         }
1432                                         }
1433                                         }
1434                                         }
1435                                         }
1436                                         }
1437                                         }
1438                                         }
1439                                         }
1439                                         }
1440                                         }
1441                                         }
1442                                         }
1443                                         }
1444                                         }
1445                                         }
1446                                         }
1447                                         }
1448                                         }
1449                                         }
1449                                         }
1450                                         }
1451                                         }
1452                                         }
1453                                         }
1454                                         }
1455                                         }
1456                                         }
1457                                         }
1458                                         }
1459                                         }
1459                                         }
1460                                         }
1461                                         }
1462                                         }
1463                                         }
1464                                         }
1465                                         }
1466                                         }
1467                                         }
1468                                         }
1469                                         }
1469                                         }
1470                                         }
1471                                         }
1472                                         }
1473                                         }
1474                                         }
1475                                         }
1476                                         }
1477                                         }
1478                                         }
1479                                         }
1479                                         }
1480                                         }
1481                                         }
1482                                         }
1483                                         }
1484                                         }
1485                                         }
1486                                         }
1487                                         }
1488                                         }
1488                                         }
1489                                         }
1489                                         }
1490                                         }
1491                                         }
1492                                         }
1493                                         }
1494                                         }
1495                                         }
1496                                         }
1497                                         }
1498                                         }
1498                                         }
1499                                         }
1499                                         }
1500                                         }
1501                                         }
1502                                         }
1503                                         }
1504                                         }
1505                                         }
1506                                         }
1507                                         }
1508                                         }
1509                                         }
1509                                         }
1510                                         }
1511                                         }
1512                                         }
1513                                         }
1514                                         }
1515                                         }
1516                                         }
1517                                         }
1518                                         }
1519                                         }
1519                                         }
1520                                         }
1521                                         }
1522                                         }
1523                                         }
1524                                         }
1525                                         }
1526                                         }
1527                                         }
1528                                         }
1529                                         }
1529                                         }
1530                                         }
1531                                         }
1532                                         }
1533                                         }
1534                                         }
1535                                         }
1536                                         }
1537                                         }
1538                                         }
1539                                         }
1539                                         }
1540                                         }
1541                                         }
1542                                         }
1543                                         }
1544                                         }
1545                                         }
1546                                         }
1547                                         }
1548                                         }
1549                                         }
1549                                         }
1550                                         }
1551                                         }
1552                                         }
1553                                         }
1554                                         }
1555                                         }
1556                                         }
1557                                         }
1558                                         }
1559                                         }
1559                                         }
1560                                         }
1561                                         }
1562                                         }
1563                                         }
1564                                         }
1565                                         }
1566                                         }
1567                                         }
1568                                         }
1569                                         }
1569                                         }
1570                                         }
1571                                         }
1572                                         }
1573                                         }
1574                                         }
1575                                         }
1576                                         }
1577                                         }
1578                                         }
1579                                         }
1579                                         }
1580                                         }
1581                                         }
1582                                         }
1583                                         }
1584                                         }
1585                                         }
1586                                         }
1587                                         }
1588                                         }
1588                                         }
1589                                         }
1589                                         }
1590                                         }
1591                                         }
1592                                         }
1593                                         }
1594                                         }
1595                                         }
1596                                         }
1597                                         }
1598                                         }
1598                                         }
1599                                         }
1599                                         }
1600                                         }
1601                                         }
1602                                         }
1603                                         }
1604                                         }
1605                                         }
1606                                         }
1607                                         }
1608                                         }
1609                                         }
1609                                         }
1610                                         }
1611                                         }
1612                                         }
1613                                         }
1614                                         }
1615                                         }
1616                                         }
1617                                         }
1618                                         }
1619                                         }
1619                                         }
1620                                         }
1621                                         }
1622                                         }
1623                                         }
1624                                         }
1625                                         }
1626                                         }
1627                                         }
1628                                         }
1629                                         }
1629                                         }
1630                                         }
1631                                         }
1632                                         }
1633                                         }
1634                                         }
1635                                         }
1636                                         }
1637                                         }
1638                                         }
1639                                         }
1639                                         }
1640                                         }
1641                                         }
1642                                         }
1643                                         }
1644                                         }
1645                                         }
1646                                         }
1647                                         }
1648                                         }
1649                                         }
1649                                         }
1650                                         }
1651                                         }
1652                                         }
1653                                         }
1654                                         }
1655                                         }
1656                                         }
1657                                         }
1658                                         }
1659                                         }
1659                                         }
1660                                         }
1661                                         }
1662                                         }
1663                                         }
1664                                         }
1665                                         }
1666                                         }
1667                                         }
1668                                         }
1669                                         }
1669                                         }
1670                                         }
1671                                         }
1672                                         }
1673                                         }
1674                                         }
1675                                         }
1676                                         }
1677                                         }
1678                                         }
1679                                         }
1679                                         }
1680                                         }
1681                                         }
1682                                         }
1683                                         }
1684                                         }
1685                                         }
1686                                         }
1687                                         }
1688                                         }
1688                                         }
1689                                         }
1689                                         }
1690                                         }
1691                                         }
1692                                         }
1693                                         }
1694                                         }
1695                                         }
1696                                         }
1697                                         }
1698                                         }
1698                                         }
1699                                         }
1699                                         }
1700                                         }
1701                                         }
1702                                         }
1703                                         }
1704                                         }
1705                                         }
1706                                         }
1707                                         }
1708                                         }
1709                                         }
1709                                         }
1710                                         }
1711                                         }
1712                                         }
1713                                         }
1714                                         }
1715                                         }
1716                                         }
1717                                         }
1718                                         }
1719                                         }
1719                                         }
1720                                         }
1721                                         }
1722                                         }
1723                                         }
1724                                         }
1725                                         }
1726                                         }
1727                                         }
1728                                         }
1729                                         }
1729                                         }
1730                                         }
1731                                         }
1732                                         }
1733                                         }
1734                                         }
1735                                         }
1736                                         }
1737                                         }
1738                                         }
1739                                         }
1739                                         }
1740                                         }
1741                                         }
1742                                         }
1743                                         }
1744                                         }
1745                                         }
1746                                         }
1747                                         }
1748                                         }
1749                                         }
1749                                         }
1750                                         }
1751                                         }
1752                                         }
1753                                         }
1754                                         }
1755                                         }
1756                                         }
1757                                         }
1758                                         }
1759                                         }
1759                                         }
1760                                         }
1761                                         }
1762                                         }
1763                                         }
1764                                         }
1765                                         }
1766                                         }
1767                                         }
1768                                         }
1769                                         }
1769                                         }
1770                                         }
1771                                         }
1772                                         }
1773                                         }
1774                                         }
1775                                         }
1776                                         }
1777                                         }
1778                                         }
1779                                         }
1779                                         }
1780                                         }
1781                                         }
1782                                         }
1783                                         }
1784                                         }
1785                                         }
1786                                         }
1787                                         }
1788                                         }
1788                                         }
1789                                         }
1789                                         }
1790                                         }
1791                                         }
1792                                         }
1793                                         }
1794                                         }
1795                                         }
1796                                         }
1797                                         }
1798                                         }
1798                                         }
1799                                         }
1799                                         }
1800                                         }
1801                                         }
1802                                         }
1803                                         }
1804                                         }
1805                                         }
1806                                         }
1807                                         }
1808                                         }
1809                                         }
1809                                         }
1810                                         }
1811                                         }
1812                                         }
1813                                         }
1814                                         }
1815                                         }
1816                                         }
1817                                         }
1818                                         }
1819                                         }
1819                                         }
1820                                         }
1821                                         }
1822                                         }
1823                                         }
1824                                         }
1825                                         }
1826                                         }
1827                                         }
1828                                         }
1829                                         }
1829                                         }
1830                                         }
1831                                         }
1832                                         }
1833                                         }
1834                                         }
1835                                         }
1836                                         }
1837                                         }
1838                                         }
1839                                         }
1839                                         }
1840                                         }
1841                                         }
1842                                         }
1843                                         }
1844                                         }
1845                                         }
1846                                         }
1847                                         }
1848                                         }
1849                                         }
1849                                         }
1850                                         }
1851                                         }
1852                                         }
1853                                         }
1854                                         }
1855                                         }
1856                                         }
1857                                         }
1858                                         }
1859                                         }
1859                                         }
1860                                         }
1861                                         }
1862                                         }
1863                                         }
1864                                         }
1865                                         }
1866                                         }
1867                                         }
1868                                         }
1869                                         }
1869                                         }
1870                                         }
1871                                         }
1872                                         }
1873                                         }
1874                                         }
1875                                         }
1876                                         }
1877                                         }
1878                                         }
1879                                         }
1879                                         }
1880                                         }
1881                                         }
1882                                         }
1883                                         }
1884                                         }
1885                                         }
1886                                         }
1887                                         }
1888                                         }
1889                                         }
1889                                         }
1890                                         }
1891                                         }
1892                                         }
1893                                         }
1894                                         }
1895                                         }
1896                                         }
1897                                         }
1898                                         }
1898                                         }
1899                                         }
1899                                         }
1900                                         }
1901                                         }
1902                                         }
1903                                         }
1904                                         }
1905                                         }
1906                                         }
1907                                         }
1908                                         }
1909                                         }
1909                                         }
1910                                         }
1911                                         }
1912                                         }
1913                                         }
1914                                         }
1915                                         }
1916                                         }
1917                                         }
1918                                         }
1919                                         }
1919                                         }
1920                                         }
1921                                         }
1922                                         }
1923                                         }
1924                                         }
1925                                         }
1926                                         }
1927                                         }
1928                                         }
1929                                         }
1929                                         }
1930                                         }
1931                                         }
1932                                         }
1933                                         }
1934                                         }
1935                                         }
1936                                         }
1937                                         }
1938                                         }
1939                                         }
1939                                         }
1940                                         }
1941                                         }
1942                                         }
1943                                         }
1944                                         }
1945                                         }
1946                                         }
1947                                         }
1948                                         }
1949                                         }
1949                                         }
1950                                         }
1951                                         }
1952                                         }
1953                                         }
1954                                         }
1955                                         }
1956                                         }
1957                                         }
1958                                         }
1959                                         }
1959                                         }
1960                                         }
1961                                         }
1962                                         }
1963                                         }
1964                                         }
1965                                         }
1966                                         }
1967                                         }
1968                                         }
1969                                         }
1969                                         }
1970                                         }
1971                                         }
1972                                         }
1973                                         }
1974                                         }
1975                                         }
1976                                         }
1977                                         }
1978                                         }
1979                                         }
1979                                         }
1980                                         }
1981                                         }
1982                                         }
1983                                         }
1984                                         }
1985                                         }
1986                                         }
1987                                         }
1988                                         }
1988                                         }
1989                                         }
1989                                         }
1990                                         }
1991                                         }
1992                                         }
1993                                         }
1994                                         }
1995                                         }
1996                                         }
1997                                         }
1998                                         }
1998                                         }
1999                                         }
1999                                         }
2000                                         }
2001                                         }
2002                                         }
2003                                         }
2004                                         }
2005                                         }
2006                                         }
2007                                         }
2008                                         }
2009                                         }
2009                                         }
2010                                         }
2011                                         }
2012                                         }
2013                                         }
2014                                         }
2015                                         }
2016                                         }
2017                                         }
2018                                         }
2019                                         }
2019                                         }
2020                                         }
2021                                         }
2022                                         }
2023                                         }
2024                                         }
2025                                         }
2026                                         }
2027                                         }
2028                                         }
2029                                         }
2029                                         }
2030                                         }
2031                                         }
2032                                         }
2033                                         }
2034                                         }
2035                                         }
2036                                         }
2037                                         }
2038                                         }
2039                                         }
2039                                         }
2040                                         }
2041                                         }
2042                                         }
2043                                         }
2044                                         }
2045                                         }
2046                                         }
2047                                         }
2048                                         }
2049                                         }
2049                                         }
2050                                         }
2051                                         }
2052                                         }
2053                                         }
2054                                         }
2055                                         }
2056                                         }
2057                                         }
2058                                         }
2059                                         }
2059                                         }
2060                                         }
2061                                         }
2062                                         }
2063                                         }
2064                                         }
2065                                         }
2066                                         }
2067                                         }
2068                                         }
2069                                         }
2069                                         }
2070                                         }
2071                                         }
2072                                         }
2073                                         }
2074                                         }
2075                                         }
2076                                         }
2077                                         }
2078                                         }
2079                                         }
2079                                         }
2080                                         }
2081                                         }
2082                                         }
2083                                         }
2084                                         }
2085                                         }
2086                                         }
2087                                         }
2088                                         }
2088                                         }
2089                                         }
2089                                         }
2090                                         }
2091                                         }
2092                                         }
2093                                         }
2094                                         }
2095                                         }
2096                                         }
2097                                         }
2098                                         }
2098                                         }
2099                                         }
2099                                         }
2100                                         }
2101                                         }
2102                                         }
2103                                         }
2104                                         }
2105                                         }
2106                                         }
2107                                         }
2108                                         }
2109                                         }
2109                                         }
2110                                         }
2111                                         }
2112                                         }
2113                                         }
2114                                         }
2115                                         }
2116                                         }
2117                                         }
2118                                         }
2119                                         }
2119                                         }
2120                                         }
2121                                         }
2122                                         }
2123                                         }
2124                                         }
2125                                         }
2126                                         }
2127                                         }
2128                                         }
2129                                         }
2129                                         }
2130                                         }
2131                                         }
2132                                         }
2133                                         }
2134                                         }
2135                                         }
2136                                         }
2137                                         }
2138                                         }
2139                                         }
2139                                         }
2140                                         }
2141                                         }
2142                                         }
2143                                         }
2144                                         }
2145                                         }
2146                                         }
2147                                         }
2148                                         }
2149                                         }
214
```

```
429             Optional<String> result = dialog.  
430             showAndWait();  
431             if (result.isPresent()) {  
432                 for (Watchlist w : p.watchlists) {  
433                     if (w.tickerSymbol.equals(result.  
434                         get())) {  
435                         w.lowtriggered = "";  
436                         w.hightriggered = "";  
437                     }  
438                     Scene scene = ViewWatchlist(  
439                         primaryStage, p, FPTS);  
440                     FPTS.stage.setScene(scene);  
441                 }  
442             }  
443         }  
444         toolBar2.getItems().addAll(spacer, add, remove,  
445             high, low, reset);  
446  
447         border.setCenter(vbox);  
448         Scene scene = new Scene(border, 1200, 700);  
449         return scene;  
450     }  
451 }
```

```

1 package trunk.View;
2
3 import javafx.event.ActionEvent;
4 import javafx.event.EventHandler;
5 import javafx.geometry.Insets;
6 import javafx.geometry.Pos;
7 import javafx.scene.Scene;
8 import javafx.scene.chart.LineChart;
9 import javafx.scene.chart.NumberAxis;
10 import javafx.scene.chart.XYChart;
11 import javafx.scene.control.*;
12 import javafx.scene.layout.BorderPane;
13 import javafx.scene.layout.GridPane;
14 import javafx.scene.layout.VBox;
15 import javafx.scene.paint.Color;
16 import javafx.scene.text.Font;
17 import javafx.scene.text.FontWeight;
18 import javafx.scene.text.Text;
19 import javafx.stage.Stage;
20 import trunk.Control.*;
21 import trunk.Model.Portfolio;
22
23 /**
24 * Created by Daniel on 3/25/2016.
25 */
26 public class simulationView {
27     //GUI for simulations
28     public Scene simulation(final Stage primaryStage,
29         Portfolio p, FPTS FPTS) {
30         Text title = new Text("Simulation");
31         BorderPane border = FPTS.borderPane(primaryStage,p
32         ,title);
33
34         GridPane grid = new GridPane();
35         grid.setAlignment(Pos.TOP_CENTER);
36         grid.setHgap(10);
37         grid.setVgap(10);
38         grid.setPadding(new Insets(5, 10, 5, 10));
39         title.setFont(Font.font("Calibri", FontWeight.
40             NORMAL, 20));
41         grid.add(title,0,0,2,1);
42
43         Label type = new Label("Simulation Type:");
44         grid.add(type,0,1);
45         ComboBox typeBox = new ComboBox();

```

```

43         typeBox.getItems () .addAll (
44             "Bull Market",
45             "Bear Market",
46             "No Growth"
47         );
48         grid.add(typeBox,1,1);
49
50         Label intervals = new Label ("Interval:");
51         grid.add(intervals,0,2);
52         ComboBox intervalBox = new ComboBox();
53         intervalBox.getItems () .addAll (
54             "Days",
55             "Months",
56             "Years"
57         );
58         grid.add(intervalBox,1,2);
59
60         Label steps = new Label ("Steps: ");
61         grid.add(steps,0,3);
62         TextField s = new TextField();
63         grid.add(s,1,3);
64
65         Label per = new Label ("Percentage: ");
66         grid.add(per,0,4);
67         TextField pp = new TextField();
68         grid.add(pp,1,4);
69
70         Label sbs = new Label ("Step By Step ");
71         CheckBox cb = new CheckBox();
72         grid.addRow(5, sbs, cb);
73
74         Button submit = new Button ("Submit");
75         grid.add(submit,1,6);
76         submit.setOnAction(new EventHandler<ActionEvent>()
{
77             @Override
78             public void handle(ActionEvent event) {
79                 if(typeBox.getSelectionModel().isEmpty()
    || intervalBox.getSelectionModel().isEmpty() || s.getText()
    .isEmpty() || pp.getText().isEmpty()) {
80                     Text error = new Text ("Fill in all
fields");
81                     error.setFill(Color.RED);
82                     grid.add(error,0,8);
83                 }
}

```

```

84             if(!FPTS.isInt(pp.getText()) || !FPTS.
85               isDouble(s.getText())) {
86                 Text error = new Text("Correctly fill
87                   in all fields");
88                 error.setFill(Color.RED);
89                 grid.add(error,0,10);
90             }
91             else {
92               String type = typeBox.
93                 getSelectionModel().getSelectedItem().toString();
94               String interval = intervalBox.
95                 getSelectionModel().getSelectedItem().toString();
96               int time = 365;
97
98               final NumberAxis yAxis = new
99                 NumberAxis();
100              final NumberAxis xAxis = new
101                NumberAxis();
102              final LineChart<Number,Number>
103                lineChart =
104                  new LineChart<Number,Number>(
105                    xAxis,yAxis);
106
107              lineChart.setTitle("Portfolio Value
108                Simulation");
109
110              XYChart.Series series1 = new XYChart.
111                Series();
112              series1.setName("Portfolio Value");
113
114
115
116
117              if (type.equals("Bull Market")) {
118                BullMarket bullMarket = new

```

```

118 BullMarket();
119                                     double[] val = bullMarket.
120                                     simulate(p,Double.parseDouble(pp.getText().trim()),time,
121                                     Integer.parseInt(s.getText().trim()));
122                                     series1.getData().add(new XYChart
123                                     .Data(0, val[0]));
124                                     if (cb.isSelected()) {
125                                     Button next = new Button(">
126                                     Next Step");
127                                     grid.add(next, 0, 7);
128                                     Button skip = new Button(">>
129                                     Final Step");
130                                     grid.add(skip, 1, 7);
131                                     VBox data = new VBox();
132                                     lineChart.getData().add(
133                                     series1);
134                                     border.setBottom(lineChart);
135                                     next.setOnAction(new
136                                     EventHandler<ActionEvent>() {
137                                     int i = 1;
138                                     public void handle(
139                                     ActionEvent e) {
140                                     series1.getData().add
141                                     (new XYChart.Data(i, val[i]));
142                                     if (i + 1 < val.
143                                     length)
144                                     i++;
145                                     }
146                                     });
147                                     skip.setOnAction(new
148                                     EventHandler<ActionEvent>() {
149                                     @Override
150                                     public void handle(
151                                     ActionEvent event) {
152                                     for (int i = 1; i <
153                                     val.length; i++) {
154                                     series1.getData()
155                                     .add(new XYChart.Data(i, val[i]));
156                                     }
157                                     }
158                                     });

```

```

149 }
150     else {
151         for (int i = 1; i < val.
152             length; i++) {
153             series1.getData().add(new
154                 XYChart.Data(i, val[i]));
155             }
156         }
157     }
158     else if (type.equals("Bear Market"))
159     {
160         BearMarket bearMarket = new
161             BearMarket();
162         double[] val = bearMarket.
163             simulate(p, Double.parseDouble(pp.getText().trim()), time,
164             Integer.parseInt(s.getText().trim()));
165         series1.getData().add(new XYChart
166             .Data(0, val[0]));
167         if (cb.isSelected())
168             {
169                 Button next = new Button(""
170                     "Next Step");
171                 grid.add(next, 0, 7);
172                 Button skip = new Button(">>">
173                     Final Step");
174                 grid.add(skip, 1, 7);
175                 VBox data = new VBox();
176                 lineChart.getData().add(
177                     series1);
178                 border.setBottom(lineChart);
179                 next.setOnAction(new
180                     EventHandler<ActionEvent>() {
181                         int i = 1;
182                         public void handle(
183                             ActionEvent e) {
184                             series1.getData().add
185                             (new XYChart.Data(i, val[i]));
186                             if (i + 1 < val.
187                                 length)
188                                 i++;
189                         }

```

```

180                     } );
181
182                     skip.setOnAction( new
183                         EventHandler<ActionEvent>() {
184                             @Override
185                             public void handle(
186                                 ActionEvent event) {
187                                     for (int i = 1; i <
188                                         val.length; i++) {
189                                         series1.getData()
190                                         .add( new XYChart.Data(i, val[i]));
191                                     }
192                                 }
193                             else {
194                                 for (int i = 1; i < val.
195                                     length; i++) {
196                                     series1.getData().add( new
197                                         XYChart.Data(i, val[i]));
198                                     }
199
200                             else if (type.equals("No Growth")) {
201                                 NoGrowth noGrowth = new NoGrowth(
202                                     );
203                                 double[] val = noGrowth.simulate(
204                                     p, Double.parseDouble(pp.getText().trim()), time, Integer.
205                                     parseInt(s.getText().trim()));
206                                 series1.getData().add( new XYChart
207                                     .Data(0, val[0]));
208
209                                 if (cb.isSelected()) {
210                                     Button next = new Button(""
211                                         Next Step);
212                                     grid.add(next, 0, 7);
213                                     Button skip = new Button(">>">
214                                         Final Step);
215                                     grid.add(skip, 1, 7);
216                                     VBox data = new VBox();
217                                     lineChart.getData().add(

```

```
212 series1);  
213 border.setBottom(lineChart);  
214  
215 next.setOnAction(new  
    EventHandler<ActionEvent>() {  
216        int i = 1;  
217        public void handle(  
            ActionEvent e) {  
218            series1.getData().add  
                (new XYChart.Data(i, val[i]));  
219            if (i + 1 < val.  
                length)  
                i++;  
220        }  
221    } );  
222  
223 skip.setOnAction(new  
    EventHandler<ActionEvent>() {  
224        @Override  
225        public void handle(  
            ActionEvent event) {  
226            for (int i = 1; i <  
                val.length; i++) {  
227                series1.getData()  
                    .add(new XYChart.Data(i, val[i]));  
228            }  
229        }  
230    } );  
231  
232 }  
233 else {  
234     for (int i = 1; i < val.  
         length; i++) {  
235         series1.getData().add(new  
             XYChart.Data(i, val[i]));  
236     }  
237 }  
238 }  
239 }  
240 }  
241 }  
242 }  
243 } );  
244  
245 border.setCenter(grid);  
246 Scene scene = new Scene(border, 1200, 700);
```

```
247         return scene;  
248     }  
249 }  
250
```

```

1 package trunk.View;
2
3 import javafx.collections.FXCollections;
4 import javafx.collections.ObservableList;
5 import javafx.event.ActionEvent;
6 import javafx.event.EventHandler;
7 import javafx.geometry.Insets;
8 import javafx.geometry.Orientation;
9 import javafx.geometry.Pos;
10 import javafx.scene.Scene;
11 import javafx.scene.control.*;
12 import javafx.scene.control.cell.PropertyValueFactory;
13 import javafx.scene.layout.BorderPane;
14 import javafx.scene.layout.GridPane;
15 import javafx.scene.layout.VBox;
16 import javafx.scene.text.Font;
17 import javafx.scene.text.Text;
18 import javafx.stage.FileChooser;
19 import javafx.stage.Stage;
20 import trunk.Model.*;
21
22 import java.io.*;
23 import java.util.ArrayList;
24
25 /**
26  * Created by Daniel on 3/25/2016.
27 */
28 public class transactionView {
29
30     /*
31         * Code for saving transaction history
32     */
33     public void saveHistory(Portfolio p){
34         ArrayList<Portfolio> port = null;
35         try {
36             FileInputStream fileIn = new FileInputStream("myfile");
37             ObjectInputStream in = new ObjectInputStream(
38                 fileIn);
39             port = (ArrayList<Portfolio>) in.readObject();
40             in.close();
41             fileIn.close();
42         } catch (ClassNotFoundException ee) {
43             ee.printStackTrace();
44         } catch (FileNotFoundException ee) {
45

```

```

44             ee.printStackTrace();
45         } catch (IOException ee) {
46             ee.printStackTrace();
47         }
48         port.forEach(iter -> {
49             iter.cashAccountHistory = p.cashAccountHistory
50 ;
51             iter.equityHistory = p.equityHistory;
52         });
53
53     try
54     {
55         FileOutputStream fileOut =
56             new FileOutputStream("myfile");
57         ObjectOutputStream out = new
58             ObjectOutputStream(fileOut);
59         out.writeObject(port);
60         out.close();
61         fileOut.close();
62     }catch(IOException i)
63     {
64         i.printStackTrace();
65     }
66     /*
67      * GUI for Transactions page
68     */
69     public Scene Transactions(final Stage primaryStage,
70     Portfolio p, FPTS FPTS) {
71         Text title = new Text(p.loginID + "'s Transaction
72         History");
73         BorderPane border = FPTS.borderPane(primaryStage,
74     p, title);
75         VBox left = new VBox();
76
77         ToolBar toolBar2 = new ToolBar();
78         toolBar2.setOrientation(Orientation.VERTICAL);
79
80         left.getChildren().add(toolBar2);
81         left.setSpacing(10);
82         Label spacer = new Label(" ");
83         Button importCA = new Button("Import Account
84         History");
85         Button importEQ = new Button("Import Equity
86         History");

```

```

82         Button exportCA = new Button("Export Account
83             History");
84         Button exportEQ = new Button("Export Equity
85             History");
86
87         VBox histories = new VBox();
88         VBox eqhistory = new VBox();
89         TableView eqTable = new TableView();
90         eqTable.setEditable(true);
91
92         TableColumn type = new TableColumn("Type");
93         TableColumn action = new TableColumn("Action");
94         TableColumn equity = new TableColumn("Equity");
95         TableColumn date1 = new TableColumn("Date");
96         TableColumn shares = new TableColumn("Amount
97             Transferred");
98
99
100        type.setCellValueFactory(
101            new PropertyValueFactory<History, String>(
102                "type")
103            );
104        action.setCellValueFactory(
105            new PropertyValueFactory<History, String>(
106                "action")
107            );
108        equity.setCellValueFactory(
109            new PropertyValueFactory<History, String>(
110                "equityName")
111            );
112
113        date1.setCellValueFactory(
114            new PropertyValueFactory<History, String>(
115                "date1")
116            );
117        shares.setCellValueFactory(
118            new PropertyValueFactory<History, Double>(
119                "shares")
120            );
121
122
123        ArrayList<Portfolio> port = null;
124        try {
125            FileInputStream fileIn = new FileInputStream(
126                "myfile");
127            ObjectInputStream in = new ObjectInputStream(
128                fileIn);

```

```

117         port = (ArrayList<Portfolio>) in.readObject()
118     ;
119         in.close();
120         fileIn.close();
121     } catch (ClassNotFoundException ee) {
122         ee.printStackTrace();
123     } catch (FileNotFoundException ee) {
124         ee.printStackTrace();
125     } catch (IOException ee) {
126         ee.printStackTrace();
127     }
128     port.forEach(account -> {
129         account.equityHistory = p.equityHistory;
130     });
131
132
133
134
135     for (Portfolio n : port) {
136         temp = n;
137     }
138
139
140     final ObservableList<History> equities =
141         FXCollections.observableArrayList(temp.equityHistory);
142
143     eqTable.setItems(equities);
144     eqTable.getColumns().addAll(type,action,equity,
145         date1,shares);
146
147     final Label label = new Label("Equity History");
148     label.setFont(new Font("Arial", 20));
149     eqhistory.getChildren().addAll(label, eqTable);
150
151
152     VBox cahistory = new VBox();
153     TableView caTable = new TableView();
154     eqTable.setEditable(true);
155
156     TableColumn type2 = new TableColumn("Type");
157     TableColumn action2 = new TableColumn("Action");
158     TableColumn ca = new TableColumn("Cash Account");

```

```

159         TableColumn date2 = new TableColumn("Date");
160         TableColumn amount2 = new TableColumn("Amount
161             Transferred");
162         type2.setCellValueFactory(
163             new PropertyValueFactory<History, String>(
164                 "type")
165             );
166         action2.setCellValueFactory(
167             new PropertyValueFactory<History, String>(
168                 "action")
169             );
170         ca.setCellValueFactory(
171             new PropertyValueFactory<History, String>(
172                 "cashAccName")
173             );
174         date2.setCellValueFactory(
175             new PropertyValueFactory<History, String>(
176                 "date2")
177             );
178         try {
179             FileInputStream fileIn = new FileInputStream(
180                 "myfile");
181             ObjectInputStream in = new ObjectInputStream(
182                 fileIn);
183             port = (ArrayList<Portfolio>) in.readObject()
184             ;
185             in.close();
186             fileIn.close();
187         } catch (ClassNotFoundException ee) {
188             ee.printStackTrace();
189         } catch (FileNotFoundException ee) {
190             ee.printStackTrace();
191         } catch (IOException ee) {
192             ee.printStackTrace();
193         }
194         port.forEach(account -> {
195             account.cashAccountHistory = p.
196             cashAccountHistory;
197         });

```

```

194
195
196     final ObservableList<History> cas = FXCollections
197         .observableArrayList(p.cashAccountHistory);
198         caTable.setItems(cas);
199         caTable.getColumns().addAll(type2,action2,ca,
200             amount2,date2);
201         type2.prefWidthProperty().bind(caTable.
202             widthProperty().multiply(0.1));
203         action2.prefWidthProperty().bind(caTable.
204             widthProperty().multiply(0.1));
205         ca.prefWidthProperty().bind(caTable.widthProperty
206             ().multiply(0.30));
207         final Label label2 = new Label("Cash Account
208             History");
209         label2.setFont(new Font("Arial", 20));
210         cahistory.getChildren().addAll(label2, caTable);
211
212         histories.getChildren().addAll(eqhistory,
213             cahistory);
214         toolBar2.getItems().addAll(spacer, importCA,
215             importEQ, exportCA, exportEQ);
216
217         importCA.setOnAction(new EventHandler<ActionEvent
218             >() {
219             @Override
220             public void handle(ActionEvent event) {
221                 String csv = null;
222                 FileChooser fileChooser = new FileChooser
223                     ();
224                     fileChooser.setTitle("Open Resource File"
225                     );
226                     File file = fileChooser.showOpenDialog(
227                         FPTS.stage);
228                     if(file != null){
229                         Equity e = null;
230                         History h = new History("", "", e, "", 0,
231                             0);
232                         h.importCAHistory(file.toString(), p);
233                         Scene scene = Transactions(
234                             primaryStage, p, FPTS);
235                             FPTS.stage.setScene(scene);
236                         }
237                     }
238                 );

```

```

225         importEQ.setOnAction(new EventHandler<ActionEvent>() {
226             @Override
227             public void handle(ActionEvent event) {
228                 String csv = null;
229                 FileChooser fileChooser = new FileChooser();
230                 fileChooser.setTitle("Open Resource File");
231                 File file = fileChooser.showOpenDialog(FPTS.stage);
232                 if(file != null) {
233                     Equity e = null;
234                     History h = new History("", "", e, "", 0, 0);
235                     h.importEQHistory(file.toString(), p);
236                     Scene scene = Transactions(primaryStage, p, FPTS);
237                     FPTS.stage.setScene(scene);
238                 }
239             }
240         });
241         //Action for exporting equities
242         exportEQ.setOnAction(new EventHandler<ActionEvent>() {
243             @Override
244             public void handle(ActionEvent event) {
245                 final Stage dialog = new Stage();
246                 dialog.initOwner(primaryStage);
247                 GridPane grid2 = new GridPane();
248                 grid2.setAlignment(Pos.CENTER);
249                 grid2.setHgap(10);
250                 grid2.setVgap(10);
251                 grid2.setPadding(new Insets(5, 10, 5, 10));
252
253                 TextField file = new TextField();
254                 grid2.add(new Label("Filename: "), 0, 1);
255                 grid2.add(file, 1, 1);
256
257                 Scene dialogScene = new Scene(grid2, 400, 400);
258                 dialog.setScene(dialogScene);
259                 dialog.show();
260                 Button cancel = new Button("Cancel");

```

```

261                     grid2.add(cancel,1,7);
262                     Button submit = new Button("Submit");
263                     grid2.add(submit,0,7);
264                     cancel.setOnAction(new EventHandler<
265                         ActionEvent>() {
266                             @Override
267                             public void handle(ActionEvent e) {
268                                 dialog.close();
269                             }
270                         submit.setOnAction(new EventHandler<
271                             ActionEvent>() {
272                                 @Override
273                                 public void handle(ActionEvent event)
274                                 {
275                                     Equity e = null;
276                                     History h = new History("", "", e,
277                                     "", 0, 0);
278                                     h.exportEquityHistory(file.
279                                     getText(), p);
280                                     dialog.close();
281                                 }
282                             });
283                         //Action for exporting a cash account
284                         exportCA.setOnAction(new EventHandler<ActionEvent
285 >() {
286                             @Override
287                             public void handle(ActionEvent event) {
288                                 final Stage dialog = new Stage();
289                                 dialog.initOwner(primaryStage);
290                                 GridPane grid2 = new GridPane();
291                                 grid2.setAlignment(Pos.CENTER);
292                                 grid2.setHgap(10);
293                                 grid2.setVgap(10);
294                                 grid2.setPadding(new Insets(5, 10, 5, 10));
295                                 TextField file = new TextField();
296                                 grid2.add(new Label("Filename: "), 0, 1);
297                                 grid2.add(file, 1, 1);
298                                 Scene dialogScene = new Scene(grid2, 400,
299                                 400);

```

```
298                     dialog.setScene(dialogScene) ;  
299                     dialog.show() ;  
300                     Button cancel = new Button("Cancel") ;  
301                     grid2.add(cancel,1,7) ;  
302                     Button submit = new Button("Submit") ;  
303                     grid2.add(submit,0,7) ;  
304                     cancel.setOnAction(new EventHandler<  
    ActionEvent>() {  
305                         @Override  
306                         public void handle(ActionEvent e) {  
307                             dialog.close() ;  
308                         }  
309                     }) ;  
310                     submit.setOnAction(new EventHandler<  
    ActionEvent>() {  
311                         @Override  
312                         public void handle(ActionEvent event)  
{  
313                             Equity e = null ;  
314                             History h = new History("", "", e,  
    "", 0, 0) ;  
315                             h.exportCAHistory(file.getText(),  
    p) ;  
316                             dialog.close() ;  
317                         }  
318                     }) ;  
319                 }  
320             } ;  
321             border.setCenter(histories) ;  
322             border.setLeft(left) ;  
323             Scene scene = new Scene(border, 1200, 700) ;  
324             return scene ;  
325         }  
326     }  
327 }
```

```

1 package trunk.Model;
2
3 import trunk.Control.HistoryInvoker;
4 import trunk.Control.UpdateEQ;
5
6 import java.io.*;
7 import java.lang.reflect.Array;
8 import java.text.DateFormat;
9 import java.text.ParseException;
10 import java.text.SimpleDateFormat;
11 import java.util.ArrayList;
12 import java.util.Date;
13 import java.util.Dictionary;
14 import java.util.HashMap;
15
16 /**
17 * A class to represent an instance of an Equity.
18 * An Equity object represents various real-life equities
19 * including stock, bonds, mutual funds, etc.
20 */
21
22 public class Equity implements Serializable {
23     public String tickerSymbol;
24     public String name;
25     public ArrayList<String> marketAverage;
26     public String MA;
27
28     public int shares;
29     public double acquisitionPrice;
30     public String acquisitionDate;
31
32 /**
33 * Constructor for an Equity object.
34 * @param tickerSymbol: A String representing the
35 market symbol associated
36 * with this equity.
37 * @param name : A string representing the name of the
38 equity.
39 * @param shares : An int representing the number of
40 shares of this equity owned.
41 * @param acquisitionPrice : A double representing the
42 orginal price paid for
43 * this equity, per share.
44 * @param acquisitionDate : A String representing the
45 date this equity was purchased.

```

```

41      * param marketAverage : An ArrayList representing
42      * the index the equity belongs to.
43      */
44      public Equity(String tickerSymbol, String name, int
45      shares, double acquisitionPrice,
46      String acquisitionDate, ArrayList<String>
47      > marketAverage) {
48
49          this.tickerSymbol = tickerSymbol;
50          this.name = name;
51          this.shares = shares;
52          this.acquisitionPrice = acquisitionPrice;
53          this.acquisitionDate = acquisitionDate;
54          this.marketAverage = marketAverage;
55          if(marketAverage != null) {
56              this.MA = marketAverage.toString();
57          }
58
59      /**
60      * A getter for the ticker/market symbol associated
61      * with this equity.
62      * return : A string representing the market symbol.
63      */
64      public String getTickerSymbol() {
65          return this.tickerSymbol;
66      }
67
68      /**
69      * A getter for the name of this equity.
70      * return : A string representing the equity's name.
71      */
72      public String getName() {
73          return this.name;
74      }
75
76      /**
77      * A getter for the market averages this equity
78      * belongs to.
79      * return : An ArrayList of the sector index.
80      */
81      public ArrayList<String> getMarketAverage() {return
82          this.marketAverage; }
83
84      /**

```

```

80      * A getter for the number of shares of this equity
81      * @return : A int representing the number of shares.
82      */
83      public int getShares(){
84          return this.shares;
85      }
86
87      /**
88      * A getter for the original price paid for this
89      * @return : A double representing the acquisition
90      * price.
91      */
92      public double getAcquisitionPrice(){
93          return this.acquisitionPrice;
94      }
95
96      /**
97      * A getter for the date this equity was purchased.
98      * @return : A string representing the acquisition
99      * date.
100     */
101
102
103     public String getAcquisitionDate(){
104         return this.acquisitionDate;
105     }
106
107     /**
108     * A getter for the total value of this equity.
109     * @return : A double that consists of the
110     * acquisition price multiplied by the
111     * number of shares of this equity owned.
112     */
113     public double getTotal(){
114         return Math.round((acquisitionPrice * shares) *
115         100.0) / 100.0;
116     }
117
118     /**
119      * Import equities
120      */

```

```

119     public Equity importEQ(String csv, Portfolio p) {
120         BufferedReader br = null;
121         String line = "";
122         String[] equity = null;
123         Equity e = new Equity("N/A", "N/A", 0, 0, null, null);
124
125
126         try {
127             br = new BufferedReader(new FileReader(csv));
128             while ((line = br.readLine()) != null) {
129                 equity = line.split(",(?=([^\"]*\"[^\"]*"
129                  \")*[^\""]*$)", -1);
130                 if(equity[0].equals("equity")){
131                     if (equity.length == 6) {
132                         ArrayList<String>
133                         marketAverageList = new ArrayList<String>();
134                         if(equity[2].contains("^")){
134                             //e.tickerSymbol = equity[2].
135                             substring(0, equity[2].lastIndexOf("^")).replaceAll
135                             ("^\"|\"$", ""));
136                             e.tickerSymbol = equity[2].
136                             substring(0, equity[2].lastIndexOf("^")).replaceAll("[^=,
136                             \\\da-zA-Z]", "");
137                         }else{
137                             e.tickerSymbol = equity[2].
137                             replaceAll("^\\s+|\\s+$", "").replaceAll("[^=,\\\da-zA-Z]"
137                             , "");
138                         }
139                         e.shares = Integer.parseInt(
139                           equity[1].replaceAll("^\"|\"$", ""));
140                         e.acquisitionPrice = Double.
140                           parseDouble(equity[3].replace("\\"", "").replaceAll("^\"|
140                           \"$", ""));
141                         e.name = equity[4];
142                         marketAverageList.add(equity[5].
142                           replaceAll("^\"|\"$", ""));
143                         e.marketAverage =
143                           marketAverageList;
144                         e.MA = e.marketAverage.toString()
144                           .replace("[", "").replace("]", "");
145                         Date date = null;
146                         SimpleDateFormat format = new
146                           SimpleDateFormat("dd/MM/yyyy");
147                         try {
148                             date = format.parse("00/00/

```

```

148 0000");
149
150 } catch (ParseException ee) {
151     ee.printStackTrace();
152 }
153     e.addEquity(e.tickerSymbol, e.
154         name, e.shares, e.acquisitionPrice, "N/A", e.
155         marketAverage, true, p);
156     } else if (equity.length > 6) {
157         ArrayList<String>
158         marketAverageList = new ArrayList<String>();
159
160         if (equity[2].contains("^")){
161             e.tickerSymbol = equity[2].
162                 substring(0, equity[2].lastIndexOf("^")).replaceAll("[^=,
163 \da-zA-Z]", "");
164         }else{
165             e.tickerSymbol = equity[2].
166                 replaceAll("^\\s+|\\s+$", "").replaceAll("[^=,\da-zA-Z]" ,
167 "", );
168         }
169
170         e.shares = Integer.parseInt(
171             equity[1].replaceAll("^|"|"$", ""));
172         e.acquisitionPrice = Double.
173             parseDouble(equity[3].replace("\", "").replaceAll("^|"|
174 "\$", ""));
175         e.name = equity[4];
176         int x = 5;
177         while (x < equity.length && !
178             equity[x].equals("")) {
179             marketAverageList.add(equity[
180                 x].replaceAll("^|"|"$", ""));
181             x++;
182         }
183         e.marketAverage =
184             marketAverageList;
185         e.MA = e.marketAverage.toString()
186             .replace("[", "").replace("]", "");
187         e.addEquity(e.tickerSymbol, e.
188             name, e.shares, e.acquisitionPrice, "N/A", e.
189             marketAverage, true, p);
190     }
191 }
192 else if(equity[0].equals("marketAverage"))
193 {

```

```

176                     Equity c = new Equity("", "", 0, 0, "",  

177                         null);  

178                     c.MA = equity[1];  

179                     c.shares = Integer.parseInt(equity[2]  

180                         );  

181                     c.acquisitionPrice = Double.  

182                         parseDouble(equity[3]);  

183                     p.marketAverages.add(c);  

184                     }  

185                     else {  

186                         if (equity.length == 4) {  

187                             ArrayList<String>  

188                             marketAverageList = new ArrayList<String>();  

189                             if (equity[0].contains("^")){  

190                                 e.tickerSymbol = equity[0].  

191                                 substring(0, equity[0].lastIndexOf("^")).replaceAll("[^=,  

192                                     \\da-zA-Z]", "");  

193                                 }  

194                                 e.name = equity[1].replaceAll("^  

195                                     |\"$\", "");  

196                                 e.acquisitionPrice = Double.  

197                                 parseDouble(equity[2].replace("\\"", "")).replaceAll("^\\|  

198                                     \"$\", "");  

199                                 marketAverageList.add(equity[3].  

200                                     replaceAll("^\\|\"$\", ""));  

201                                 e.marketAverage =  

202                                     marketAverageList;  

203                                 e.MA = e.marketAverage.toString()  

204                                     .replace("[", "").replace("]", "");  

205                                 Date date = null;  

206                                 SimpleDateFormat format = new  

207                                     SimpleDateFormat("dd/MM/yyyy");  

208                                 try {  

209                                     date = format.parse("00/00/  

210                                         0000");  

211                                 } catch (ParseException ee) {  

212                                     ee.printStackTrace();  

213                                 }  

214                                 addEquity(e.tickerSymbol, e.name,  

215                                     0, e.acquisitionPrice, "N/A", e.marketAverage, true, p);  

216                                 } else if (equity.length > 4) {  


```

```

204                     ArrayList<String>
205             marketAverageList = new ArrayList<String>();
206             if (equity[0].contains("^")){
207                 e.tickerSymbol = equity[0].
208                 substring(0, equity[0].lastIndexOf("^")).replaceAll("[^=,
209                 \da-zA-Z]", " ");
210             } else{
211                 e.tickerSymbol = equity[0].
212                 replaceAll("^\\s+|\\s+$", "").replaceAll("[^=,\da-zA-Z]" ,
213                 " ");
214             }
215             e.name = equity[1].replaceAll("^
216             \"|$", " ");
217             e.acquisitionPrice = Double.
218             parseDouble(equity[2].replace("\"", "").replaceAll("^\"|
219             \"$", ""));
220             int x = 3;
221             while (x < equity.length && !
222                 equity[x].equals("")){
223                 marketAverageList.add(equity[
224                     x].replaceAll("^\"|$", ""));
225                 x++;
226             }
227             e.marketAverage =
228                 marketAverageList;
229             e.MA = e.marketAverage.toString()
230             .replace("[", "").replace("]", "");
231             addEquity(e.tickerSymbol, e.name,
232                 e.shares, e.acquisitionPrice, "N/A", e.marketAverage,
233                 true, p);
234         }
235     }
236 }
237
238 } catch (FileNotFoundException ee) {
239     ee.printStackTrace();
240 } catch (IOException ee) {
241     ee.printStackTrace();
242 } finally {
243     if (br != null) {
244         try {
245             br.close();
246         } catch (IOException ee) {
247             ee.printStackTrace();
248         }
249     }
250 }
```

```

235         }
236     }
237
238     return e;
239 }
240
241 /**
242 * Export equities.
243 */
244 public void exportEQ(String eFileName, Portfolio p) {
245     try{
246         FileWriter writer = new FileWriter(eFileName)
247 ;
248         for (Equity equity : p.equityList1){
249             writer.append("equity" + ',');
250             writer.append(equity.shares + "");
251             writer.append(',');
252             writer.append(equity.tickerSymbol);
253             writer.append(',');
254             writer.append(equity.getTotal() + "");
255             writer.append(',');
256             writer.append(equity.name);
257             writer.append(",");
258             equity.marketAverage.forEach((SI) -> {
259                 try {
260                     writer.append(SI.toString());
261                     writer.append(",");
262                 }
263                 catch (IOException IO) {
264                     IO.printStackTrace();
265                 } catch (NullPointerException e) {
266                     e.printStackTrace();
267                 }
268             });
269
270             writer.append('\n');
271         }
272         for(Equity ma : p.marketAverages) {
273             writer.append("marketAverage");
274             writer.append(',');
275             writer.append(ma.MA.replace("[", ""));
276             replace("]", ""));
277             writer.append(',');
278             writer.append(Integer.toString(ma.shares))

```

```

277 );
278         writer.append(',');
279         writer.append(Double.toString(ma.
280             acquisitionPrice));
280         writer.append("\n");
281     }
282
283     writer.flush();
284     writer.close();
285
286 } catch (IOException e) {
287     e.printStackTrace();
288 }
289
290 /**
291 * Add an equity
292 */
293
294 public void addEquity(String tickerSymbol, String
295   name, int numShares, double salePrice,
296   String saleDate, ArrayList
297   marketAverage, Boolean imported, Portfolio p) {
298   Boolean exists = false;
299   for(Equity equity : p.equityList1){
300     if(!tickerSymbol.equals("")) {
301       if (equity.tickerSymbol.equals(
302         tickerSymbol)) {
303         exists = true;
304       }
305     } else{
306       if(equity.marketAverage.get(0).equals(
307         marketAverage.get(0))){
308         exists = true;
309       }
310     }
311     if(!exists) {
312       Equity e = new Equity("", "", 0, 0, null,
313         null);
314       e.tickerSymbol = tickerSymbol;
315       e.name = name;
316       e.shares = numShares;

```

```
316         e.acquisitionPrice = salePrice;
317         e.acquisitionDate = saleDate;
318         e.marketAverage = marketAverage;
319         e.MA = marketAverage.toString().replace("[",
320             "").replace("]", ""));
321
322         if (imported && numShares > 0) {
323             p.equityList2.add(e);
324             p.equityList1.add(e);
325         } else if (imported && numShares == 0) {
326             p.equityList2.add(e);
327         } else {
328             p.equityList1.add(e);
329         }
330
331         for (String avg : e.MA.split(",")) {
332             ArrayList<String> avgArray = new
333                 ArrayList<>();
334             avg = avg.trim();
335             avgArray.add(avg);
336             Equity marketAvg = new Equity("", "", 0,
337                 salePrice, null, avgArray);
338             boolean exist = true;
339             for (Equity equity : p.marketAverages) {
340                 if (equity.MA.equals(avg)) {
341                     exist = false;
342                 }
343             }
344             if (exist == false) {
345                 p.marketAverages.add(marketAvg);
346             } else {
347                 double price = 0;
348                 int count = 0;
349                 ArrayList<Integer> shares = new
350                     ArrayList<>();
351
352                 for (Equity q : p.equityList2) {
353                     if (q.marketAverage.contains(avg)) {
354                         price += q.acquisitionPrice;
355                         count += 1;
356                         shares.add(q.shares);
357                     }
358                 }
359                 for (Equity q : p.equityList1) {
```

```

356                     if(q.marketAverage.contains(avg) )
357                     {
358                         shares.add(q.shares);
359                     }
360                     p.marketAverages.remove(marketAvg);
361                     int totalshares = 0;
362                     for(int s : shares) {
363                         totalshares += s;
364                     }
365                     marketAvg.shares = totalshares;
366                     if(!avg.equals("DOW")) {
367                         marketAvg.acquisitionPrice = Math
368                         .round((price / count) * 100.0) / 100.0;
369                     } else{
370                         marketAvg.acquisitionPrice = Math
371                         .round(((price / count) / .14602128057775) * 100.0) / 100
372                         .0;
373                     }
374                     p.marketAverages.add(marketAvg);
375                 }
376             }
377             if(imported == false) {
378                 DateFormat dateFormat = new
379                 SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
380                 Date date = new Date();
381                 HistoryInvoker test = new HistoryInvoker(
382                 );
383                 test.setLog(new LogEquityHistory(), p,
384                 new History("Add", "EQUITY", e, dateFormat.format(date),
385                 0, numShares));
386                 test.logAction();
387             }
388         }
389         else{
390             Equity e = null;
391             double amount = 0;
392             for(Equity equity : p.equityList1){
393                 if(!tickerSymbol.equals("")) {
394                     if (equity.tickerSymbol.equals(
395                         tickerSymbol)) {
396                         equity.shares += numShares;
397                         e = equity;
398                     }
399                 }
400             }
401         }
402     }
403 
```

```

392                     amount = numShares * equity.
393             getAcquisitionPrice();
394         }
394     } else{
395         if(equity.marketAverage.get(0).equals
395 (marketAverage.get(0))) {
396             equity.shares += numShares;
397             e = equity;
398             amount = numShares * equity.
399             getAcquisitionPrice();
400         }
401     }
402     DateFormat dateFormat = new SimpleDateFormat(
402 "yyyy/MM/dd HH:mm:ss");
403     Date date = new Date();
404     HistoryInvoker test = new HistoryInvoker();
405     test.setLog(new LogEquityHistory(), p, new
405 History("Add Shares", "EQUITY", e, dateFormat.format(date)
405 , amount, numShares));
406     test.logAction();
407 }
408 }
409 /**
410  * Remove Equity based on symbol given.
411  */
413 public void removeEquity(String symbol, String
413 cashAcct, Portfolio p) {
414     ArrayList<Equity> copyList = new ArrayList<>(p.
414 equityList1);
415
416     for(Equity equity : copyList) {
417         double amount = 0;
418         double shares = 0;
419         if (equity.tickerSymbol.equals(symbol)) {
420             if (!cashAcct.equals("None")) {
421                 CashAccount c = new CashAccount("", ""
421 , 0, "", p);
422                 amount = -equity.getTotal();
423                 shares = -equity.shares;
424                 c.updateBalance(cashAcct, equity.
424 getTotal(), true, p);
425             }
426             p.equityList1.remove(equity);

```

```
427
428             DateFormat dateFormat = new
429             SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
430
431             HistoryInvoker test = new HistoryInvoker(
432             );
433             test.setLog(new LogEquityHistory(), p,
434             new History("Remove", "EQUITY", equity,dateFormat.format(
435             date),amount,shares));
436             test.logAction();
437
438     @Override
439     public boolean equals(Object obj) {
440         if(obj instanceof Equity){
441             if(this.tickerSymbol.equals(((Equity) obj) .
442             tickerSymbol)){
443                 if(this.name.equals(((Equity) obj).name))
444                     return true;
445             }
446         }
447     }
448
449     return false;
450 }
451 }
```

```

1 package trunk.Model;
2
3
4 import trunk.Model.*;
5
6 import java.io.*;
7
8
9 /**
10  * A class to represent an instance of an event or
11  * transaction
12  * that should be logged to history.
13  */
14
15
16 public class History implements Serializable{
17     String type;
18     String action;
19
20     Equity equity;
21     String equityName;
22
23     CashAccount cashAccount;
24     String cashAccName;
25     Double transferAmount1;
26     Double transferAmount2;
27
28     Double shares;
29
30     String date1;
31     String date2;
32     /**
33      * Constructor for a History Object for a event
34      * related to an Equity.
35      * @param action : A String representing the action. (
36      * e.g. add, remove, etc.)
37      * @param type : A String representing which history
38      * type referenced. (e.g. "Equity")
39      * @param equity : The Equity involved in the action.
40      */
41
42     public History(String action, String type, Equity
43     equity, String date, double transferAmount, double shares)
44     {
45         this.type = type;
46         this.action = action;

```

```

40         this.equityName = equity.name;
41         this.date1 = date;
42         this.transferAmount1 = transferAmount;
43         this.shares = shares;
44     }
45
46     /**
47      * Constructor for a History Object for an event
48      * related to Cash Accounts.
49      * @param action : A String representing the action. (e.g. add, remove, etc.)
50      * @param type : A String representing which history type referenced. (e.g. "Cash Account")
51      * @param cashAccount : The Cash Account involved in the action.
52      */
53     public History(String action, String type, CashAccount
54     cashAccount, String date, double transferAmount) {
55         this.type = type;
56         this.action = action;
57         this.cashAccName = cashAccount.name;
58         this.date2 = date;
59         this.transferAmount2 = transferAmount;
60     }
61
62     /**
63      * A getter for the history type referenced.
64      * @return : A string representing this type.
65      */
66     public String getType() {
67         return this.type;
68     }
69
70     /**
71      * A getter for the action associated with the event
72      * that occurred.
73      * @return : A string representing this action.
74      */
75     public String getAction() {
76         return this.action;
77     }
78
79     /**
80      * A getter for the name of the Equity involved in an
81      * event.

```

```

78      * @return : A string representing the name.
79      */
80      public String getEquityName() {
81          return this.equityName;
82      }
83
84      /**
85      * A getter for the name of the Cash Account involved
86      * in an event.
87      * @return : A string representing the name.
88      */
89      public String getCashAccName() {
90          return this.cashAccName;
91      }
92
93      public Double getTransferAmount1(){return this.
94      transferAmount1;}
95
96      public Double getTransferAmount2(){return this.
97      transferAmount2;}
98
99      public String getDate1(){return this.date1;}
100
101     /*
102     * Import cash account history
103     */
104     public void importCAHistory(String csv, Portfolio p)
105     {
106         BufferedReader br = null;
107         String line = "";
108         String[] ca = null;
109         CashAccount c = new CashAccount("", "", 0, "", null);
110         //History caH = new History("N/A", "N/A", c);
111
112         try {
113
114             br = new BufferedReader(new FileReader(csv));
115             while ((line = br.readLine()) != null) {
116                 History caH = new History("N/A", "N/A", c
117                 , "", 0);

```

```

118             ca = line.split(", (?=([^\"]*\"[^\"]*\"
119                           )*[^\"]*$)", -1);;
120
121             caH.type = ca[2];
122             caH.action = ca[1];
123             caH.cashAccName = ca[3];
124
125             p.cashAccountHistory.add(caH);
126
127         } catch (FileNotFoundException ee) {
128             ee.printStackTrace();
129         } catch (IOException ee) {
130             ee.printStackTrace();
131         } finally {
132             if (br != null) {
133                 try {
134                     br.close();
135                 } catch (IOException ee) {
136                     ee.printStackTrace();
137                 }
138             }
139         }
140     }
141 }
142 /*
143 * Import equity history
144 */
145
146 public void importEQHistory(String csv, Portfolio p)
{
147     BufferedReader br = null;
148     String line = "";
149     String[] ca = null;
150     Equity c = new Equity("", "", 0, 0, "", null);
151
152     try {
153
154         br = new BufferedReader(new FileReader(csv));
155         while ((line = br.readLine()) != null) {
156             History caH = new History("N/A", "N/A", c,
157                                         "", 0, 0);
158             ca = line.split(", (?=([^\"]*\"[^\"]*\"
159                           )*[^\"]*$)", -1);;

```

```

159                     caH.type = ca[2];
160                     caH.action = ca[1];
161                     caH.equityName = ca[3];
162
163                     p.equityHistory.add(caH);
164                 }
165
166             } catch (FileNotFoundException ee) {
167                 ee.printStackTrace();
168             } catch (IOException ee) {
169                 ee.printStackTrace();
170             } finally {
171                 if (br != null) {
172                     try {
173                         br.close();
174                     } catch (IOException ee) {
175                         ee.printStackTrace();
176                     }
177                 }
178             }
179
180         }
181
182     /**
183      * Export Cash Account History
184      * Export example : 'Add', 'CashAccount', '
185      * NameOfAccount'
186      */
187     public void exportCAHistory(String eFileName,
188         Portfolio p){
189         try{
190             FileWriter writer = new FileWriter(eFileName)
191             ;
192             for (History h : p.cashAccountHistory) {
193                 writer.append("transaction");
194                 writer.append(",");
195                 writer.append(h.action);
196                 writer.append(',');
197                 writer.append(h.type);
198                 writer.append(',');
199                 writer.append(h.cashAccName);
200                 writer.append("\n");
201             }
202         }
203     }

```

```
201             writer.flush();
202             writer.close();
203
204         }catch (IOException e) {
205             e.printStackTrace();
206         }
207     }
208
209     /**
210      * Export equity History
211      * Export example: 'Remove', 'Equity', 'NameOfEquity'
212      */
213     public void exportEquityHistory(String eFileName,
214     Portfolio p) {
215         try{
216             FileWriter writer = new FileWriter(eFileName)
217             ;
218             for (History h : p.equityHistory) {
219                 writer.append("transaction");
220                 writer.append(",");
221                 writer.append(h.action);
222                 writer.append(',');
223                 writer.append(h.type);
224                 writer.append(',');
225                 writer.append(h.equityName);
226                 writer.append("\n");
227             }
228             writer.flush();
229             writer.close();
230
231         }catch (IOException e) {
232             e.printStackTrace();
233         }
234     }
235
236 }
237
```

```

1 package trunk.Model;
2
3 //import trunk.Model.*;
4
5
6
7 import trunk.Control.UpdateEQ;
8
9 import java.io.*;
10 import java.text.ParseException;
11 import java.text.SimpleDateFormat;
12 import java.util.ArrayList;
13 import java.util.Date;
14 import java.util.List;
15 import java.util.stream.Collectors;
16
17
18 /**
19  * Class representation of the Financial Portfolio
20 */
21 public class Portfolio implements Serializable{
22     public String loginID; // Login ID of portfolio user
23     public String password; // Password associated with
        loginID of user
24     public ArrayList<Equity> equityList1 = new ArrayList<>()
        (); //List of owned equities
25     public ArrayList<Equity> equityList2 = new ArrayList<>()
        (); //List of available equities
26     public ArrayList<Equity> marketAverages = new
        ArrayList<>(); //List of market averages
27     public ArrayList<Watchlist> watchlists = new ArrayList
        <>(); //List of watchlist items
28
29     public ArrayList<CashAccount> cashAccList = new
        ArrayList<>();
30     public ArrayList<History> cashAccountHistory = new
        ArrayList<>();
31     public ArrayList<History> equityHistory = new
        ArrayList<>();
32
33
34     public void initialEquities(){
35         BufferedReader br = null;
36         String line = "";
37         String[] equity = null;

```

```

38         ArrayList<Equity> equities = new ArrayList<>();
39         try {
40
41             br = new BufferedReader(new FileReader("svn/
equities.csv"));
42             while ((line = br.readLine()) != null) {
43                 equity = line.split(",(?=([^\"]*\"[^\"]*\""
44                 )*[^\"]*$)", -1);
45                 if (equity.length == 4) {
46                     Equity e = new Equity("N/A", "N/A", 0, 0,
null,null);
47                     ArrayList<String> marketAverageList =
48                     new ArrayList<String>();
49                     if (equity[0].contains("^")){
50                         e.tickerSymbol = equity[0].
51                         substring(0, equity[0].lastIndexOf("^")).replaceAll("[^=,
52                         \da-zA-Z]", " ");
53                         }else{
54                             e.tickerSymbol = equity[0].
55                             replaceAll("[^=,\da-zA-Z]", " ");
56                         }
57                         e.name = equity[1].replaceAll("^\"|\\"$",
58                         " ");
59                         e.acquisitionPrice = Double.
60                         parseDouble(equity[2].replace("\\"", "") .
61                         replaceAll("^\"|\\"$", " "));
62                         marketAverageList.add(equity[3] .
63                         replaceAll("^\"|\\"$", " "));
64                         e.marketAverage = marketAverageList;
65
66                         e.MA = e.marketAverage.toString() .
67                         replace("[", "").replace("]", "");
68                         Date date = null;
69                         SimpleDateFormat format = new
70                         SimpleDateFormat("dd/MM/yyyy");
71                         try {
72                             date = format.parse("00/00/0000");
73                         } catch (ParseException ee) {
74                             ee.printStackTrace();
75                         }
76                         e.addEquity(e.tickerSymbol, e.name, 0,
77                         e.acquisitionPrice, "N/A", e.marketAverage, true, this);
78                     } else if (equity.length > 4) {
79                         Equity e = new Equity("N/A", "N/A", 0, 0,
80                         
```

```

68 null,null);
69                     ArrayList<String> marketAverageList =
70                     new ArrayList<String>();
71                     if (equity[0].contains("^")){
72                         e.tickerSymbol = equity[0].
73                         substring(0, equity[0].lastIndexOf("^")).replaceAll("[^=,
74                           \da-zA-Z]", " ");
75                     }else{
76                         e.tickerSymbol = equity[0].
77                         replaceAll("[^=,\da-zA-Z]", " ");
78                     }
79                     e.name = equity[1].replaceAll("^\"|\\"
80                         $", " ");
81                     e.acquisitionPrice = Double.
82                     parseDouble(equity[2].replace("\\"", "").replaceAll("^\"|
83                         \"$", ""));
84                     int x = 3;
85                     while (x < equity.length && !equity[x
86                         ].equals("")) {
87                         marketAverageList.add(equity[x].
88                         replaceAll("^\"|\\"$", ""));
89                         x++;
90                     }
91                     e.marketAverage = marketAverageList;
92
93                     e.MA = e.marketAverage.toString().
94                     replace("[", "").replace("]", "");
95                     e.addEquity(e.tickerSymbol, e.name, 0
96                     , e.acquisitionPrice, "N/A", e.marketAverage, true, this)
97                     ;
98
99
100                }

```

```

101         }
102
103     }
104
105     public Portfolio(String loginID, String password) {
106         this.loginID = loginID;
107         this.password = password;
108         this.equityList1 = new ArrayList<>();
109         this.equityList2 = new ArrayList<>();
110         initialEquities();
111         try{
112             UpdateEQ updateEQ = new UpdateEQ();
113             updateEQ.updateEquities(this, this.
114             equityList2);
115             }catch(Exception ex) {
116                 ex.getMessage();
117             }
118             this.cashAccList = new ArrayList<>();
119         }
120
121         public Portfolio(ArrayList<Equity> ownedE, ArrayList<
122             CashAccount> ownedCA) {
123             this.equityList1 = ownedE;
124             this.cashAccList = ownedCA;
125             //this.cashAccountHistory = new ArrayList<>();
126             //this.equityHistory = new ArrayList<>();
127         }
128
129         public void importP(String eFileName, Portfolio p) {
130             BufferedReader br = null;
131             String line = "";
132             String[] string = null;
133             Equity e = new Equity("N/A", "N/A", 0, 0, null,
134             null);
135
136             try {
137                 br = new BufferedReader(new FileReader(
138                     eFileName));
139                 while ((line = br.readLine()) != null) {
140                     string = line.split(",(?=([^\\"]*\"[^\\"]*"
141                     \"*)*[^\"]*$)", -1);
142
143                     if (string[0].equals("equity")) {
144                         if (string.length == 6) {
145                             ArrayList<String>
146                             marketAverageList = new ArrayList<String>();

```

```

140                               e.tickerSymbol = string[2].  

141                               replaceAll("^\"|\"$", "");  

142                               e.shares = Integer.parseInt(  

143                               string[1].replaceAll("^\"|\"$", ""));  

144                               e.acquisitionPrice = (Double.  

145                               parseDouble(string[3].replace("\\"", "").replaceAll("^\"|  

146                               \"$", ""))) / e.shares;  

147                               e.name = string[4];  

148                               marketAverageList.add(string[5].  

149                               replaceAll("^\"|\"$", ""));  

150                               e.marketAverage =  

151                               marketAverageList;  

152                               e.MA = e.marketAverage.toString()  

153                               .replace("[", "").replace("]", "");  

154                               Date date = null;  

155                               SimpleDateFormat format = new  

156                               SimpleDateFormat("dd/MM/yyyy");  

157                               try {  

158                                   date = format.parse("00/00/  

159                                   0000");  

160                               } catch (ParseException ee) {  

161                                   ee.printStackTrace();  

162                               }  

163                               e.addEquity(e.tickerSymbol, e.  

164                               name, e.shares, e.acquisitionPrice, "N/A", e.  

165                               marketAverage, false, p);  

166                               } else if (string.length > 6) {  

167                               ArrayList<String>  

168                               marketAverageList = new ArrayList<String>();  

169                               e.tickerSymbol = string[2].  

170                               replaceAll("^\"|\"$", "");  

171                               e.shares = Integer.parseInt(  

172                               string[1].replaceAll("^\"|\"$", ""));  

173                               e.acquisitionPrice = (Double.  

174                               parseDouble(string[3].replace("\\"", "").replaceAll("^\"|  

175                               \"$", ""))) / e.shares;  

176                               if (string[5].contains("Inc")){  

177                                   e.name = string[4] += ", Inc  

178                               .";  

179                               }  

180                               }  

181                               e.name = string[4];  

182                               }  

183                               int x = 5;  

184                               while (x < string.length && !

```

```

167 string[x].equals(""))) {
168                                     if (!string[x].contains("Inc"))
169 ) {
170                                     marketAverageList.add(
171                                     string[x].replaceAll("^\"|\"$", ""));
172                                     x++;
173                                     } else{
174                                     x++;
175                                     }
176                                     e.marketAverage =
177                                     marketAverageList;
178                                     e.MA = e.marketAverage.toString()
179                                     .replace("[", "").replace("]", "");
180                                     e.addEquity(e.tickerSymbol, e.
181                                     name, e.shares, e.acquisitionPrice, "N/A", e.
182                                     marketAverage, false, p);
183                                     }
184                                     } else if (string[0].equals("cashacc")) {
185                                     CashAccount c = new CashAccount("N/A"
186                                     , "N/A", 0.0, "", null);
187                                     c.name = string[2];
188                                     c.balance = Double.parseDouble(string
189                                     [1].replace("\\\"", ""));
190                                     //c.dateCreated = string[3];
191                                     }
192                                     if (!p.cashAccList.contains(c)) {
193                                     c.addCashAccount(c.name, c.
194                                     balance, c.dateCreated, p);
195                                     }
196                                     } else if (string[0].equals("transaction"
197                                     )) {
198                                     if (string[1].equals("EQUITY") ||
199                                     string[1].equals("equity")) {
200                                     Equity c = new Equity("", "", 0,
201                                     0, "", null);
202                                     History caH = new History("N/A",
203                                     "N/A", c, "", 0, 0);
204                                     if (string.length == 5) {
205                                     caH.type = string[1];
206                                     caH.equityName = string[2];
207                                     caH.transferAmount1 = Double.
208                                     parseDouble(string[3]);
209                                     caH.date1 = string[4];
210                                     }
211                                     }
212                                     }
213                                     }
214                                     }
215                                     }
216                                     }
217                                     }
218                                     }
219                                     }
220                                     }
221                                     }
222                                     }
223                                     }
224                                     }
225                                     }
226                                     }
227                                     }
228                                     }
229                                     }
230                                     }
231                                     }
232                                     }
233                                     }
234                                     }
235                                     }
236                                     }
237                                     }
238                                     }
239                                     }
240                                     }
241                                     }
242                                     }
243                                     }
244                                     }
245                                     }
246                                     }
247                                     }
248                                     }
249                                     }
250                                     }
251                                     }
252                                     }
253                                     }
254                                     }
255                                     }
256                                     }
257                                     }
258                                     }
259                                     }
260                                     }
261                                     }
262                                     }
263                                     }
264                                     }
265                                     }
266                                     }
267                                     }
268                                     }
269                                     }
270                                     }
271                                     }
272                                     }
273                                     }
274                                     }
275                                     }
276                                     }
277                                     }
278                                     }
279                                     }
280                                     }
281                                     }
282                                     }
283                                     }
284                                     }
285                                     }
286                                     }
287                                     }
288                                     }
289                                     }
290                                     }
291                                     }
292                                     }
293                                     }
294                                     }
295                                     }
296                                     }
297                                     }
298                                     }
299                                     }
300                                     }
301                                     }
302                                     }
303                                     }
304                                     }
305                                     }
306                                     }
307                                     }
308                                     }
309                                     }
310                                     }
311                                     }
312                                     }
313                                     }
314                                     }
315                                     }
316                                     }
317                                     }
318                                     }
319                                     }
320                                     }
321                                     }
322                                     }
323                                     }
324                                     }
325                                     }
326                                     }
327                                     }
328                                     }
329                                     }
330                                     }
331                                     }
332                                     }
333                                     }
334                                     }
335                                     }
336                                     }
337                                     }
338                                     }
339                                     }
340                                     }
341                                     }
342                                     }
343                                     }
344                                     }
345                                     }
346                                     }
347                                     }
348                                     }
349                                     }
350                                     }
351                                     }
352                                     }
353                                     }
354                                     }
355                                     }
356                                     }
357                                     }
358                                     }
359                                     }
360                                     }
361                                     }
362                                     }
363                                     }
364                                     }
365                                     }
366                                     }
367                                     }
368                                     }
369                                     }
370                                     }
371                                     }
372                                     }
373                                     }
374                                     }
375                                     }
376                                     }
377                                     }
378                                     }
379                                     }
380                                     }
381                                     }
382                                     }
383                                     }
384                                     }
385                                     }
386                                     }
387                                     }
388                                     }
389                                     }
390                                     }
391                                     }
392                                     }
393                                     }
394                                     }
395                                     }
396                                     }
397                                     }
398                                     }
399                                     }
400                                     }
401                                     }
402                                     }
403                                     }
404                                     }
405                                     }
406                                     }
407                                     }
408                                     }
409                                     }
410                                     }
411                                     }
412                                     }
413                                     }
414                                     }
415                                     }
416                                     }
417                                     }
418                                     }
419                                     }
420                                     }
421                                     }
422                                     }
423                                     }
424                                     }
425                                     }
426                                     }
427                                     }
428                                     }
429                                     }
430                                     }
431                                     }
432                                     }
433                                     }
434                                     }
435                                     }
436                                     }
437                                     }
438                                     }
439                                     }
440                                     }
441                                     }
442                                     }
443                                     }
444                                     }
445                                     }
446                                     }
447                                     }
448                                     }
449                                     }
450                                     }
451                                     }
452                                     }
453                                     }
454                                     }
455                                     }
456                                     }
457                                     }
458                                     }
459                                     }
460                                     }
461                                     }
462                                     }
463                                     }
464                                     }
465                                     }
466                                     }
467                                     }
468                                     }
469                                     }
470                                     }
471                                     }
472                                     }
473                                     }
474                                     }
475                                     }
476                                     }
477                                     }
478                                     }
479                                     }
480                                     }
481                                     }
482                                     }
483                                     }
484                                     }
485                                     }
486                                     }
487                                     }
488                                     }
489                                     }
490                                     }
491                                     }
492                                     }
493                                     }
494                                     }
495                                     }
496                                     }
497                                     }
498                                     }
499                                     }
500                                     }
501                                     }
502                                     }
503                                     }
504                                     }
505                                     }
506                                     }
507                                     }
508                                     }
509                                     }
510                                     }
511                                     }
512                                     }
513                                     }
514                                     }
515                                     }
516                                     }
517                                     }
518                                     }
519                                     }
520                                     }
521                                     }
522                                     }
523                                     }
524                                     }
525                                     }
526                                     }
527                                     }
528                                     }
529                                     }
530                                     }
531                                     }
532                                     }
533                                     }
534                                     }
535                                     }
536                                     }
537                                     }
538                                     }
539                                     }
540                                     }
541                                     }
542                                     }
543                                     }
544                                     }
545                                     }
546                                     }
547                                     }
548                                     }
549                                     }
550                                     }
551                                     }
552                                     }
553                                     }
554                                     }
555                                     }
556                                     }
557                                     }
558                                     }
559                                     }
560                                     }
561                                     }
562                                     }
563                                     }
564                                     }
565                                     }
566                                     }
567                                     }
568                                     }
569                                     }
570                                     }
571                                     }
572                                     }
573                                     }
574                                     }
575                                     }
576                                     }
577                                     }
578                                     }
579                                     }
580                                     }
581                                     }
582                                     }
583                                     }
584                                     }
585                                     }
586                                     }
587                                     }
588                                     }
589                                     }
590                                     }
591                                     }
592                                     }
593                                     }
594                                     }
595                                     }
596                                     }
597                                     }
598                                     }
599                                     }
600                                     }
601                                     }
602                                     }
603                                     }
604                                     }
605                                     }
606                                     }
607                                     }
608                                     }
609                                     }
610                                     }
611                                     }
612                                     }
613                                     }
614                                     }
615                                     }
616                                     }
617                                     }
618                                     }
619                                     }
620                                     }
621                                     }
622                                     }
623                                     }
624                                     }
625                                     }
626                                     }
627                                     }
628                                     }
629                                     }
630                                     }
631                                     }
632                                     }
633                                     }
634                                     }
635                                     }
636                                     }
637                                     }
638                                     }
639                                     }
640                                     }
641                                     }
642                                     }
643                                     }
644                                     }
645                                     }
646                                     }
647                                     }
648                                     }
649                                     }
650                                     }
651                                     }
652                                     }
653                                     }
654                                     }
655                                     }
656                                     }
657                                     }
658                                     }
659                                     }
660                                     }
661                                     }
662                                     }
663                                     }
664                                     }
665                                     }
666                                     }
667                                     }
668                                     }
669                                     }
670                                     }
671                                     }
672                                     }
673                                     }
674                                     }
675                                     }
676                                     }
677                                     }
678                                     }
679                                     }
680                                     }
681                                     }
682                                     }
683                                     }
684                                     }
685                                     }
686                                     }
687                                     }
688                                     }
689                                     }
690                                     }
691                                     }
692                                     }
693                                     }
694                                     }
695                                     }
696                                     }
697                                     }
698                                     }
699                                     }
700                                     }
701                                     }
702                                     }
703                                     }
704                                     }
705                                     }
706                                     }
707                                     }
708                                     }
709                                     }
710                                     }
711                                     }
712                                     }
713                                     }
714                                     }
715                                     }
716                                     }
717                                     }
718                                     }
719                                     }
720                                     }
721                                     }
722                                     }
723                                     }
724                                     }
725                                     }
726                                     }
727                                     }
728                                     }
729                                     }
730                                     }
731                                     }
732                                     }
733                                     }
734                                     }
735                                     }
736                                     }
737                                     }
738                                     }
739                                     }
740                                     }
741                                     }
742                                     }
743                                     }
744                                     }
745                                     }
746                                     }
747                                     }
748                                     }
749                                     }
750                                     }
751                                     }
752                                     }
753                                     }
754                                     }
755                                     }
756                                     }
757                                     }
758                                     }
759                                     }
760                                     }
761                                     }
762                                     }
763                                     }
764                                     }
765                                     }
766                                     }
767                                     }
768                                     }
769                                     }
770                                     }
771                                     }
772                                     }
773                                     }
774                                     }
775                                     }
776                                     }
777                                     }
778                                     }
779                                     }
779                                     }
780                                     }
781                                     }
782                                     }
783                                     }
784                                     }
785                                     }
786                                     }
787                                     }
788                                     }
789                                     }
789                                     }
790                                     }
791                                     }
792                                     }
793                                     }
794                                     }
795                                     }
796                                     }
797                                     }
798                                     }
799                                     }
799                                     }
800                                     }
801                                     }
802                                     }
803                                     }
804                                     }
805                                     }
806                                     }
807                                     }
808                                     }
809                                     }
809                                     }
810                                     }
811                                     }
812                                     }
813                                     }
814                                     }
815                                     }
816                                     }
817                                     }
818                                     }
819                                     }
819                                     }
820                                     }
821                                     }
822                                     }
823                                     }
824                                     }
825                                     }
826                                     }
827                                     }
828                                     }
829                                     }
829                                     }
830                                     }
831                                     }
832                                     }
833                                     }
834                                     }
835                                     }
836                                     }
837                                     }
838                                     }
839                                     }
839                                     }
840                                     }
841                                     }
842                                     }
843                                     }
844                                     }
845                                     }
846                                     }
847                                     }
848                                     }
849                                     }
849                                     }
850                                     }
851                                     }
852                                     }
853                                     }
854                                     }
855                                     }
856                                     }
857                                     }
858                                     }
859                                     }
859                                     }
860                                     }
861                                     }
862                                     }
863                                     }
864                                     }
865                                     }
866                                     }
867                                     }
868                                     }
869                                     }
869                                     }
870                                     }
871                                     }
872                                     }
873                                     }
874                                     }
875                                     }
876                                     }
877                                     }
878                                     }
879                                     }
879                                     }
880                                     }
881                                     }
882                                     }
883                                     }
884                                     }
885                                     }
886                                     }
887                                     }
888                                     }
889                                     }
889                                     }
890                                     }
891                                     }
892                                     }
893                                     }
894                                     }
895                                     }
896                                     }
897                                     }
898                                     }
899                                     }
899                                     }
900                                     }
901                                     }
902                                     }
903                                     }
904                                     }
905                                     }
906                                     }
907                                     }
908                                     }
909                                     }
909                                     }
910                                     }
911                                     }
912                                     }
913                                     }
914                                     }
915                                     }
916                                     }
917                                     }
918                                     }
919                                     }
919                                     }
920                                     }
921                                     }
922                                     }
923                                     }
924                                     }
925                                     }
926                                     }
927                                     }
928                                     }
929                                     }
929                                     }
930                                     }
931                                     }
932                                     }
933                                     }
934                                     }
935                                     }
936                                     }
937                                     }
938                                     }
939                                     }
939                                     }
940                                     }
941                                     }
942                                     }
943                                     }
944                                     }
945                                     }
946                                     }
947                                     }
948                                     }
949                                     }
949                                     }
950                                     }
951                                     }
952                                     }
953                                     }
954                                     }
955                                     }
956                                     }
957                                     }
958                                     }
959                                     }
959                                     }
960                                     }
961                                     }
962                                     }
963                                     }
964                                     }
965                                     }
966                                     }
967                                     }
968                                     }
969                                     }
969                                     }
970                                     }
971                                     }
972                                     }
973                                     }
974                                     }
975                                     }
976                                     }
977                                     }
978                                     }
979                                     }
979                                     }
980                                     }
981                                     }
982                                     }
983                                     }
984                                     }
985                                     }
986                                     }
987                                     }
988                                     }
989                                     }
989                                     }
990                                     }
991                                     }
992                                     }
993                                     }
994                                     }
995                                     }
996                                     }
997                                     }
998                                     }
999                                     }
999                                     }

```

```

198 } else{
199     caH.type = string[1];
200     caH.equityName = string[2] +=
201     ", Inc.";
202     caH.transferAmount1 = Double.
203     parseDouble(string[4]);
204 }
205     p.equityHistory.add(caH);
206 } else if (string[1].equals(
207     "CASHACCOUNT") || string[1].equals("cash account")) {
208     CashAccount c = new CashAccount(
209     "", "", 0, "", null);
210     History caH = new History("N/A",
211     "N/A", c, "", 0);
212
213     caH.type = string[1];
214     caH.cashAccName = string[2];
215     caH.transferAmount2 = Double.
216     parseDouble(string[3]);
217     caH.date2 = string[4];
218
219     p.cashAccountHistory.add(caH);
220 }
221 } else if (string[0].equals(
222     "watchlistItem")) {
223     Watchlist w = new Watchlist("", "", "",
224     "", 0, "", "", "");
225     w.tickerSymbol = string[1];
226     w.lowtrigger = string[2];
227     w.hightrigger = string[3];
228
229     p.watchlists.add(w);
230 } else if (string[0].equals(
231     "marketAverage")) {
232
233     String tmp = string[1];
234     List<Equity> curMarAvgs = this.
235     marketAverages.stream().filter(u -> u.name.equals(tmp)).
236     collect(Collectors.toList());
237
238     if (!(curMarAvgs.get(0).name.equals(
239         string[1]))){

```

```

231                     Equity c = new Equity("", "", 0,
232                         0, "", null);
233                     c.MA = string[1];
234                     c.shares = Integer.parseInt(
235                         string[2]);
236                     c.acquisitionPrice = Double.
237                         parseDouble(string[3]);
238                     p.marketAverages.add(c);
239
240
241                     /**
242                     Equity c = new Equity("", "", 0, 0,
243                         "", null);
244                     c.MA = string[1];
245                     c.shares = Integer.parseInt(string[2
246                         ]);
247                     c.acquisitionPrice = Double.
248                         parseDouble(string[3]);
249                     p.marketAverages.add(c); */
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267     public void exportP(String eFileName, Portfolio p) {
268         try{

```

```

269         FileWriter writer = new FileWriter(eFileName)
270     ;
271
272     for (Equity equity : p.equityList1) {
273         writer.append("equity");
274         writer.append(',');
275         writer.append(equity.shares + "");
276         writer.append(',');
277         writer.append(equity.tickerSymbol);
278         writer.append(',');
279         writer.append(equity.getTotal() + "");
280         writer.append(',');
281         writer.append(equity.name);
282         equity.marketAverage.forEach((SI) -> {
283             try {
284                 writer.append(",");
285                 writer.append(SI.toString());
286             }
287             catch (IOException IO) {
288                 IO.printStackTrace();
289             } catch (NullPointerException e) {
290                 e.printStackTrace();
291             }
292         });
293         writer.append('\n');
294     }
295
296     for (CashAccount ca : p.cashAccList) {
297         writer.append("cashacc");
298         writer.append(',');
299         writer.append(ca.balance + "");
300         writer.append(',');
301         writer.append(ca.getName());
302         writer.append('\n');
303     }
304
305     for (History h : p.cashAccountHistory) {
306         writer.append("transaction");
307         writer.append(',');
308         writer.append(h.type);
309         writer.append(',');
310         writer.append(h.cashAccName);
311         writer.append(',');
312         writer.append(h.transferAmount2.toString())

```

```

312 );
313         writer.append(',');
314         writer.append(h.date2);
315         writer.append("\n");
316     }
317
318     for (History h : p.equityHistory) {
319         writer.append("transaction");
320         writer.append(',');
321         writer.append(h.type);
322         writer.append(',');
323         writer.append(h.equityName);
324         writer.append(',');
325         writer.append(h.transferAmount1.toString());
326     }
327     writer.append(',');
328     writer.append(h.date1);
329     writer.append("\n");
330 }
331
332     for (Watchlist w : p.watchlists) {
333         writer.append("watchlistItem");
334         writer.append(',');
335         writer.append(w.getTickerSymbol());
336         writer.append(',');
337         writer.append(w.lowtrigger);
338         writer.append(',');
339         writer.append(w.hightrigger);
340         writer.append("\n");
341     }
342
343     for (Equity ma : p.marketAverages) {
344         writer.append("marketAverage");
345         writer.append(',');
346         writer.append(ma.MA.replace("[", ""));
347         writer.append(replace("]", ""));
348         writer.append(',');
349         writer.append(Integer.toString(ma.shares));
350     }
351     writer.append(',');
352     writer.append(Double.toString(ma.acquisitionPrice));
353     writer.append("\n");
354 }
```

```

353             writer.flush();
354             writer.close();
355
356         }catch (IOException e) {
357             e.printStackTrace();
358         }
359     }
360
361     /*
362      * Code for saving market averages
363      */
364     public void saveMAs(Portfolio p) {
365         ArrayList<Portfolio> port = null;
366         try {
367             FileInputStream fileIn = new FileInputStream(
368             "myfile");
369             ObjectInputStream in = new ObjectInputStream(
370             fileIn);
371             port = (ArrayList<Portfolio>) in.readObject()
372             ;
373             in.close();
374             fileIn.close();
375         } catch (ClassNotFoundException ee) {
376             ee.printStackTrace();
377         } catch (FileNotFoundException ee) {
378             ee.printStackTrace();
379         } catch (IOException ee) {
380             ee.printStackTrace();
381         }
382
383         try
384         {
385             FileOutputStream fileOut =
386                 new FileOutputStream("myfile");
387             ObjectOutputStream out = new
388             ObjectOutputStream(fileOut);
389             out.writeObject(port);
390             out.close();
391             fileOut.close();
392         }catch(IOException i)
393         {
394             i.printStackTrace();

```

```
394      }
395    }
396  }
397
```

```

1 package trunk.Model;
2
3 import java.io.Serializable;
4
5 /**
6  * Created by Daniel on 4/14/2016.
7 */
8 public class Watchlist implements Serializable,
9     WatchlistVisitor {
10    public String tickerSymbol;
11    public String lowtrigger;
12    public String hightigger;
13    public double pershareprice;
14
15    public String compare;
16    public String lowtriggered;
17    public String highttriggered;
18
19    /**
20     * Constructor for an Equity object.
21     * @param tickerSymbol: A String representing the
22     market symbol associated with this equity.
23     * @param low : A string representing the low trigger.
24     * @param high : A string representing the hightigger
25
26     * @param price : A double representing the orginal
27     price pad for this equity, per share.
28     * @param compare : A String representing that a
29     trigger was tripped
30     * @param lowtriggered : A string representing that
31     the low trigger has been tripped.
32     * @param highttriggered : A string representing that
33     the high trigger has been tripped.
34
35    /**
36    public Watchlist(String tickerSymbol, String low,
37        String high, double price, String compare, String
38        lowtriggered, String highttriggered) {
39        this.tickerSymbol = tickerSymbol;
40        this.lowtrigger = low;
41        this.hightigger = high;
42        this.pershareprice = price;
43        this.compare = compare;
44        this.lowtriggered = lowtriggered;
45        this.highttriggered = highttriggered;
46    }

```

```

37
38     @Override
39     public void visit(Portfolio p) {
40         for(Watchlist w : p.watchlists) {
41             w.compare = "Within range";
42             if(!w.lowtrigger.equals("")) {
43                 if (w.pershareprice < Double.parseDouble(w
44 .lowtrigger)) {
45                     w.compare = "Low Trigger";
46                     w.lowtriggered = "Was Tripped";
47                 }
48             if(!w.hightrigger.equals("")) {
49                 if (w.pershareprice > Double.parseDouble(w
50 .hightrigger)) {
51                     w.compare = "High Trigger";
52                     w.hightriggered = "Was Tripped";
53                 }
54             }
55         }
56
57         //A String representing the market symbol associated
58         //with this equity.
59         public String getTickerSymbol(){return this.
60         tickerSymbol;}
61
62         //A string representing the low trigger.
63         public String getLowtrigger(){return this.lowtrigger;}
64
65         //A string representing the hightrigger.
66         public String getHightrigger(){return this.hightrigger
67 ;}
68
69         //A double representing the orginal price pad for this
70         //equity, per share.
71         public double getPershareprice(){return this.
72         pershareprice;}
73
74         //A String representing that a trigger was tripped
75         public String getCompare(){return this.compare;}
76
77         //A string representing that the low trigger has been
78         //tripped.
79         public String getLowtriggered(){return this.

```

```
73 lowtriggered; }  
74  
75     //A string representing that the high trigger has  
    been tripped.  
76     public String getHightriggered() {return this.  
        hightriggered; }  
77 }  
78
```

```
1 package trunk.Model;  
2  
3 /**  
4   * An interface to handle various types of logging.  
5 */  
6  
7 public interface LogHistory {  
8   public void log(Portfolio portfolio, History history);  
9 }  
10
```

```

1 package trunk.Model;
2
3 import javafx.scene.Scene;
4 import javafx.stage.Stage;
5 import trunk.View.CAView;
6 import trunk.View.EquitiesView;
7 import trunk.View.FPTS;
8
9 import java.io.BufferedReader;
10 import java.io.FileNotFoundException;
11 import java.io.FileReader;
12 import java.io.IOException;
13 import java.util.ArrayDeque;
14 import java.util.ArrayList;
15 import java.util.HashSet;
16 import java.util.Iterator;
17
18 /**
19  * Created by sadaf345 on 4/5/2016.
20 */
21 public class UndoAction implements handleCommand{
22     public String actionType;
23     public String objType;
24     public String file;
25     public CashAccount obj;
26     public CashAccount transfer;
27     public double transferAmount;
28
29     public Equity item;
30     public Portfolio p;
31     public static ArrayDeque<UndoAction> undo = new
32         ArrayDeque<>();
33     public static ArrayDeque<UndoAction> redo = new
34         ArrayDeque<>();
35
36     public UndoAction(String action, String objectType,
37     CashAccount c1, CashAccount c2, double num,Equity eq,
38     String file) {
39         this.actionType = action;
40         this.obj = c1;
41         this.objType = objectType;
42         this.transfer = c2;
43         this.transferAmount = num;
44         this.item = eq;

```

```

42         this.file = file;
43     }
44
45
46     public UndoAction(Portfolio p) {
47         this.p = p;
48     }
49
50     @Override
51     public void handleCommand(final Stage primaryStage,
52                             FPTS fpts, EquitiesView equitiesView, CAView caView) {
53         Iterator undoIterator = createUndoIterator();
54
55         for( Iterator iterate = undoIterator; iterate.
56               hasNext();) {
57             UndoAction iter = (UndoAction) iterate.next();
58             if (iter.actionType.equals("Add") && iter.
59                  objType.equals("CashAccount")) {
60                 UndoAddCashAcc(iter.obj, this.p);
61                 redo.addFirst(iter);
62                 undo.remove(iter);
63                 Scene scene = caView.CashAccounts(
64                     primaryStage,p, fpts);
65                 fpts.stage.setScene(scene);
66                 break;
67             }
68             else if (iter.actionType.equals("Remove") &&
69             iter.objType.equals("CashAccount")) {
70                 UndoRemoveCashAcc(iter.obj, this.p);
71                 redo.addFirst(iter);
72                 undo.remove(iter);
73                 Scene scene = caView.CashAccounts(
74                     primaryStage,p, fpts);
75                 fpts.stage.setScene(scene);
76                 break;
77             }
78
79             else if(iter.actionType.equals("Transfer") &&
80             iter.objType.equals("CashAccount")){
81                 UndoTransfer(iter.obj, iter.transfer, iter
82                 .transferAmount, this.p);
83                 redo.addFirst(iter);
84                 undo.remove(iter);
85                 Scene scene = caView.CashAccounts(
86                     primaryStage,p, fpts);

```

```

78                     fpts.stage.setScene(scene);
79                     break;
80                 }
81
82             else if (iter.actionType.equals("Add") &&
83             iter.objType.equals("Equity")) {
84                 UndoAddEq(iter.item, this.p);
85                 redo.addFirst(iter);
86                 undo.remove(iter);
87                 Scene scene = equitiesView.ViewEquities(
88                   primaryStage,p, fpts);
89                 fpts.stage.setScene(scene);
90                 break;
91             }
92             else if (iter.actionType.equals("Remove") &&
93             iter.objType.equals("Equity")) {
94                 UndoRemoveEq(iter.item, this.p);
95                 redo.addFirst(iter);
96                 undo.remove(iter);
97                 Scene scene = equitiesView.ViewEquities(
98                   primaryStage,p, fpts);
99                 fpts.stage.setScene(scene);
100                break;
101            }
102
103            else if (iter.actionType.equals("Import") &&
104             iter.objType.equals("CashAccount")) {
105                 UndoImportCA(iter.file, this.p);
106                 redo.addFirst(iter);
107                 undo.remove(iter);
108                 Scene scene = caView.CashAccounts(
109                   primaryStage,p, fpts);
110                 fpts.stage.setScene(scene);
111                 break;
112             }
113             else if (iter.actionType.equals("Import") &&
114             iter.objType.equals("Equity")) {
115                 UndoImportEQ(iter.file, this.p);
116                 redo.addFirst(iter);
117                 undo.remove(iter);
118                 Scene scene = equitiesView.ViewEquities(
119                   primaryStage,p, fpts);
120                 fpts.stage.setScene(scene);
121                 break;
122             }
123         }
124     }
125 }
```

```

115         }
116     }
117
118     public Iterator createUndoIterator() {
119         Iterator undoIter = undo.iterator();
120         return undoIter;
121     }
122     public Iterator createRedoIterator() {
123         Iterator undoIter = redo.iterator();
124         return undoIter;
125     }
126
127     public void redo(final Stage primaryStage, FPTS fpts,
128         EquitiesView equitiesView, CAView caView) {
129         Iterator undoIterator = createRedoIterator();
130         for( Iterator iterate = undoIterator; iterate.
131             hasNext();) {
132             UndoAction iter = (UndoAction) iterate.next()
133             ;
134             if (iter.actionType.equals("Add") && iter.
135                 objType.equals("CashAccount") ) {
136                 UndoRemoveCashAcc(iter.obj, this.p);
137                 addToQueue(iter);
138                 redo.remove(iter);
139                 Scene scene = caView.CashAccounts(
140                     primaryStage,p, fpts);
141                 fpts.stage.setScene(scene);
142                 break;
143             }
144             else if (iter.actionType.equals("Remove") &&
145                 iter.objType.equals("CashAccount") ) {
146                 UndoAddCashAcc(iter.obj, this.p);
147                 addToQueue(iter);
148                 redo.remove(iter);
149                 Scene scene = caView.CashAccounts(
150                     primaryStage,p, fpts);
151                 fpts.stage.setScene(scene);
152                 break;
153             }
154             else if(iter.actionType.equals("Transfer") &&
155                 iter.objType.equals("CashAccount")){
156                 UndoTransfer(iter.transfer,iter.obj, iter
157                 .transferAmount, this.p);
158                 addToQueue(iter);
159                 redo.remove(iter);

```



```

186             c.importEQ(iter.file,p);
187
188             addToQueue(iter);
189             redo.remove(iter);
190             Scene scene = equitiesView.ViewEquities(
191             primaryStage,p, fpts);
192             fpts.stage.setScene(scene);
193             break;
194         }
195     }
196     public void UndoAddCashAcc(CashAccount obj, Portfolio
197 p) {
198     obj.deleteCashAccount(obj.getName(), p); //  

deletes
199   }
200
201   public void UndoRemoveCashAcc(CashAccount obj,
202 Portfolio p) { // need Redo class
203     obj.addCashAccount(obj.name,obj.balance,obj.
204 getDate(),p);
205   }
206
207
208
209
210   public void UndoAddEq(Equity obj, Portfolio p) {
211     obj.removeEquity(obj.tickerSymbol,"None",p);
212   }
213
214   public void UndoRemoveEq(Equity obj, Portfolio p) {
215     obj.addEquity(obj.tickerSymbol,obj.name,obj.
216 shares,obj.acquisitionPrice,obj.acquisitionDate,obj.
217 marketAverage,false,p);
218   }
219
220   public void UndoImportCA(String csv, Portfolio p) {
221     BufferedReader br = null;
222     String line = "";

```

```

223         String[] ca = null;
224         ArrayList<CashAccount> toRemove = new ArrayList<>()
225             ();
226
227         try {
228
229             br = new BufferedReader(new FileReader(csv));
230             while ((line = br.readLine()) != null) {
231                 ca = line.split(",(?=([^\"]*\"[^\"]*\")*\""
232                     )*[^"]*$)", -1);
233
234                 for(CashAccount c : p.cashAccList) {
235                     if(c.name.equals(ca[2])) {
236                         toRemove.add(c);
237                     }
238                     for(CashAccount c : toRemove) {
239                         p.cashAccList.remove(c);
240                     }
241                 }
242
243             } catch (FileNotFoundException ee) {
244                 ee.printStackTrace();
245             } catch (IOException ee) {
246                 ee.printStackTrace();
247             } finally {
248                 if (br != null) {
249                     try {
250                         br.close();
251                     } catch (IOException ee) {
252                         ee.printStackTrace();
253                     }
254                 }
255             }
256         }
257
258     public void UndoImportEQ(String csv, Portfolio p) {
259         BufferedReader br = null;
260         String line = "";
261         String[] equity = null;
262         ArrayList<Equity> toRemove = new ArrayList<>();
263
264         try {
265             br = new BufferedReader(new FileReader(csv));

```

```

266             while ((line = br.readLine()) != null) {
267                 equity = line.split(",(?=([" + "\"]*\"[" + "\"]*"
268                 \")*[" + "\"]*\$)", -1);
269
270                 for(Equity c : p.equityList2) {
271                     if(c.tickerSymbol.equals(equity[0] .
272                         replaceAll("^\"|\"$", ""))) {
273                         toRemove.add(c);
274                     }
275                 }
276                 for(Equity c : toRemove) {
277                     p.equityList2.remove(c);
278                     //Resetting Market Averages
279                     for(String m : c.marketAverage) {
280                         ArrayList<String> avgArray = new
281                             ArrayList<>();
282                         avgArray.add(m);
283                         Equity marketAvg = new Equity("", ,
284                             "", 0, 0, null, avgArray);
285                         double price = 0;
286                         int count = 0;
287                         for(Equity q : p.equityList2) {
288                             if(q.marketAverage.contains(m
289 )) {
290                                 price += q.
291                             acquisitionPrice;
292                                 count += 1;
293                             }
294                         }
295                         p.equityList2.remove(marketAvg);
296                         if(!m.equals("DOW")) {
297                             marketAvg.acquisitionPrice =
298                             Math.round((price / count) * 100.0) / 100.0;
299                         } else{
300                             marketAvg.acquisitionPrice =
301                             Math.round(((price / count) / .14602128057775) * 100.0) /
302                             100.0;
303                         }
304                         p.equityList2.add(marketAvg);
305                     }
306                 }
307             }
308         } catch (FileNotFoundException ee) {
309             ee.printStackTrace();

```

```
302         } catch (IOException ee) {
303             ee.printStackTrace();
304         } finally {
305             if (br != null) {
306                 try {
307                     br.close();
308                 } catch (IOException ee) {
309                     ee.printStackTrace();
310                 }
311             }
312             ArrayList<Equity> equityListCopy = new
313             ArrayList<>(p.equityList2);
314             HashSet<String> existingAvgs = new HashSet<>(
315             );
316             for(Equity e : equityListCopy){
317                 if(!e.tickerSymbol.equals("")){
318                     existingAvgs.addAll(e.marketAverage);
319                 }
320                 for(Equity e : equityListCopy){
321                     if(e.tickerSymbol.equals("") && !
322                     existingAvgs.contains(e.marketAverage.get(0))){
323                         p.equityList2.remove(e);
324                     }
325                 }
326             }
327             public void allclear(){
328                 undo.clear();
329                 redo.clear();
330             }
331
332             public void addToQueue(UndoAction inst) {
333                 undo.addFirst(inst);
334             }
335
336 }
337
```

```

1 package trunk.Model;
2
3 /**
4  * A class to represent an instance of a Cash Account.
5  * A Cash Account is responsible for maintaining funds,
6  * interacting
7  * with equities, and interacting with a Portfolio.
8  */
9
10
11 import trunk.Control.HistoryInvoker;
12
13 import java.io.*;
14 import java.text.DateFormat;
15 import java.text.DecimalFormat;
16 import java.text.SimpleDateFormat;
17 import java.util.ArrayList;
18 import java.util.Date;
19
20
21
22
23
24
25 /**
26  * Constructor for a CashAccount Object.
27  * @param portfolioID : A string representing the
28  * portfolioID this account belongs to.
29  * @param name : A string representing the name of
30  * this account.
31  * @param dateCreated : A String representing the date
32  * the account was created.
33  * @param portfolio : The Portfolio object this
34  * account belongs to.
35  */
36
37 public CashAccount(String portfolioID, String name,
38                      double initAmount, String
39                      dateCreated, trunk.Model.Portfolio portfolio) {
40
41     this.portfolioID = portfolioID;
42     this.name = name;
43     this.dateCreated = dateCreated;
44     DecimalFormat df = new DecimalFormat("#.00");
45     this.balance = Double.parseDouble(df.format(

```

```

39 initAmount));
40         this.portfolio = portfolio;
41     }
42
43
44     /**
45      * A getter for the portfolioID associated with this
46      * account.
47      * @return : A string representing the portfolioID.
48
49      public String getPortfolioID(){
50         return this.portfolioID;
51     }
52
53     /**
54      * A getter for the name given to this account.
55      * @return : A string representing the account's name.
56
57      public String getName() {
58         return this.name;
59     }
60
61     /**
62      * A getter for the date associated with this account'
63      * s creation.
64      * @return : A string representing the creation date.
65
66      public String getDate() {
67         return this.dateCreated;
68     }
69
70     /**
71      * A getter for balance associated with this account
72      * @return : The double amount representing current
73      * cash amount in this account
74
75      * Transfer funds from one account to another.
76      * @param amount : An int representing the amount to
77      * transfer.
78      * @param receiver : A CashAccount representing the
79      * account
80      * receiving the amount.

```

```

79      */
80      public void transferFunds(int amount, CashAccount
81      receiver) {
82          if (amount > this.balance) {
83              return;
84          } else {
85              receiver.balance += amount;
86              this.balance -= amount;
87
88              DateFormat dateFormat = new SimpleDateFormat(
89                  "yyyy/MM/dd HH:mm:ss");
90              Date date = new Date();
91
92              HistoryInvoker test = new HistoryInvoker();
93              test.setLog(new LogCAHistory(), this.
94              portfolio, new History("Update", "CASHACCOUNT", this,
95              dateFormat.format(date), -amount));
96              test.logAction();
97
98          /**
99             * Add a cash account
100            */
101         public void addCashAccount(String name, double
102             initAmount, String date, Portfolio p){
103             boolean same = false;
104             for(CashAccount c : p.cashAccList){
105                 if(c.getName().equals(name)) {
106                     same = true;
107                     break;
108                 }
109             }
110             CashAccount newAcc = new CashAccount(p.
111             loginID, name, initAmount, date, p);
112             p.cashAccList.add(newAcc);
113
114             DateFormat dateFormat = new SimpleDateFormat(
115                 "yyyy/MM/dd HH:mm:ss");
116             Date date2 = new Date();

```

```

117         test.setLog(new LogCAHistory(), p, new
118             History("Add", "CASHACCOUNT", newAcc, dateFormat.format(
119                 date2), 0));
120         test.logAction();
121     }
122     else{
123         for(int i = 0; i < p.cashAccList.size(); i++)
124         {
125             if(p.cashAccList.get(i).getName().equals(
126                 name)) {
127                 p.cashAccList.get(i).balance =
128                     initAmount;
129                 p.cashAccList.get(i).dateCreated =
130                     date;
131                 break;
132             }
133         }
134     }
135     /*
136      * Import cash accounts
137      */
138     public ArrayList<CashAccount> importCA(String csv,
139         Portfolio p) {
140         BufferedReader br = null;
141         String line = "";
142         String[] ca = null;
143
144         ArrayList<CashAccount> existingAccts = new
145             ArrayList<>();
146
147         try {
148             br = new BufferedReader(new FileReader(csv));
149             while ((line = br.readLine()) != null) {
150                 ca = line.split(",(?=([" + "\"])*\"[" + "\"]*\\" +
151                     ")*[" + "\"]*$)", -1);
152                 CashAccount c = new CashAccount("N/A", "N
153 /A", 0.0, "", null);
154                 c.name = ca[2];
155                 c.balance = Double.parseDouble(ca[1].
156                     replace("\\"", ""));
157                 c.dateCreated = ca[3];
158             }
159         }
160     }

```

```

151             if(!p.cashAccList.contains(c)) {
152                 addCashAccount(c.name, c.balance, c.
153                             dateCreated, p);
154             } else{
155                 existingAccts.add(c);
156             }
157         }
158     } catch (FileNotFoundException ee) {
159         ee.printStackTrace();
160     } catch (IOException ee) {
161         ee.printStackTrace();
162     } finally {
163         if (br != null) {
164             try {
165                 br.close();
166             } catch (IOException ee) {
167                 ee.printStackTrace();
168             }
169         }
170     }
171     return existingAccts;
172 }
173 /**
174 * Export Cash Account
175 */
176 public void exportCA(String eFileName, Portfolio p) {
177     try{
178         FileWriter writer = new FileWriter(eFileName)
179 ;
180         for (CashAccount ca : p.cashAccList){
181             writer.append("cashacc");
182             writer.append(',');
183             writer.append(ca.balance + "");
184             writer.append(',');
185             writer.append(ca.getName());
186             writer.append(',');
187             writer.append(ca.dateCreated);
188             writer.append('\n');
189         }
190     }
191     writer.flush();
192     writer.close();

```

```

194
195         }catch (IOException e) {
196             e.printStackTrace();
197         }
198
199     }
200
201     /**
202      * Remove cash account based on name given.
203      */
204     public void deleteCashAccount(String name, Portfolio
p) {
205         ArrayList<CashAccount> copyList = new ArrayList<>
(p.cashAccList);
206
207         copyList.forEach((account) -> {
208             if (account.name.equals(name)) {
209                 p.cashAccList.remove(account);
210
211                 DateFormat dateFormat = new
SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
212                 Date date = new Date();
213
214                 HistoryInvoker test = new HistoryInvoker(
);
215                 test.setLog(new LogCAHistory(), p, new
History("Remove", "CASHACCOUNT", account, dateFormat.format
(date), 0));
216                 test.logAction();
217             }
218         });
219     }
220
221     /**
222      * Update funds of account depending on transaction
type
223      */
224     public void updateBalance(String name, double amount,
Boolean deposit, Portfolio p){
225         p.cashAccList.forEach((account) -> {
226             if (account.name.equals(name)) {
227                 if(deposit) {
228                     account.balance += amount;
229                     DateFormat dateFormat = new
SimpleDateFormat("yyyy/MM/dd HH:mm:ss");

```

```

230                     Date date = new Date();
231
232                     HistoryInvoker test = new
233                         HistoryInvoker();
234                         test.setLog(new LogCAHistory(), p,
235                         new History("Update", "CASHACCOUNT", account, dateFormat.
236                         format(date), amount));
237                         test.logAction();
238
239                     }
240
241                     HistoryInvoker test = new
242                         HistoryInvoker();
243                         test.setLog(new LogCAHistory(), p,
244                         new History("Update", "CASHACCOUNT", account, dateFormat.
245                         format(date), -amount));
246                         test.logAction();
247
248                     }
249
250                     /*
251                     * Transfer money between two cash accounts
252                     */
253                     public void transfer(CashAccount c1, CashAccount c2,
254                     Double num, Portfolio p){
255                         DateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
256                         Date date = new Date();
257                         for(CashAccount c : p.cashAccList){
258                             if(c.name.equals(c2.name)){
259                                 c.balance -= num;
260
261                                 HistoryInvoker test = new HistoryInvoker(
262 );
263                                 test.setLog(new LogCAHistory(), p, new
264                                 History("Update", "CASHACCOUNT", c, dateFormat.format(date
265 ), -num));
266                                 test.logAction();

```

```
263                     break;
264                 }
265             }
266             for(CashAccount c : p.cashAccList) {
267                 if(c.name.equals(c1.name)) {
268                     c.balance += num;
269
270                     HistoryInvoker test = new HistoryInvoker(
271 );
271                     test.setLog(new LogCAHistory(), p, new
272 History("Update", "CASHACCOUNT", c, dateFormat.format(date
272 ), num));
273                     test.logAction();
274                     break;
275                 }
276             }
277
278
279
280     @Override
281     public boolean equals(Object obj) {
282         if(obj instanceof CashAccount) {
283             if(this.name.equals(((CashAccount) obj).name)
283 ) {
284                 return true;
285             }
286         }
287         return false;
288     }
289 }
290
```

```
1 package trunk.Model;
2
3 /**
4  * A LogHistory class that represents Cash Account History
5  *
6
7
8 public class LogCAHistory implements LogHistory {
9     @Override
10    public void log(Portfolio portfolio, History history)
11    {
12        portfolio.cashAccountHistory.add(history);
13    }
14}
```

```
1 package trunk.Model;
2
3 import javafx.stage.Stage;
4 import trunk.View.CAView;
5 import trunk.View.EquitiesView;
6 import trunk.View.FPTS;
7
8 /**
9  * Created by sadaf345 on 4/12/2016.
10 */
11 public interface handleCommand {
12     public void handleCommand(final Stage primaryStage,
13                             FPTS fpts, EquitiesView equitiesView, CAView caView);
14 }
```

```
1 package trunk.Model;
2
3 /**
4  * A LogHistory class that represents Equity History.
5  */
6 public class LogEquityHistory implements LogHistory {
7     @Override
8     public void log(Portfolio portfolio, History history)
9     {
10         portfolio.equityHistory.add(history);
11     }
12 }
```

```
1 package trunk.Model;  
2  
3 /**  
4   * Created by sadaf345 on 4/14/2016.  
5 */  
6 public interface WatchlistVisitor {  
7   public void visit(Portfolio p);  
8 }  
9
```

```

1 package trunk.Control;
2
3 import trunk.Model.*;
4
5 /**
6  * With this simulation, the equity prices never change.
7  * This class implements
8  * the FutureSimulation interface, and acts as a concrete
9  * strategy within the
10 * Strategy design pattern.
11 */
12 /**
13  * The method to run the simulation.
14  * @param portfolio : A Portfolio object that will run
15  * the simulation.
16  * @param percentage : A double that represents a per
17  * annum percentage.
18  * @param interval : An int that represents a period
19  * of time.
20  * @param steps : An int that represents the number of
21  * intervals to run.
22  * @return : A double representing the equity price
23  * after the simulation.
24 */
25
26 public double[] simulate(Portfolio portfolio, double
27 percentage, int interval, int steps) {
28     steps += 1;
29     double[] finalValues = new double[steps];
30     double finalValue = 0;
31     double cashValue = 0;
32
33     for (int i = 0; i < portfolio.equityList1.size();
34     i++) {
35         Equity e = portfolio.equityList1.get(i);
36         finalValue += e.shares * e.acquisitionPrice;
37     }
38
39     for (int i = 0; i < portfolio.cashAccList.size();
40     i++) {
41         CashAccount ca = portfolio.cashAccList.get(i);
42         cashValue += ca.balance;
43     }
44 }
```

```
36         finalValues[0] = finalValue + cashValue;
37         for (int j = 1; j < steps; j++) {
38             finalValues[j] = finalValue + cashValue;
39         }
40
41     return finalValues;
42 }
43
44     @Override
45     public double[] simulate(Equity e, double percentage,
int interval, int steps) {
46         double[] vals = new double[steps];
47         double initialValue = e.shares * e.acquisitionPrice;
48         vals[0] = initialValue;
49
50         for (int i = 1; i < steps; i++) {
51             vals[i] = initialValue;
52         }
53
54     return vals;
55 }
56 }
57 }
```

```
1 package trunk.Control;
2
3 import trunk.Model.Equity;
4 import trunk.Model.Portfolio;
5
6 import java.util.ArrayList;
7 import java.util.TimerTask;
8
9 public class UpdateEQ {
10
11     public void updateEquities(Portfolio portfolio,
12         ArrayList<Equity> equities) throws Exception {
13         YahooGET get = new YahooGET();
14         String yahooXML = get.yahooCall(get.formatSymbols(
15             equities));
16
17         YahooParse parse = new YahooParse();
18         ArrayList<Double> askPrice = parse.parseXML(
19             yahooXML, portfolio);
20
21         for (int i = 0; i < equities.size(); i++) {
22             equities.get(i).acquisitionPrice = askPrice.
23                 get(i);
24         }
25     }
26 }
```

```
1 package trunk.Control;
2
3 import trunk.Model.Equity;
4 import trunk.Model.Portfolio;
5
6 import java.io.BufferedReader;
7 import java.io.IOException;
8 import java.io.InputStreamReader;
9
10 import java.net.HttpURLConnection;
11 import java.net.URL;
12 import java.util.ArrayList;
13
14 public class YahooGET {
15
16     String urlCall = "";
17
18     public String yahooCall(String symbolString) throws
19         IOException {
20
21         String url =
22             String.format("http://query.yahooapis.com/
23             v1/public/yql?q=select%%20*%%20from%%20yahoo.finance.
24             quotes%%20where%%20symbol%%20in%%20(%s)&env=store://
25             datatables.org/alltableswithkeys", symbolString);
26
27         URL YahooURL = new URL(url);
28         HttpURLConnection con = (HttpURLConnection)
29             YahooURL.openConnection();
30
31         con.setRequestMethod("GET");
32
33         BufferedReader in = new BufferedReader(
34             new InputStreamReader(con.getInputStream())
35         );
36
37         String inputLine;
38         StringBuilder response = new StringBuilder();
39
40         while ((inputLine = in.readLine()) != null) {
41             response.append(inputLine);
42         }
43
44         in.close();
45
46         return response.toString();
47 }
```

```
40      }
41
42      public String formatSymbols(ArrayList<Equity> equities
43 ) {
44         int i = 0;
45
46         while( i < equities.size() ) {
47             if (i == equities.size() - 1) {
48                 urlCall += "%22";
49                 urlCall += equities.get(i).tickerSymbol;
50                 urlCall += "%22";
51             } else {
52                 urlCall += "%22";
53                 urlCall += equities.get(i).tickerSymbol;
54                 urlCall += "%22";
55                 urlCall += "%2C";
56                 urlCall += "%20";
57             }
58         }
59         return urlCall;
60     }
61
62 }
63
```

```

1 package trunk.Control;
2 import trunk.Model.*;
3
4
5 /**
6  * With this simulation, the user shall indicate a per
7  * annum percentage that each
8  * equity will decrease. This class implements the
9  * FutureSimulation interface, and
10 * acts as a concrete strategy within the Strategy design
11 * pattern.
12 */
13 /**
14  * The method to run the simulation.
15  * @param portfolio : A Portfolio object that will run
16  * the simulation.
17  * @param percentage : A double that represents a per
18  * annum percentage.
19  * @param interval : An int that represents a period
20  * of time.
21  * @param steps : An int that represents the number of
22  * intervals to run.
23  * @return : A double representing the equity price
24  * after the simulation.
25 */
26 public double[] simulate(Portfolio portfolio, double
27 percentage, int interval, int steps) {
28
29
30     for (int i = 0; i < portfolio.equityList1.size();
31 i++) {
32         Equity e = portfolio.equityList1.get(i);
33         initialValue += e.shares * e.acquisitionPrice;
34     }
35 }
```

```
36         for (int i = 0; i < portfolio.cashAccList.size();  
37             i++) {  
38             CashAccount ca = portfolio.cashAccList.get(i);  
39             cashValue += ca.balance;  
40         }  
41         initialValue += cashValue;  
42         finalValues[0] = initialValue;  
43         double sub = initialValue * (percentage / 100);  
44  
45         for (int j = 1; j < steps; j++) {  
46             if (finalValues[j - 1] - sub >= 0)  
47                 finalValues[j] = finalValues[j - 1] - sub;  
48             else finalValues[j] = 0;  
49         }  
50     }  
51     return finalValues;  
52 }  
53 }  
54  
55 @Override  
56 public double[] simulate(Equity e, double percentage,  
int interval, int steps) {  
57     double[] vals = new double[steps];  
58     double initialValue = e.shares * e.acquisitionPrice;  
59     vals[0] = initialValue;  
60  
61     for (int i = 1; i < steps; i++) {  
62         initialValue -= initialValue * (percentage / 100);  
63         if (initialValue >= 0)  
64             vals[i] = initialValue;  
65         else vals[i] = 0;  
66     }  
67  
68     return vals;  
69 }  
70 }  
71 }
```

```

1 package trunk.Control;
2
3 import trunk.Model.*;
4
5 /**
6  * With this simulation, the user shall indicate a per
7  * annum percentage that each
8  * equity will increase. This class implements the
9  * FutureSimulation interface, and
10 * acts as a concrete strategy within the Strategy design
11 * pattern.
12 */
13 public class BullMarket implements FutureSimulation {
14
15     /**
16      * The method to run the simulation.
17      * @param portfolio : A Portfolio object that will run
18      * the simulation.
19      * @param percentage : A double that represents a per
20      * annum percentage.
21      * @param interval : An int that represents a period
22      * of time.
23      * @param steps : An int that represents the number of
24      * intervals to run.
25      * @return : A double representing the equity price
26      * after the simulation.
27      */
28
29     public double[] simulate(Portfolio portfolio, double
30     percentage, int interval, int steps) {
31         steps += 1;
32         double[] finalValues = new double[steps];
33         double initialValue = 0;
34         double finalValue = 0;
35         double cashValue = 0;
36
37         for (int i = 0; i < portfolio.equityList1.size();
38             i++) {
39             Equity e = portfolio.equityList1.get(i);
40             initialValue += e.shares * e.acquisitionPrice;
41         }
42
43         for (int i = 0; i < portfolio.cashAccList.size();
44             i++) {
45             CashAccount ca = portfolio.cashAccList.get(i);
46             cashValue += ca.balance;

```

```
35         }
36
37         initialValue += cashValue;
38         finalValues[0] = initialValue;
39         double add = initialValue * (percentage / 100);
40         for (int j = 1; j < steps; j++) {
41             finalValues[j] = finalValues[j - 1] + add;
42         }
43
44         return finalValues;
45     }
46
47     @Override
48     public double[] simulate(Equity e, double percentage,
49     int interval, int steps) {
50         double[] vals = new double[steps];
51         double initialValue = e.shares * e.acquisitionPrice;
52         vals[0] = initialValue;
53
54         for (int i = 1; i < steps; i++) {
55             initialValue += initialValue * (percentage / 100);
56             vals[i] = initialValue;
57         }
58
59         return vals;
60     }
61 }
```

```
1 package trunk.Control;
2
3 import org.w3c.dom.*;
4 import org.xml.sax.InputSource;
5 import trunk.Model.Equity;
6 import trunk.Model.Portfolio;
7
8 import javax.sound.sampled.Port;
9 import javax.xml.parsers.DocumentBuilder;
10 import javax.xml.parsers.DocumentBuilderFactory;
11 import java.io.StringReader;
12 import java.util.ArrayList;
13 import java.util.List;
14 import java.util.stream.Collectors;
15
16 public class YahooParse {
17     public ArrayList<Double> parseXML(String xmlRecords,
18                                         Portfolio portfolio) throws Exception{
19         ArrayList<Double> askingPrices = new ArrayList<
20                                         Double>();
21         DocumentBuilder db = DocumentBuilderFactory.
22             newInstance().newDocumentBuilder();
23         InputSource is = new InputSource();
24         is.setCharacterStream(new StringReader(xmlRecords));
25
26         Document doc = db.parse(is);
27         NodeList nodes = doc.getElementsByTagName("quote");
28         ;
29
30         for (int i = 0; i < nodes.getLength(); i++) {
31             Element element = (Element) nodes.item(i);
32
33             NodeList askPrice = element.
34                 getElementsByTagName("Ask");
35             Element line1 = (Element) askPrice.item(0);
36
37             NodeList tickerSym = element.
38                 getElementsByTagName("Symbol");
39             Element line2 = (Element) tickerSym.item(0);
40
41             String sym = getCharacterDataFromElement(line2);
42         };
43     }
44 }
```

```
38         if (getCharacterDataFromElement(line1).isEmpty()
39             ()) {
40             List<Equity> match = portfolio.equityList2
41               .stream().filter(e -> e.tickerSymbol.equals(sym)).collect(
42                 Collectors.toList());
43             askingPrices.add(match.get(0).
44               acquisitionPrice);
45           } else{
46             askingPrices.add(Double.parseDouble(
47               getCharacterDataFromElement(line1)));
48           }
49
50         return askingPrices;
51     }
52
53   public String getCharacterDataFromElement(Element e) {
54     Node child = e.getFirstChild();
55     if (child instanceof CharacterData) {
56       CharacterData cd = (CharacterData) child;
57       return cd.getData();
58     }
59   }
```

```
1 /**
2  * A class that acts as the Invoker in the command design
3  * pattern.
4  * Gets a request from a Portfolio or a Cash Account and
5  * has LogHistory
6  * run the appropriate command.
7 */
8 package trunk.Control;
9
10 import trunk.Model.*;
11
12 public class HistoryInvoker {
13     private LogHistory event;
14     private trunk.Model.Portfolio portfolio;
15     private History history;
16
17     /**
18      * A constructor for the HistoryInvoker.
19      * Instantiates which type of log should be used.
20      * @param event : A LogHistory object representing the
21      * type of log to be used.
22      * @param portfolio : The Portfolio involved with the
23      * events occurring.
24      * @param history : A History object representing the
25      * event that occurred.
26     */
27
28     /**
29      * Sends the portfolio involved with the event that
30      * occurred and the history object
31      * representing that event to the appropriate log type
32
33     */
34
35 }
```

```
1 package trunk.Control;
2 /**
3  * Created by zachary on 3/6/2016.
4  */
5
6 import trunk.Model.*;
7
8 /**
9  * Created by zachary on 3/6/2016.
10 */
11 public interface FutureSimulation {
12     public double[] simulate(Portfolio p, double
percentage, int interval, int steps);
13     public double[] simulate(Equity e, double percentage,
int interval, int steps);
14 }
15
```