```
 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
   ArrayList<Portfolio>();
39     public Stage stage;
40     int pindex = 0;
41     public static String id;
42
43     public static void main(String[] args) {
44         //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
   FileInputStream("myfile");
52                          ObjectInputStream in = new
   ObjectInputStream(fileIn);
53                          port = (ArrayList<Portfolio>) in.
   readObject();
54                          in.close();
55                          fileIn.close();
56                      } catch (ClassNotFoundException e) {
57                          e.printStackTrace();
58                      } catch (FileNotFoundException e) {
59                          e.printStackTrace();
60                      } catch (IOException e) {
61                          e.printStackTrace();
62                      }
63                      for (Portfolio p : port) {
64                          if (p.loginID.equals(args[i+1])) {
65                              port.remove(p);
66                              try {
67                                  FileOutputStream fos = new
    FileOutputStream("myfile");
68                                  ObjectOutputStream oos =
   new ObjectOutputStream(fos);
69                                  oos.writeObject(port);
70                                  oos.close();
71                                  fos.close();
72                              } catch (IOException ioe) {
73                                  ioe.printStackTrace();
74                              }
75                          }
76                      }
77                  }
78                  i++;
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
    DateTimeStringConverter("MM/dd/YYYY");
113         try{
114             format.fromString(date);
115         } catch (Exception ex){
116             return false;
117         }
118         return true;
119     }
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(
    "myfile");
131             ObjectInputStream in = new ObjectInputStream(
    fileIn);
132             port2 = (ArrayList<Portfolio>)in.readObject()
    ;
133             in.close();
134             fileIn.close();
135         } catch (ClassNotFoundException ee) {
136             ee.printStackTrace();
137         } catch (FileNotFoundException ee) {
138             ee.printStackTrace();
139         } catch (IOException ee) {
140             ee.printStackTrace();
141         }
142         if(port2 != null) {
143             portfolios = port2;
144         }
145         portfolios.add(pindex, portfolio);
146         pindex++;
147         try {
148             FileOutputStream fos = new FileOutputStream("
    myfile");
149             ObjectOutputStream oos = new
    ObjectOutputStream(fos);
150             oos.writeObject(portfolios);
151             oos.close();
152             fos.close();
153         } catch (IOException ioe) {
154             ioe.printStackTrace();
155         }
156     }
157
158     /*
159      *  Code for basic skeleton of border pane
160      */
161     public BorderPane borderPane(Stage primaryStage,
    Portfolio p, Text t){
162         EquitiesView equitiesView = new EquitiesView();
163         CAView caView = new CAView();
164         simulationView simulationView = new
    simulationView();
165         transactionView transactionView = new
```

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

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

```
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             @Override
252             public void handle(ActionEvent event) {
253                 FileChooser fileChooser = new FileChooser
    ();
254                 fileChooser.setTitle("Open Resource File"
    );
255                 File file = fileChooser.showOpenDialog(
    stage);
256                 if(file != null){
257                     p.importP(file.toString(),p);
258                 }
259             }
260         });
261 
262         exportP.setOnAction(new EventHandler<ActionEvent>
    () {
263             @Override
264             public void handle(ActionEvent event) {
265                 final Stage dialog = new Stage();
266                 dialog.initOwner(primaryStage);
267                 GridPane grid2 = new GridPane();
268                 grid2.setAlignment(Pos.CENTER);
269                 grid2.setHgap(10);
270                 grid2.setVgap(10);
271                 grid2.setPadding(new Insets(5, 10, 5, 10)
    );
272 
273                 TextField file = new TextField();
274                 grid2.add(new Label("Filename: "),0,1);
275                 grid2.add(file,1,1);
276 
277                 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                         Portfolio pInfo = new Portfolio(
    "","");
294                         pInfo.exportP(file.getText(),p);
295                         dialog.close();
296                     }
297                 });
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<
    ActionEvent>() {
317             @Override
318             public void handle(ActionEvent event) {
319                 Scene scene = watchlistView.ViewWatchlist
    (primaryStage,p,FPTS.this);
320                 stage.setScene(scene);
321             }
322         });
323         equities.setOnAction(new EventHandler<ActionEvent
    >() {
324             @Override
325             public void handle(ActionEvent event) {
326                 Scene scene = equitiesView.ViewEquities(
    primaryStage,p,FPTS.this);
327                 stage.setScene(scene);
328             }
329         });
330         history.setOnAction(new EventHandler<ActionEvent>
    () {
331             @Override
332             public void handle(ActionEvent event) {
333                 Scene scene = transactionView.
    Transactions(primaryStage,p,FPTS.this);
334                 stage.setScene(scene);
335             }
336         });
337         cas.setOnAction(new EventHandler<ActionEvent>() {
338             @Override
339             public void handle(ActionEvent event) {
340                 Scene scene = caView.CashAccounts(
    primaryStage,p,FPTS.this);
341                 stage.setScene(scene);
342             }
343         });
344         sim.setOnAction(new EventHandler<ActionEvent>() {
345             @Override
346             public void handle(ActionEvent event) {
347                 Scene scene = simulationView.simulation(
    primaryStage, p, FPTS.this);
348                 stage.setScene(scene);
349             }
350         });
351         update.setOnAction(new EventHandler<ActionEvent>(
    ) {
```

```
352             @Override
353             public void handle(ActionEvent event) {
354                 try{
355                     UpdateEQ updateEQ = new UpdateEQ();
356                     updateEQ.updateEquities(p, p.
    equityList2);
357                 }catch(Exception ex){
358                     ex.getMessage();
359                 }
360             }
361         });
362 
363         buttonBar.getChildren().addAll(space, importP,
    exportP, spacer2, watchList, equities, history, cas,
364                 sim, update, spacer, undo, redo, logout,
    space2);
365 
366         border.setCenter(grid);
367         return border;
368     }
369     @Override
370     public void start(Stage primaryStage) {
371         stage = primaryStage;
372 
373         Scene login = login(primaryStage);
374 
375         primaryStage.setScene(login);
376         primaryStage.show();
377     }
378 
379 
380     /*
381      * GUI for login page
382      */
383     public Scene login(final Stage primaryStage){
384         primaryStage.setTitle("FPTS");
385         GridPane grid = new GridPane();
386         grid.setAlignment(Pos.CENTER);
387         grid.setHgap(10);
388         grid.setVgap(10);
389         grid.setPadding(new Insets(5, 10, 5, 10));
390 
391         Text title = new Text("Financial Portfolio
    Tracking System");
392         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(
    "myfile");
414             ObjectInputStream in = new ObjectInputStream(
    fileIn);
415             port = (ArrayList<Portfolio>) in.readObject()
    ;
416             in.close();
417             fileIn.close();
418         } catch (ClassNotFoundException e) {
419             e.printStackTrace();
420         } catch (FileNotFoundException e) {
421             e.printStackTrace();
422         } catch (IOException e) {
423             e.printStackTrace();
424         }
425         final ArrayList<Portfolio> ports = port;
426 
427         final Text error = new Text();
428         grid.add(error, 1, 6);
429         login.setOnAction(new EventHandler<ActionEvent>()
     {
430             @Override
431             public void handle(ActionEvent e) {
432                 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
    .getText().trim()) && p.password.equals(password.getText(
    ).trim())) {
440                                  Scene scene = portfolio(
    primaryStage, p);
441                                  stage.setScene(scene);
442                              }
443                          }
444                      }
445                      error.setFill(Color.FIREBRICK);
446                      error.setText("Invalid username or
    password");
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.
    NORMAL, 20));
477         grid.add(title,0,0,2,1);
478 
479         Label user = new Label("Username:");
480         grid.add(user,0,1);
481         final TextField username = new TextField();
482         grid.add(username,1,1);
483 
484         Label pass = new Label("Password:");
485         grid.add(pass,0,2);
486         PasswordField password = new PasswordField();
487         grid.add(password,1,2);
488 
489         final ToggleGroup group = new ToggleGroup();
490 
491         RadioButton rb1 = new RadioButton("Yes");
492         rb1.setToggleGroup(group);
493 
494         RadioButton rb2 = new RadioButton("No");
495         rb2.setToggleGroup(group);
496 
497         Label imp = new Label("Import Portfolio?");
498         grid.add(imp,0,3);
499         grid.add(rb1,0,4);
500         grid.add(rb2,1,4);
501 
502         final Text error = new Text();
503         grid.add(error, 1, 6);
504         Button register = new Button("Register");
505         grid.add(register,0,5);
506         register.setOnAction(new EventHandler<ActionEvent
    >() {
507 
508             @Override
509             public void handle(ActionEvent e) {
510                 final String name = username.getText().
    trim();
511                 if(username.getText().trim().isEmpty() ||
     password.getText().trim().isEmpty()){
512                     error.setFill(Color.FIREBRICK);
```

```
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
    FileInputStream("myfile");
521                         ObjectInputStream in = new
    ObjectInputStream(fileIn);
522                         port = (ArrayList<Portfolio>) in.
    readObject();
523                         in.close();
524                         fileIn.close();
525                     } catch (ClassNotFoundException ee) {
526                         ee.printStackTrace();
527                     } catch (FileNotFoundException ee) {
528                         ee.printStackTrace();
529                     } catch (IOException ee) {
530                         ee.printStackTrace();
531                     }
532                     if(port != null) {
533                         for (Portfolio p : port) {
534                             if (p.loginID.equals(username
    .getText().trim())) {
535                                 error.setFill(Color.
    FIREBRICK);
536                                 error.setText("Username
    is already taken");
537                                 usertaken = true;
538                             }
539                         }
540                     }
541                     if(usertaken == false) {
542                         final Stage dialog = new Stage();
543                         dialog.initOwner(primaryStage);
544                         addPortfolio(username.getText().
    trim(),password.getText().trim());
545 
546                         try{
547                             File file = new File("
    encryptedAccounts");
548                             FileWriter fwriter = new
    FileWriter(file.getAbsoluteFile());
```

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

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

```
610                                     equitiesView.saveEquities
    (p);
611                                     transactionView.
    saveHistory(p);
612                                     watchlistView.
    saveWatchlist(p);
613                                     p.saveMAs(p);
614                                 }
615                                 Scene scene = login(
    primaryStage);
616                                 stage.setScene(scene);
617                             }
618                             else{
619                                 Scene scene = login(
    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,
    Portfolio p){
646             Text title = new Text("Successfully logged into "
     + p.loginID + "'s Portfolio");
```

```
647         BorderPane border = borderPane(primaryStage, p,
    title);
648         Scene scene = new Scene(border, 1200, 700);
649         return scene;
650     }
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(
   fileIn);
42             port = (ArrayList<Portfolio>) in.readObject();
43             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
   ObjectOutputStream(fileOut);
64              out.writeObject(port);
65              out.close();
66              fileOut.close();
67          }catch(IOException i)
68          {
69              i.printStackTrace();
70          }
71      }
72      /*
73       * GUI for Cash Accounts page
74       */
75      public Scene CashAccounts(final Stage primaryStage,
   Portfolio p, FPTS FPTS){
76          Text title = new Text(p.loginID + "'s Cash
   Accounts");
77          BorderPane border = FPTS.borderPane(primaryStage,
   p, title);
78
79          VBox left = new VBox();
80
81          ToolBar toolBar2 = new ToolBar();
82          toolBar2.setOrientation(Orientation.VERTICAL);
83
84          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 =
    FXCollections.observableArrayList(p.cashAccList);
 92          TableView table = new TableView();
 93          table.setEditable(true);
 94
 95          vbox.getChildren().addAll(title, table);
 96
 97          TableColumn name = new TableColumn("Account Name"
    );
 98          TableColumn amount = new TableColumn("Balance");
 99          TableColumn date = new TableColumn("Date Created"
    );
100
101          name.setCellValueFactory(
102                  new PropertyValueFactory<CashAccount,
    String>("name")
103          );
104          amount.setCellValueFactory(
105                  new PropertyValueFactory<CashAccount,
    Double>("balance")
106          );
107          date.setCellValueFactory(
108                  new PropertyValueFactory<CashAccount,
    String>("date")
109          );
110          Label spacer = new Label(" ");
111          Button add = new Button("Add Account");
112          Button remove = new Button("Remove Account");
113          Button deposit = new Button("Deposit Funds");
114          Button withdrawal = new Button("Withdraw Funds");
115          Button transfer = new Button("Transfer Funds");
116          Button exportCA = new Button("Export Account");
117          Button importCA = new Button("Import Account");
118
119          toolBar2.getItems().addAll(spacer, add, remove,
    deposit, withdrawal, transfer, exportCA, importCA);
120
121          transfer.setOnAction(new EventHandler<ActionEvent
    >() {
```

```
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                 TextField to = new TextField();
133                 grid2.add(new Label("Account to Transfer
    to: "),0,1);
134                 grid2.add(to,1,1);
135 
136                 TextField from = new TextField();
137                 grid2.add(new Label("Account to Transfer
    from: "),0,2);
138                 grid2.add(from,1,2);
139 
140                 TextField num = new TextField();
141                 grid2.add(new Label("Amount to Transfer
    : "),0,3);
142                 grid2.add(num,1,3);
143                 Scene dialogScene = new Scene(grid2, 400,
     400);
144                 dialog.setScene(dialogScene);
145                 dialog.show();
146                 Button cancel = new Button("Cancel");
147                 grid2.add(cancel,1,7);
148                 Button submit = new Button("Submit");
149                 grid2.add(submit,0,7);
150                 cancel.setOnAction(new EventHandler<
    ActionEvent>() {
151                     @Override
152                     public void handle(ActionEvent e) {
153                         dialog.close();
154                     }
155                 });
156                 submit.setOnAction(new EventHandler<
    ActionEvent>() {
157                     @Override
158                     public void handle(ActionEvent event)
     {
```

```
159                         CashAccount c1 = new CashAccount(
    "","",0,"",null);
160                         CashAccount c2 = new CashAccount(
    "","",0,"",null);
161                         for(CashAccount c : p.cashAccList
    ){
162                             if(c.getName().equals(to.
    getText().trim())){
163                                 c1 = c;
164                                 break;
165                             }
166                         }
167                         for(CashAccount c : p.cashAccList
    ){
168                             if(c.getName().equals(from.
    getText().trim())){
169                                 c2 = c;
170                                 break;
171                             }
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
187                             UndoAction transferCA = new
    UndoAction("Transfer", "CashAccount", c1, c2, Double.
    parseDouble(num.getText()),null,"");
```

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

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

```
255 , 5, 10));
256                         grid2.add(title, 0, 0);
257
258                         Button ignore = new Button("
    Overwrite");
259                         Button addbtn = new Button("Add
    to Balance");
260                         Button cancel = new Button("
    Cancel");
261
262                         grid2.add(ignore, 0, 1);
263                         grid2.add(addbtn, 0, 2);
264                         grid2.add(cancel, 0, 3);
265                         ignore.setOnAction(new
    EventHandler<ActionEvent>() {
266                             @Override
267                             public void handle(
    ActionEvent event) {
268                                 CashAccount c = new
    CashAccount("", "", 0, "", p);
269                                 c.addCashAccount(exAcct.
    name, exAcct.balance, exAcct.dateCreated, p);
270
271                                 Scene scene =
    CashAccounts(primaryStage, p, FPTS);
272
273                                 FPTS.stage.setScene(scene
    );
274                                 dialog.close();
275                             }
276                         });
277                         addbtn.setOnAction(new
    EventHandler<ActionEvent>() {
278                             @Override
279                             public void handle(
    ActionEvent event) {
280                                 CashAccount c = new
    CashAccount("", "", 0, "", null);
281                                 c.updateBalance(exAcct.
    name, exAcct.balance, true, p);
282                                 Scene scene =
    CashAccounts(primaryStage, p, FPTS);
283
284                                 FPTS.stage.setScene(scene
    );
```

```
285                                 dialog.close();
286                                 dialog.close();
287                             }
288                         });
289                         cancel.setOnAction(new
    EventHandler<ActionEvent>() {
290                             @Override
291                             public void handle(
    ActionEvent event) {
292                                 dialog.close();
293                             }
294                         });
295                         Scene dialogScene = new Scene(
    grid2, 300, 200);
296                         dialog.setScene(dialogScene);
297                         dialog.show();
298                     }
299 
300                     Scene scene = CashAccounts(
    primaryStage,p, FPTS);
301                     FPTS.stage.setScene(scene);
302                 }
303             }
304         });
305 
306         //Action for depositing funds to a cash account
307         deposit.setOnAction(new EventHandler<ActionEvent>
    () {
308             @Override
309             public void handle(ActionEvent event) {
310                 ArrayList<String> choices = new ArrayList
    <>();
311                 for(CashAccount e : p.cashAccList){
312                     if(!e.name.equals("")) {
313                         choices.add(e.name);
314                     }
315                 }
316                 ChoiceDialog<String> dialog = new
    ChoiceDialog<>("None", choices);
317                 dialog.setContentText("Choose the cash
    account to deposit to:");
318 
319                 Optional<String> result = dialog.
    showAndWait();
320                 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,
    10));
328
329                     VBox dialogVbox = new VBox(20);
330                     dialogVbox.getChildren().add(new Text
    ("This is a Dialog"));
331                     TextField shares = new TextField();
332
333                     grid2.add(new Label("Deposit to " +
    result.get() + ": "),0,1);
334                     grid2.add(shares,1,1);
335                     Scene dialogScene = new Scene(grid2,
    400, 400);
336                     Button cancel = new Button("Cancel");
337                     grid2.add(cancel,1,3);
338                     Button submit = new Button("Submit");
339                     grid2.add(submit,0,3);
340                     cancel.setOnAction(new EventHandler<
    ActionEvent>() {
341                         @Override
342                         public void handle(ActionEvent e)
    {dialog2.close();}
343                     });
344                     final Text error = new Text();
345                     grid2.add(error, 1, 8);
346                     submit.setOnAction(new EventHandler<
    ActionEvent>() { // this one
347                         @Override
348                         public void handle(ActionEvent
    event) {
349                             if(shares.getText().isEmpty()
    ){
350                                 error.setFill(Color.
    FIREBRICK);
351                                 error.setText("Fill in
    all fields");
352                             }
353                             else if(!FPTS.isInt(shares.
    getText()))
```

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

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

```
421                     });
422                     final Text error = new Text();
423                     grid2.add(error, 1, 8);
424                     submit.setOnAction(new EventHandler<
    ActionEvent>() { // this one
425                         @Override
426                         public void handle(ActionEvent
    event) {
427                             if(amount.getText().isEmpty()
    ){
428                                 error.setFill(Color.
    FIREBRICK);
429                                 error.setText("Fill in
    all fields");
430                             }
431                             else if(!FPTS.isInt(amount.
    getText()))
432                             {
433                                 error.setFill(Color.
    FIREBRICK);
434                                 error.setText("Enter
    information in correct format");
435                             }
436                             else if(Integer.parseInt(
    amount.getText()) < 0){
437                                 error.setFill(Color.
    FIREBRICK);
438                                 error.setText("Enter
    valid number");
439                             }
440                             else{
441                                 int amountInt = Integer.
    parseInt(amount.getText());
442                                 for(CashAccount e : p.
    cashAccList){
443                                     if(e.name.equals(
    result.get())){
444                                         if(e.balance >
    amountInt) {
445                                             e.balance -=
    amountInt;
446
447                                             dialog2.close
    ();
448                                             Scene scene =
```

```
448  CashAccounts(primaryStage,p, FPTS);
449                                                 FPTS.stage.
    setScene(scene);
450                                             } else{
451                                                 error.setFill
    (Color.FIREBRICK);
452                                                 error.setText
    ("Cannot overdraw account");
453                                             }
454                                         }
455                                     }
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<
    ActionEvent>() {
490                     @Override
491                     public void handle(ActionEvent e) {
492                         dialog.close();
493                     }
494                 });
495                 submit.setOnAction(new EventHandler<
    ActionEvent>() {
496                     @Override
497                     public void handle(ActionEvent event)
     {
498                         CashAccount c = new CashAccount(
    "","",0,"",null);
499                         c.exportCA(file.getText(),p);
500                         dialog.close();
501                     }
502                 });
503             }
504         });
505 
506         //Action for adding a cash account
507         add.setOnAction(new EventHandler<ActionEvent>() {
508             @Override
509             public void handle(ActionEvent event) {
510                 final Stage dialog = new Stage();
511                 dialog.initOwner(primaryStage);
512                 GridPane grid2 = new GridPane();
513                 grid2.setAlignment(Pos.CENTER);
514                 grid2.setHgap(10);
515                 grid2.setVgap(10);
516                 grid2.setPadding(new Insets(5, 10, 5, 10)
    );
517                 Label name = new Label("Name:");
518                 grid2.add(name,0,1);
519                 final TextField CAname = new TextField();
520                 grid2.add(CAname,1,1);
521 
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<
    ActionEvent>() {
537                     @Override
538                     public void handle(ActionEvent event)
     {
539                         boolean same = false;
540                         for(CashAccount c : p.cashAccList
    ){
541                             if(c.getName().equals(CAname.
    getText())){
542                                 same = true;
543                                 break;
544                             }
545                         }
546                         if(Double.parseDouble(Iamount.
    getText()) > 0) {
547                             if (!same) {
548                                 CashAccount c = new
    CashAccount("", "", 0, "", p);
549                                 c.addCashAccount(CAname.
    getText(), Double.parseDouble(Iamount.getText()), CAdate.
    getText(), p);
550                                 CashAccount temp = null;
551                                 for (CashAccount n : p.
    cashAccList) {
552                                     if (n.getName().
    equals(CAname.getText())) {
553                                         temp = n;
554                                     }
555                                 }
556                                 UndoAction add = new
    UndoAction("Add", "CashAccount", temp,null,0,null,"");
557                                 add.redo.clear();
558                                 add.addToQueue(add);
559 
560                                 Scene scene =
```

```
560 CashAccounts(primaryStage, p, FPTS);
561
562                                     FPTS.stage.setScene(scene
    );
563                                     dialog.close();
564                                 } else if (same) {
565                                     dialog.close();
566                                     Text title = new Text("
    Account already exists");
567                                     final Stage dialog = new
    Stage();
568                                     dialog.initOwner(
    primaryStage);
569                                     GridPane grid2 = new
    GridPane();
570                                     grid2.setAlignment(Pos.
    CENTER);
571                                     grid2.setHgap(10);
572                                     grid2.setVgap(10);
573                                     grid2.setPadding(new
    Insets(5, 10, 5, 10));
574                                     grid2.add(title, 0, 0);
575
576                                     Button ignore = new
    Button("Overwrite");
577                                     Button add = new Button("
    Add to Balance");
578                                     Button cancel = new
    Button("Cancel");
579
580                                     grid2.add(ignore, 0, 1);
581                                     grid2.add(add, 0, 2);
582                                     grid2.add(cancel, 0, 3);
583                                     ignore.setOnAction(new
    EventHandler<ActionEvent>() {
584                                         @Override
585                                         public void handle(
    ActionEvent event) {
586                                             CashAccount c =
    new CashAccount("", "", 0, "", p);
587                                             c.addCashAccount(
    CAname.getText(), Double.parseDouble(Iamount.getText()),
    CAdate.getText(), p);
588
589                                             Scene scene =
```

```
589 CashAccounts(primaryStage, p, FPTS);
590
591                                         FPTS.stage.
    setScene(scene);
592                                         dialog.close();
593                                     }
594                                 });
595                                 add.setOnAction(new
    EventHandler<ActionEvent>() {
596                                     @Override
597                                     public void handle(
    ActionEvent event) {
598                                         CashAccount c =
    new CashAccount("", "", 0, "", null);
599                                         c.updateBalance(
    CAname.getText(), Double.parseDouble(Iamount.getText()),
    true, p);
600                                         Scene scene =
    CashAccounts(primaryStage, p, FPTS);
601
602                                         FPTS.stage.
    setScene(scene);
603                                         dialog.close();
604                                         dialog.close();
605                                     }
606                                 });
607                                 cancel.setOnAction(new
    EventHandler<ActionEvent>() {
608                                     @Override
609                                     public void handle(
    ActionEvent event) {
610                                         dialog.close();
611                                     }
612                                 });
613                                 Scene dialogScene = new
    Scene(grid2, 300, 200);
614                                 dialog.setScene(
    dialogScene);
615                                 dialog.show();
616                             }
617                         } else {
618                             error.setFill(Color.FIREBRICK
    );
619                             error.setText("Enter a valid
    amount of money");
```

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

```
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          {
61              FileOutputStream fileOut =
62                      new FileOutputStream("myfile");
63              ObjectOutputStream out = new
    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,
    FilteredList<Equity> filteredData2, TextField tickerFilter
    , TextField nameFilter, TextField maFilter, String
    newValue){
77          filteredData.setPredicate(equity -> {
78              if (newValue == null || newValue.isEmpty()) {
79                  return true;
80              }
81              boolean exists = false;
82              if(equity.getMarketAverage().isEmpty() &&
    maFilter.getText().equals("")){
83                  exists = true;
```

```
 84             }
 85
 86             for(String s : equity.getMarketAverage()){
 87                 if(s.toLowerCase().contains(maFilter.
    getText().toLowerCase())){
 88                     exists = true;
 89                 }
 90             }
 91             if(equity.getTickerSymbol().toLowerCase().
    contains(tickerFilter.getText().toLowerCase())
 92                     && equity.getName().toLowerCase().
    contains(nameFilter.getText().toLowerCase())
 93                     && exists
 94                     ){
 95                 return true;
 96             }
 97             else{
 98                 return false;
 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() &&
    maFilter.getText().equals("")){
107                 exists = true;
108             }
109
110             for(String s : equity.getMarketAverage()){
111                 if(s.toLowerCase().contains(maFilter.
    getText().toLowerCase())){
112                     exists = true;
113                 }
114             }
115             if(equity.getTickerSymbol().toLowerCase().
    contains(tickerFilter.getText().toLowerCase())
116                     && equity.getName().toLowerCase().
    contains(nameFilter.getText().toLowerCase())
117                     && exists
118                     ){
119                 return true;
120             }
121             else{
```

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

```
158         TableColumn symbols2 = new TableColumn("Ticker
    Symbol");
159         TableColumn names2 = new TableColumn("Names");
160         TableColumn salePrice2 = new TableColumn("Sale
    Price");
161         TableColumn MarketAverage2 = new TableColumn("
    Market Average");
162 
163         Label spacer = new Label(" ");
164 
165         TableView tableAvailableE = new TableView();
166         tableAvailableE.setEditable(true);
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,
    availableE, tableAvailableE);
172         border.setLeft(left);
173         final ObservableList<Equity> equities1 =
    FXCollections.observableArrayList(p.equityList1);
174         final ObservableList<Equity> equities2 =
    FXCollections.observableArrayList(p.equityList2);
175 
176         symbols.setCellValueFactory(
177                 new PropertyValueFactory<Equity,String>("
    tickerSymbol")
178         );
179         names.setCellValueFactory(
180                 new PropertyValueFactory<Equity,String>("
    name")
181         );
182         shares.setCellValueFactory(
183                 new PropertyValueFactory<Equity,Integer>(
    "shares")
184         );
185         salePrice.setCellValueFactory(
186                 new PropertyValueFactory<Equity,Double>("
    acquisitionPrice")
187         );
188         acquisitionDate.setCellValueFactory(
189                new PropertyValueFactory<Equity,Date>("
    acquisitionDate")
190         );
191         MarketAverage.setCellValueFactory(
```

```
192                 new PropertyValueFactory<Equity,String>("
    MA")
193         );
194         totalValue.setCellValueFactory(
195                 new PropertyValueFactory<Equity, Double>(
    "total")
196         );
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,
    shares, salePrice, acquisitionDate, MarketAverage,
    totalValue);
215         symbols.prefWidthProperty().bind(tableOwnedE.
    widthProperty().multiply(0.1));
216         names.prefWidthProperty().bind(tableOwnedE.
    widthProperty().multiply(0.35));
217         shares.prefWidthProperty().bind(tableOwnedE.
    widthProperty().multiply(0.05));
218         salePrice.prefWidthProperty().bind(tableOwnedE.
    widthProperty().multiply(0.075));
219         acquisitionDate.prefWidthProperty().bind(
    tableOwnedE.widthProperty().multiply(0.1));
220         MarketAverage.prefWidthProperty().bind(
    tableOwnedE.widthProperty().multiply(0.15));
221         totalValue.prefWidthProperty().bind(tableOwnedE.
    widthProperty().multiply(0.125));
```

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

```
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
    MA");
262
263         Button goback = new Button("View Portfolio");
264
265         //Action for importing equities
266         importEq.setOnAction(new EventHandler<ActionEvent
    >() {
267             @Override
268             public void handle(ActionEvent event) {
269                 FileChooser fileChooser = new FileChooser
    ();
270                 fileChooser.setTitle("Open Resource File"
    );
271                 File file = fileChooser.showOpenDialog(
    FPTS.stage);
272                 if(file != null){
273                     Equity e = new Equity("","",0,0,"",
    null);
274                     e.importEQ(file.toString(),p);
275                     UndoAction add = new UndoAction("
    Import", "Equity", null,null,0,null,file.toString());
276                     add.redo.clear();
277                     add.addToQueue(add);
278                     try{
279                         UpdateEQ updateEQ = new UpdateEQ(
    );
280                         updateEQ.updateEquities(p, p.
    equityList2);
281                     }catch(Exception ex){
282                         ex.getMessage();
283                     }
284                     Scene scene = ViewEquities(
    primaryStage,p, FPTS);
285                     FPTS.stage.setScene(scene);
286                 }
287             }
288         });
289
```

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

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

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

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

```
427                                 SIlist.getChildren().forEach(
    (SI) -> {
428                                     SIarray.add(((TextField)
    SI).getText());
429                                 });
430                                 Equity e = new Equity(
    tickerSymbol.getText(),names.getText(),Integer.parseInt(
    numShares.getText()),Double.parseDouble(salePrice.getText
    ()),
431                                         acquisitionDate.
    getText(),SIarray);
432                                 e.addEquity(tickerSymbol.
    getText(), names.getText(), Integer.parseInt(numShares.
    getText()),
433                                         Double.parseDouble(
    salePrice.getText()), acquisitionDate.getText() , SIarray
    ,false,p);
434
435
436                                 UndoAction add = new
    UndoAction("Add", "Equity",null,null,0,e,"");
437                                 add.redo.clear();
438                                 add.addToQueue(add);
439
440
441                                 dialog.close();
442                                 Scene scene = ViewEquities(
    primaryStage,p, FPTS);
443                                 FPTS.stage.setScene(scene);
444
445                             }
446
447                         }
448
449                 });
450             }
451         });
452
453         //Action for removing equities
454         remove.setOnAction(new EventHandler<ActionEvent>(
    ) {
455             @Override
456             public void handle(ActionEvent event) {
457                 ArrayList<String> choices = new ArrayList
    <>();
```

```
458                 ArrayList<String> cashAccts = new
    ArrayList<>();
459                 for(Equity e : p.equityList1){
460                     if(!e.tickerSymbol.equals("")) {
461                         choices.add(e.tickerSymbol);
462                     }
463                 }
464                 for(CashAccount c : p.cashAccList){
465                     cashAccts.add(c.name);
466                 }
467 
468                 ChoiceDialog<String> dialog = new
    ChoiceDialog<>("None", choices);
469                 dialog.setContentText("Choose the equity
    you wish to remove:");
470 
471                 Optional<String> result = dialog.
    showAndWait();
472                 if (result.isPresent()) {
473                     ChoiceDialog<String> addCash = new
    ChoiceDialog<>("None", cashAccts);
474                     addCash.setContentText("Choose cash
    account to add equity value to:");
475                     Optional<String> cashAcct = addCash.
    showAndWait();
476                     Equity e = new Equity("","",0,0,"",
    null);
477                     for (Equity n : p.equityList1) {
478                         if (n.getTickerSymbol().equals(
    result.get())) {
479                             e = n;
480                         }
481                     }
482 
483 
484                     UndoAction add = new UndoAction("
    Remove", "Equity",null,null,0,e,"");
485                     add.redo.clear();
486                     add.addToQueue(add);
487                     e.removeEquity(result.get(), cashAcct
    .get(),p);
488                 }
489 
490                 Scene scene = ViewEquities(primaryStage,p
    , FPTS);
```

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

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

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

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

```
607 marketAverage.get(0).equals(result.get())){
608                                             ArrayList<String>
     cashAccts = new ArrayList<>();
609                                             for(CashAccount c
     : p.cashAccList){
610                                                 cashAccts.add
    (c.name);
611                                             }
612                                             ChoiceDialog<
    String> addCash = new ChoiceDialog<>("None", cashAccts);
613                                             addCash.
    setContentText("Choose cash account to pay for shares:");
614                                             Optional<String>
    cashAcct = addCash.showAndWait();
615                                             CashAccount c =
    new CashAccount("","",0,"",null);
616                                             tot = numShares *
     stuff.acquisitionPrice;
617                                             if(cashAcct.
    isPresent()) {
618                                                 for(
    CashAccount acct : p.cashAccList) {
619                                                     if(acct.
    name.equals(cashAcct.get())) {
620                                                         if(
    acct.balance >= tot) {
621                                                             c
    .updateBalance(cashAcct.get(), tot, false, p);
622
    Equity e = new Equity("","",0,0,"",null);
623                                                             e
    .addEquity(stuff.tickerSymbol,stuff.name,numShares,stuff.
    acquisitionPrice,
624

           stuff.acquisitionDate,stuff.getMarketAverage(),
    false,p);
625
    addCash.close();
626
    dialog2.close();
627
    Scene scene = ViewEquities(primaryStage,p, FPTS);
628
    FPTS.stage.setScene(scene);
```

```
629                                                         }
    else{
630
    error.setFill(Color.FIREBRICK);
631
    error.setText("Account contains too little funds.");
632                                                         }
633                                                     }
634                                                 }
635                                             }
636                                             else{
637                                                 error.setFill
    (Color.FIREBRICK);
638                                                 error.setText
    ("Select a cash account");
639                                             }
640                                             addCash.close();
641                                             dialog2.close();
642                                             Scene scene =
    ViewEquities(primaryStage,p, FPTS);
643                                             FPTS.stage.
    setScene(scene);
644                                         }
645                                     }
646
647
648                                     dialog2.close();
649                                     Scene scene =
    ViewEquities(primaryStage,p, FPTS);
650                                     FPTS.stage.setScene(scene
    );
651                                 }
652                             }
653                     });
654
655                     dialog2.setScene(dialogScene);
656                     dialog2.show();
657                 }
658             }
659         });
660         sellShares.setOnAction(new EventHandler<
    ActionEvent>() {
661             @Override
662             public void handle(ActionEvent event) {
663                 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
    ChoiceDialog<>("None", choices);
672                 dialog.setContentText("Choose the equity
    you wish to sell shares from:");
673
674
675                 Optional<String> result = dialog.
    showAndWait();
676                 if (result.isPresent()) {
677                     int num1 = 0;
678                     for (Equity stuff : p.equityList1) {
679                         if (stuff.getTickerSymbol().
    equals(result.get())) {
680                             num1 = stuff.shares;
681                         }
682                         else if(stuff.tickerSymbol.equals
    ("")
683                                 && stuff.marketAverage.
    get(0).equals(result.get())){
684                             num1 = stuff.shares;
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,
     10));
695
696                     VBox dialogVbox = new VBox(20);
697                     dialogVbox.getChildren().add(new Text
    ("This is a Dialog"));
698                     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.isInt(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
```

```
728 value");
729                                 }
730                                 else {
731                                     Equity e = null;
732                                     double tot = 0;
733                                     double amount = 0;
734                                     for (Equity stuff : p.
    equityList1) {
735                                         if (stuff.
    getTickerSymbol().equals(result.get())) {
736                                             tot = numShares *
     stuff.acquisitionPrice;
737                                             stuff.shares -=
    numShares;
738                                             amount = -(
    numShares * stuff.acquisitionPrice);
739                                             e = stuff;
740                                         }
741                                         else if(stuff.
    tickerSymbol.equals("")
742                                                 && stuff.
    marketAverage.get(0).equals(result.get())){
743                                             tot = numShares *
     stuff.acquisitionPrice;
744                                             stuff.shares -=
    numShares;
745                                             amount = -(
    numShares * stuff.acquisitionPrice);
746                                             e = stuff;
747                                         }
748                                     }
749
750                                     ArrayList<String>
    cashAccts = new ArrayList<>();
751                                     for(CashAccount c : p.
    cashAccList){
752                                         cashAccts.add(c.name)
    ;
753                                     }
754                                     ChoiceDialog<String>
    addCash = new ChoiceDialog<>("None", cashAccts);
755                                     addCash.setContentText("
    Choose cash account to add shares to:");
756                                     Optional<String> cashAcct
     = addCash.showAndWait();
```

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

```
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 "
    + result.get() + ": "),0,1);
826                     grid2.add(shares,1,1);
827                     Scene dialogScene = new Scene(grid2,
    400, 400);
828                     Button cancel = new Button("Cancel");
829                     grid2.add(cancel,1,3);
830                     Button submit = new Button("Submit");
831                     grid2.add(submit,0,3);
832                     cancel.setOnAction(new EventHandler<
    ActionEvent>() {
833                         @Override
834                         public void handle(ActionEvent e)
     {dialog2.close();}
835                     });
836                     final Text error = new Text();
837                     grid2.add(error, 1, 8);
838                     submit.setOnAction(new EventHandler<
    ActionEvent>() { // this one
839                         @Override
840                         public void handle(ActionEvent
    event) {
841                             if(shares.getText().isEmpty()
    ){
842                                 error.setFill(Color.
    FIREBRICK);
843                                 error.setText("Fill in
    all fields");
844                             }
845                             else if(!FPTS.isInt(shares.
    getText()))
846                             {
847                                 error.setFill(Color.
    FIREBRICK);
848                                 error.setText("Enter
    information in correct format");
849                             }
850                             else if(Integer.parseInt(
    shares.getText()) < 1){
851                                 error.setFill(Color.
    FIREBRICK);
852                                 error.setText("Enter
```

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

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

```
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
   ObjectOutputStream(fileOut);
61              out.writeObject(port);
62              out.close();
63              fileOut.close();
64          }catch(IOException i)
65          {
66              i.printStackTrace();
67          }
68      }
69
70      /*
71       * GUI for Watchlist page
72       */
73      public Scene ViewWatchlist(Stage primaryStage,
   Portfolio p, FPTS FPTS){
74          Text title = new Text(p.loginID + "'s Watchlist");
75          BorderPane border = FPTS.borderPane(primaryStage,
   p, title);
76          VBox left = new VBox(5);
77
78          ToolBar toolBar2 = new ToolBar();
79          toolBar2.setOrientation(Orientation.VERTICAL);
80          left.getChildren().addAll(toolBar2);
81          left.setSpacing(10);
82
83          final Label watchItems = new Label(p.loginID + "'s
    Watchlist");
84          watchItems.setFont(new Font("Arial", 20));
```

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

```
119         compare.setCellValueFactory(
120                 new PropertyValueFactory<Watchlist,
    String>("compare")
121         );
122         lowtriggered.setCellValueFactory(
123                 new PropertyValueFactory<Watchlist,
    String>("lowtriggered")
124         );
125         hightriggered.setCellValueFactory(
126                 new PropertyValueFactory<Watchlist,
    String>("hightriggered")
127         );
128
129
130
131         //watchList.setItems(equities1);
132         watchList.getColumns().addAll(symbols,
    hightrigger, lowtrigger,price,compare, hightriggered,
    lowtriggered);
133         watchList.setItems(watch);
134
135
136         symbols.prefWidthProperty().bind(watchList.
    widthProperty().multiply(0.1));
137         lowtrigger.prefWidthProperty().bind(watchList.
    widthProperty().multiply(0.075));
138         hightrigger.prefWidthProperty().bind(watchList.
    widthProperty().multiply(0.15));
139
140
141         Button add = new Button("Add Item");
142         Button remove = new Button("Remove Item");
143         Button high = new Button("Add High Trigger");
144         Button low = new Button("Add Low Trigger");
145         Button reset = new Button("Reset");
146
147
148         //Action for adding equities
149         add.setOnAction(new EventHandler<ActionEvent>() {
150             @Override
151             public void handle(ActionEvent event) {
152                 ArrayList<String> choices = new ArrayList
    <>();
153                 double price1 = 0;
154                 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
    ChoiceDialog<>("None", choices);
162                 dialog.setContentText("Choose the equity
    you wish to add to the watchlist:");
163 
164                 Optional<String> result = dialog.
    showAndWait();
165                 if (result.isPresent()) {
166                     for(Equity e : p.equityList2){
167                         if(e.tickerSymbol.equals(result.
    get())) {
168                             price1 = e.acquisitionPrice;
169                         }
170                     }
171                     final double price = price1;
172                     final Stage dialog2 = new Stage();
173                     dialog2.initOwner(primaryStage);
174                     GridPane grid2 = new GridPane();
175                     grid2.setAlignment(Pos.CENTER);
176                     grid2.setHgap(10);
177                     grid2.setVgap(10);
178                     grid2.setPadding(new Insets(5, 10, 5,
     10));
179 
180                     VBox dialogVbox = new VBox(20);
181                     dialogVbox.getChildren().add(new Text
    ("This is a Dialog"));
182                     TextField low = new TextField();
183                     TextField high = new TextField();
184 
185                     grid2.add(new Label("Low Trigger: "),
     0, 1);
186                     grid2.add(low, 1, 1);
187                     grid2.add(new Label("High Trigger: ")
    , 0, 2);
188                     grid2.add(high, 1, 2);
189                     Scene dialogScene = new Scene(grid2,
    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<
    ActionEvent>() {
195                         @Override
196                         public void handle(ActionEvent e)
     {
197                             dialog2.close();
198                         }
199                     });
200                     final Text error = new Text();
201                     grid2.add(error, 1, 8);
202                     submit.setOnAction(new EventHandler<
    ActionEvent>() { // this one
203                         @Override
204                         public void handle(ActionEvent
    event) {
205                             Watchlist e = new Watchlist(
    result.get(), low.getText(), high.getText(), price, ""," "
    ,"");
206                             p.watchlists.add(e);
207                             e.visit(p);
208
209
210                             dialog2.close();
211                             Scene scene = ViewWatchlist(
    primaryStage, p, FPTS);
212                             FPTS.stage.setScene(scene);
213                         }
214                     });
215
216                     dialog2.setScene(dialogScene);
217                     dialog2.show();
218                 }
219             }
220         });
221
222         //Action for removing equities
223         remove.setOnAction(new EventHandler<ActionEvent>(
    ) {
224             @Override
225             public void handle(ActionEvent event) {
226                 ArrayList<String> choices = new ArrayList
```

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

```
261                         if(e.tickerSymbol.equals(result.
    get())){
262                             lower = e.lowtrigger;
263                         }
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,
     10));
274 
275                     VBox dialogVbox = new VBox(20);
276                     dialogVbox.getChildren().add(new Text
    ("This is a Dialog"));
277                     TextField high = new TextField();
278 
279                     grid2.add(new Label("High Trigger: ")
    , 0, 2);
280                     grid2.add(high, 1, 2);
281                     Scene dialogScene = new Scene(grid2,
    400, 400);
282                     Button cancel = new Button("Cancel");
283                     grid2.add(cancel, 1, 3);
284                     Button submit = new Button("Submit");
285                     grid2.add(submit, 0, 3);
286                     cancel.setOnAction(new EventHandler<
    ActionEvent>() {
287                         @Override
288                         public void handle(ActionEvent e)
     {
289                             dialog2.close();
290                         }
291                     });
292                     final Text error = new Text();
293                     grid2.add(error, 1, 8);
294                     submit.setOnAction(new EventHandler<
    ActionEvent>() { // this one
295                         @Override
296                         public void handle(ActionEvent
    event) {
```

```
297                                 if (high.getText().isEmpty())
     {
298                                     error.setFill(Color.
    FIREBRICK);
299                                     error.setText("Fill in
    all fields");
300                                 } else if (!FPTS.isDouble(
    high.getText())) {
301                                     error.setFill(Color.
    FIREBRICK);
302                                     error.setText("Enter
    information in correct format");
303                                 }
304                                 else {
305                                     for (Watchlist e : p.
    watchlists) {
306                                         if(!truelow.equals(""
    )) {
307                                             if (Double.
    parseDouble(high.getText()) < Double.parseDouble(truelow)
    ) {
308                                                 error.setFill
    (Color.FIREBRICK);
309                                                 error.setText
    ("High trigger can't be lower than low trigger");
310                                                 break;
311                                             }
312                                         }
313                                         if (e.tickerSymbol.
    equals(result.get())) {
314                                             e.hightrigger =
    high.getText().trim();
315                                             e.visit(p);
316
317                                             dialog2.close();
318                                             Scene scene =
    ViewWatchlist(primaryStage, p, FPTS);
319                                             FPTS.stage.
    setScene(scene);
320                                         }
321                                     }
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                 for(Watchlist e : p.watchlists){
340                     choices.add(e.tickerSymbol);
341                 }
342                 ChoiceDialog<String> dialog = new
    ChoiceDialog<>("None", choices);
343                 dialog.setContentText("Choose the
    watchlist item you wish to add a low trigger to:");
344                 Optional<String> result = dialog.
    showAndWait();
345                 if (result.isPresent()) {
346                     String higher = "";
347                     for(Watchlist e : p.watchlists){
348                         if(e.tickerSymbol.equals(result.
    get())){
349                             higher = e.hightrigger;
350                         }
351                     }
352                     final String truehigh = higher;
353
354                     final Stage dialog2 = new Stage();
355                     dialog2.initOwner(primaryStage);
356                     GridPane grid2 = new GridPane();
357                     grid2.setAlignment(Pos.CENTER);
358                     grid2.setHgap(10);
359                     grid2.setVgap(10);
360                     grid2.setPadding(new Insets(5, 10, 5,
     10));
361
362                     VBox dialogVbox = new VBox(20);
363                     dialogVbox.getChildren().add(new Text
    ("This is a Dialog"));
```

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

```
394                                                 error.setFill
    (Color.FIREBRICK);
395                                                 error.setText
    ("Low trigger can't be higher than high trigger");
396                                                 break;
397                                             }
398                                         }
399                                         if (e.tickerSymbol.
    equals(result.get())) {
400                                             e.lowtrigger =
    low.getText().trim();
401
402                                             e.visit(p);
403
404                                             dialog2.close();
405                                             Scene scene =
    ViewWatchlist(primaryStage, p, FPTS);
406                                             FPTS.stage.
    setScene(scene);
407                                         }
408                                     }
409                                 }
410
411                             }
412
413                     });
414
415                     dialog2.setScene(dialogScene);
416                     dialog2.show();
417                 }
418             }
419         });
420         reset.setOnAction(new EventHandler<ActionEvent>()
     {
421             @Override
422             public void handle(ActionEvent event) {
423                 ArrayList<String> choices = new ArrayList
    <>();
424                 for(Watchlist e : p.watchlists){
425                     choices.add(e.tickerSymbol);
426                 }
427                 ChoiceDialog<String> dialog = new
    ChoiceDialog<>("None", choices);
428                 dialog.setContentText("Choose the
    watchlist item you wish to reset:");
```

```
429                 Optional<String> result = dialog.
    showAndWait();
430                 if (result.isPresent()) {
431                     for(Watchlist w : p.watchlists){
432                         if(w.tickerSymbol.equals(result.
    get())) {
433                             w.lowtriggered = "";
434                             w.hightriggered = "";
435                         }
436                     }
437                     Scene scene = ViewWatchlist(
    primaryStage, p, FPTS);
438                     FPTS.stage.setScene(scene);
439                 }
440             }
441         });
442 
443 
444         toolBar2.getItems().addAll(spacer, add, remove,
    high, low, reset);
445 
446 
447         border.setCenter(vbox);
448         Scene scene = new Scene(border, 1200, 700);
449         return scene;
450     }
451 }
452 
453 
```

```
 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,
   Portfolio p, FPTS FPTS) {
29         Text title = new Text("Simulation");
30         BorderPane border = FPTS.borderPane(primaryStage,p
   ,title);
31
32         GridPane grid = new GridPane();
33         grid.setAlignment(Pos.TOP_CENTER);
34         grid.setHgap(10);
35         grid.setVgap(10);
36         grid.setPadding(new Insets(5, 10, 5, 10));
37         title.setFont(Font.font("Calibri", FontWeight.
   NORMAL, 20));
38         grid.add(title,0,0,2,1);
39
40         Label type = new Label("Simulation Type:");
41         grid.add(type,0,1);
42         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.
    isDouble(s.getText())){
 85                     Text error = new Text("Correctly fill
     in all fields");
 86                     error.setFill(Color.RED);
 87                     grid.add(error,0,10);
 88                 }
 89                 else {
 90                     String type = typeBox.
    getSelectionModel().getSelectedItem().toString();
 91                     String interval = intervalBox.
    getSelectionModel().getSelectedItem().toString();
 92                     int time = 365;
 93 
 94                     final NumberAxis yAxis = new
    NumberAxis();
 95                     final NumberAxis xAxis = new
    NumberAxis();
 96                     final LineChart<Number,Number>
    lineChart =
 97                             new LineChart<Number,Number>(
    xAxis,yAxis);
 98 
 99                     lineChart.setTitle("Portfolio Value
    Simulation");
100 
101                     XYChart.Series series1 = new XYChart.
    Series();
102                     series1.setName("Portfolio Value");
103 
104 
105                     if(interval.equals("Days")){
106                         time = 1;
107                     }
108                     else if(interval.equals("Months")){
109                         time = 30;
110                     }
111                     else if(interval.equals("Years")){
112                         time = 365;
113                     }
114 
115 
116 
117                     if (type.equals("Bull Market")) {
118                         BullMarket bullMarket = new
```

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

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

```
180                                 });
181
182                                 skip.setOnAction(new
    EventHandler<ActionEvent>() {
183                                     @Override
184                                     public void handle(
    ActionEvent event) {
185                                         for (int i = 1; i <
    val.length; i++) {
186                                             series1.getData()
    .add(new XYChart.Data(i, val[i]));
187                                         }
188                                     }
189                                 });
190
191                             }
192                             else {
193                                 for (int i = 1; i < val.
    length; i++) {
194                                     series1.getData().add(new
     XYChart.Data(i, val[i]));
195                                 }
196                             }
197
198                         }
199
200                         else if (type.equals("No Growth")) {
201                             NoGrowth noGrowth = new NoGrowth(
    );
202                             double[] val = noGrowth.simulate(
    p,Double.parseDouble(pp.getText().trim()),time,Integer.
    parseInt(s.getText().trim()));
203
204                             series1.getData().add(new XYChart
    .Data(0, val[0]));
205
206                             if (cb.isSelected()) {
207                                 Button next = new Button("
    Next Step");
208                                 grid.add(next, 0, 7);
209                                 Button skip = new Button(">>
    Final Step");
210                                 grid.add(skip, 1, 7);
211                                 VBox data = new VBox();
212                                 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)
220                                         i++;
221                                 }
222                             });
223 
224                             skip.setOnAction(new
    EventHandler<ActionEvent>() {
225                                 @Override
226                                 public void handle(
    ActionEvent event) {
227                                     for (int i = 1; i <
    val.length; i++) {
228                                         series1.getData()
    .add(new XYChart.Data(i, val[i]));
229                                     }
230                                 }
231                             });
232 
233                         }
234                         else {
235                             for (int i = 1; i < val.
    length; i++) {
236                                 series1.getData().add(new
     XYChart.Data(i, val[i]));
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(
   fileIn);
38             port = (ArrayList<Portfolio>) in.readObject();
39             in.close();
40             fileIn.close();
41         } catch (ClassNotFoundException ee) {
42             ee.printStackTrace();
43         } catch (FileNotFoundException ee) {
```

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

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

```
117             port = (ArrayList<Portfolio>) in.readObject()
    ;
118             in.close();
119             fileIn.close();
120         } catch (ClassNotFoundException ee) {
121             ee.printStackTrace();
122         } catch (FileNotFoundException ee) {
123             ee.printStackTrace();
124         } catch (IOException ee) {
125             ee.printStackTrace();
126         }
127         port.forEach(account -> {
128             account.equityHistory = p.equityHistory;
129         });
130 
131         Portfolio temp = null;
132 
133 
134 
135         for (Portfolio n : port) {
136             temp = n;
137         }
138 
139 
140         final ObservableList<History> equities =
    FXCollections.observableArrayList(temp.equityHistory);
141 
142 
143         eqTable.setItems(equities);
144         eqTable.getColumns().addAll(type,action,equity,
    date1,shares);
145 
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
    Transferred");
161 
162         type2.setCellValueFactory(
163                 new PropertyValueFactory<History,String>(
    "type")
164         );
165         action2.setCellValueFactory(
166                 new PropertyValueFactory<History,String>(
    "action")
167         );
168         ca.setCellValueFactory(
169                 new PropertyValueFactory<History,String>(
    "cashAccName")
170         );
171         date2.setCellValueFactory(
172                 new PropertyValueFactory<History,String>(
    "date2")
173         );
174         amount2.setCellValueFactory(
175                 new PropertyValueFactory<History,Double>(
    "transferAmount2")
176         );
177 
178         try {
179             FileInputStream fileIn = new FileInputStream(
    "myfile");
180             ObjectInputStream in = new ObjectInputStream(
    fileIn);
181             port = (ArrayList<Portfolio>) in.readObject()
    ;
182             in.close();
183             fileIn.close();
184         } catch (ClassNotFoundException ee) {
185             ee.printStackTrace();
186         } catch (FileNotFoundException ee) {
187             ee.printStackTrace();
188         } catch (IOException ee) {
189             ee.printStackTrace();
190         }
191         port.forEach(account -> {
192             account.cashAccountHistory = p.
    cashAccountHistory;
193         });
```

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

```
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<
    ActionEvent>() {
265                     @Override
266                     public void handle(ActionEvent e) {
267                         dialog.close();
268                     }
269                 });
270                 submit.setOnAction(new EventHandler<
    ActionEvent>() {
271                     @Override
272                     public void handle(ActionEvent event)
     {
273                         Equity e = null;
274                         History h = new History("","",e,
    "",0,0);
275                         h.exportEquityHistory(file.
    getText(),p);
276                         dialog.close();
277                     }
278                 });
279             }
280         });
281         //Action for exporting a cash account
282         exportCA.setOnAction(new EventHandler<ActionEvent
    >() {
283             @Override
284             public void handle(ActionEvent event) {
285                 final Stage dialog = new Stage();
286                 dialog.initOwner(primaryStage);
287                 GridPane grid2 = new GridPane();
288                 grid2.setAlignment(Pos.CENTER);
289                 grid2.setHgap(10);
290                 grid2.setVgap(10);
291                 grid2.setPadding(new Insets(5, 10, 5, 10)
    );
292 
293                 TextField file = new TextField();
294                 grid2.add(new Label("Filename: "),0,1);
295                 grid2.add(file,1,1);
296 
297                 Scene dialogScene = new Scene(grid2, 400,
     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
   market symbol associated
35      *                      with this equity.
36      * @param name : A string representing the name of the
    equity.
37      * @param shares : An int representing the number of
   shares of this equity owned.
38      * @param acquisitionPrice : A double representing the
    orginal price pad for
39      *                           this equity, per share.
40      * @param acquisitionDate : A String representing the
   date this equity was purchased.
```

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

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

```
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(",(?=([^\"]*\"[^\"]*
    \")*[^\"]*$)", -1);
130                 if(equity[0].equals("equity")){
131                     if (equity.length == 6) {
132                         ArrayList<String>
    marketAverageList = new ArrayList<String>();
133                         if(equity[2].contains("^")){
134                             //e.tickerSymbol = equity[2].
    substring(0, equity[2].lastIndexOf("^")).replaceAll
    ("^\"|\"$", "");
135                             e.tickerSymbol = equity[2].
    substring(0, equity[2].lastIndexOf("^")).replaceAll("[^=,
    \\da-zA-Z]", "");
136                         }else{
137                             e.tickerSymbol = equity[2].
    replaceAll("^\\s+|\\s+$", "").replaceAll("[^=,\\da-zA-Z]"
    , "");
138                         }
139                         e.shares = Integer.parseInt(
    equity[1].replaceAll("^\"|\"$", ""));
140                         e.acquisitionPrice = Double.
    parseDouble(equity[3].replace("\"", "").replaceAll("^\"|
    \"$", ""));
141                         e.name = equity[4];
142                         marketAverageList.add(equity[5].
    replaceAll("^\"|\"$", ""));
143                         e.marketAverage =
    marketAverageList;
144                         e.MA = e.marketAverage.toString()
    .replace("[","").replace("]","");
145                         Date date = null;
146                         SimpleDateFormat format = new
    SimpleDateFormat("dd/MM/yyyy");
147                         try {
148                             date = format.parse("00/00/
```

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

```
176                     Equity c = new Equity("","",0,0,"",
    null);
177                     c.MA = equity[1];
178                     c.shares = Integer.parseInt(equity[2]
    );
179                     c.acquisitionPrice = Double.
    parseDouble(equity[3]);
180                     p.marketAverages.add(c);
181                 }
182                 else {
183                     if (equity.length == 4) {
184                         ArrayList<String>
    marketAverageList = new ArrayList<String>();
185                         if (equity[0].contains("^")){
186                             e.tickerSymbol = equity[0].
    substring(0, equity[0].lastIndexOf("^")).replaceAll("[^=,
    \\da-zA-Z]", "");
187                         }else{
188                             e.tickerSymbol = equity[0].
    replaceAll("^\\s+|\\s+$", "").replaceAll("[^=,\\da-zA-Z]"
    , "");
189                         }
190                         e.name = equity[1].replaceAll("^
    \"|\"$", "");
191                         e.acquisitionPrice = Double.
    parseDouble(equity[2].replace("\"", "").replaceAll("^\"|
    \"$", ""));
192                         marketAverageList.add(equity[3].
    replaceAll("^\"|\"$", ""));
193                         e.marketAverage =
    marketAverageList;
194                         e.MA = e.marketAverage.toString()
    .replace("[","").replace("]","");
195                         Date date = null;
196                         SimpleDateFormat format = new
    SimpleDateFormat("dd/MM/yyyy");
197                         try {
198                             date = format.parse("00/00/
    0000");
199                         } catch (ParseException ee) {
200                             ee.printStackTrace();
201                         }
202                         addEquity(e.tickerSymbol, e.name,
     0, e.acquisitionPrice, "N/A", e.marketAverage, true, p);
203                     } else if (equity.length > 4) {
```

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

```
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("[","").
    replace("]",""));
276                     writer.append(',');
277                     writer.append(Integer.toString(ma.shares)
```

```
277 );
278                 writer.append(',');
279                 writer.append(Double.toString(ma.
    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 
292     /**
293      * Add an equity
294      */
295     public void addEquity(String tickerSymbol, String
    name, int numShares, double salePrice,
296                           String saleDate, ArrayList
    marketAverage, Boolean imported, Portfolio p){
297         Boolean exists = false;
298         for(Equity equity : p.equityList1){
299             if(!tickerSymbol.equals("")) {
300                 if (equity.tickerSymbol.equals(
    tickerSymbol)) {
301                     exists = true;
302                 }
303             } else{
304                 if(equity.marketAverage.get(0).equals(
    marketAverage.get(0))){
305                     exists = true;
306                 }
307             }
308         }
309         if(!exists) {
310 
311             Equity e = new Equity("", "", 0, 0, null,
    null);
312 
313             e.tickerSymbol = tickerSymbol;
314             e.name = name;
315             e.shares = numShares;
```

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

```
356                         if(q.marketAverage.contains(avg))
    {
357                             shares.add(q.shares);
358                         }
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
    .round((price / count) * 100.0) / 100.0;
368                     } else{
369                         marketAvg.acquisitionPrice = Math
    .round(((price / count) / .14602128057775) * 100.0) / 100
    .0;
370                     }
371                     p.marketAverages.add(marketAvg);
372                 }
373 
374             }
375             if(imported == false) {
376                 DateFormat dateFormat = new
    SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
377                 Date date = new Date();
378                 HistoryInvoker test = new HistoryInvoker(
    );
379                 test.setLog(new LogEquityHistory(), p,
    new History("Add", "EQUITY", e, dateFormat.format(date),
    0, numShares));
380                 test.logAction();
381             }
382 
383         }
384         else{
385             Equity e = null;
386             double amount = 0;
387             for(Equity equity : p.equityList1){
388                 if(!tickerSymbol.equals("")) {
389                     if (equity.tickerSymbol.equals(
    tickerSymbol)) {
390                         equity.shares += numShares;
391                         e = equity;
```

```
392                         amount = numShares * equity.
    getAcquisitionPrice();
393                     }
394                 } else{
395                     if(equity.marketAverage.get(0).equals
    (marketAverage.get(0))){
396                         equity.shares += numShares;
397                         e = equity;
398                         amount = numShares * equity.
    getAcquisitionPrice();
399                     }
400                 }
401             }
402             DateFormat dateFormat = new SimpleDateFormat(
    "yyyy/MM/dd HH:mm:ss");
403             Date date = new Date();
404             HistoryInvoker test = new HistoryInvoker();
405             test.setLog(new LogEquityHistory(), p, new
    History("Add Shares", "EQUITY", e,dateFormat.format(date)
    ,amount,numShares));
406             test.logAction();
407         }
408     }
409 
410     /**
411      * Remove Equity based on symbol given.
412      */
413     public void removeEquity(String symbol, String
    cashAcct, Portfolio p){
414         ArrayList<Equity> copyList = new ArrayList<>(p.
    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("","
    ,0,"",p);
422                     amount = -equity.getTotal();
423                     shares = -equity.shares;
424                     c.updateBalance(cashAcct, equity.
    getTotal(), true,p);
425                 }
426                 p.equityList1.remove(equity);
```

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

```java
package trunk.Model;


import trunk.Model.*;

import java.io.*;


/**
 * A class to represent an instance of an event or transaction
 * that should be logged to history.
 */

public class History implements Serializable{
    String type;
    String action;


    Equity equity;
    String equityName;

    CashAccount cashAccount;
    String cashAccName;
    Double transferAmount1;
    Double transferAmount2;

    Double shares;

    String date1;
    String date2;
    /**
     * Constructor for a History Object for a event related to an Equity.
     * @param action : A String representing the action. (e.g. add, remove, etc.)
     * @param type : A String representing which history type referenced. (e.g. "Equity")
     * @param equity : The Equity involved in the action.
     */
    public History(String action, String type, Equity equity, String date, double transferAmount, double shares) {
        this.type = type;
        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
   related to Cash Accounts.
48       * @param action : A String representing the action. (
   e.g. add, remove, etc.)
49       * @param type : A String representing which history
   type referenced. (e.g. "Cash Account")
50       * @param cashAccount : The Cash Account involved in
   the action.
51       */
52      public History(String action, String type, CashAccount
    cashAccount, String date, double transferAmount){
53          this.type = type;
54          this.action = action;
55          this.cashAccName = cashAccount.name;
56          this.date2 = date;
57          this.transferAmount2 = transferAmount;
58      }
59
60      /**
61       * A getter for the history type referenced.
62       * @return : A string representing this type.
63       */
64      public String getType(){
65          return this.type;
66      }
67
68      /**
69       * A getter for the action associated with the event
   that occurred.
70       * @return : A string representing this action.
71       */
72      public String getAction(){
73          return this.action;
74      }
75
76      /**
77       * A getter for the name of the Equity involved in an
   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
     in an event.
 86      * @return : A string representing the name.
 87      */
 88     public String getCashAccName(){
 89         return this.cashAccName;
 90     }
 91 
 92     public Double getTransferAmount1(){return this.
    transferAmount1;}
 93 
 94     public Double getTransferAmount2(){return this.
    transferAmount2;}
 95 
 96     public String getDate1(){return this.date1;}
 97 
 98     public String getDate2(){return this.date2;}
 99 
100     public Double getShares(){return this.shares;}
101 
102     /*
103      * Import cash account history
104      */
105     public void importCAHistory(String csv, Portfolio p)
    {
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 
113         try {
114 
115             br = new BufferedReader(new FileReader(csv));
116             while ((line = br.readLine()) != null) {
117                 History caH = new History("N/A", "N/A", c
    ,"",0);
```

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

```
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', '
    NameOfAccount'
185      */
186     public void exportCAHistory(String eFileName,
    Portfolio p){
187         try{
188             FileWriter writer = new FileWriter(eFileName)
    ;
189 
190             for (History h : p.cashAccountHistory){
191                 writer.append("transaction");
192                 writer.append(",");
193                 writer.append(h.action);
194                 writer.append(',');
195                 writer.append(h.type);
196                 writer.append(',');
197                 writer.append(h.cashAccName);
198                 writer.append("\n");
199             }
200 
```

```
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,
    Portfolio p){
214         try{
215             FileWriter writer = new FileWriter(eFileName)
    ;
216 
217             for (History h : p.equityHistory){
218                 writer.append("transaction");
219                 writer.append(",");
220                 writer.append(h.action);
221                 writer.append(',');
222                 writer.append(h.type);
223                 writer.append(',');
224                 writer.append(h.equityName);
225                 writer.append("\n");
226             }
227 
228             writer.flush();
229             writer.close();
230 
231         }catch (IOException e){
232             e.printStackTrace();
233         }
234     }
235 
236 }
237 
```

```java
package trunk.Model;

//import trunk.Model.*;



import trunk.Control.UpdateEQ;

import java.io.*;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import java.util.stream.Collectors;


/**
 * Class representation of the Financial Portfolio
 */
public class Portfolio implements Serializable{
    public String loginID; // Login ID of portfolio user
    public String password; // Password associated with loginID of user
    public ArrayList<Equity> equityList1 = new ArrayList<>(); //List of owned equities
    public ArrayList<Equity> equityList2 = new ArrayList<>(); //List of available equities
    public ArrayList<Equity> marketAverages = new ArrayList<>(); //List of market averages
    public ArrayList<Watchlist> watchlists = new ArrayList<>(); //List of watchlist items

    public ArrayList<CashAccount> cashAccList = new ArrayList<>();
    public ArrayList<History> cashAccountHistory = new ArrayList<>();
    public ArrayList<History> equityHistory = new ArrayList<>();


    public void initialEquities(){
        BufferedReader br = null;
        String line = "";
        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(",(?=([^\"]*\"[^\"]*\"
    )*[^\"]*$)", -1);
44                  if (equity.length == 4) {
45                      Equity e = new Equity("N/A","N/A",0,0,
    null,null);
46                      ArrayList<String> marketAverageList =
    new ArrayList<String>();
47                      if (equity[0].contains("^")){
48                          e.tickerSymbol = equity[0].
    substring(0, equity[0].lastIndexOf("^")).replaceAll("[^=,
    \\da-zA-Z]", "");
49                      }else{
50                          e.tickerSymbol = equity[0].
    replaceAll("[^=,\\da-zA-Z]", "");
51                      }
52                      e.name = equity[1].replaceAll("^\"|\"$
    ", "");
53                      e.acquisitionPrice = Double.
    parseDouble(equity[2].replace("\"", "").replaceAll("^\"|\"
    $", ""));
54                      marketAverageList.add(equity[3].
    replaceAll("^\"|\"$", ""));
55                      e.marketAverage = marketAverageList;
56
57
58                      e.MA = e.marketAverage.toString().
    replace("[","").replace("]","");
59                      Date date = null;
60                      SimpleDateFormat format = new
    SimpleDateFormat("dd/MM/yyyy");
61                      try {
62                          date = format.parse("00/00/0000");
63                      } catch (ParseException ee) {
64                          ee.printStackTrace();
65                      }
66                      e.addEquity(e.tickerSymbol, e.name, 0,
     e.acquisitionPrice, "N/A", e.marketAverage, true, this);
67                  } else if (equity.length > 4) {
68                      Equity e = new Equity("N/A","N/A",0,0,
```

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

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

```
140                         e.tickerSymbol = string[2].
    replaceAll("^\"|\"$", "");
141                         e.shares = Integer.parseInt(
    string[1].replaceAll("^\"|\"$", ""));
142                         e.acquisitionPrice = (Double.
    parseDouble(string[3].replace("\"", "").replaceAll("^\"|
    \"$", ""))) / e.shares;
143                         e.name = string[4];
144                         marketAverageList.add(string[5].
    replaceAll("^\"|\"$", ""));
145                         e.marketAverage =
    marketAverageList;
146                         e.MA = e.marketAverage.toString()
    .replace("[", "").replace("]", "");
147                         Date date = null;
148                         SimpleDateFormat format = new
    SimpleDateFormat("dd/MM/yyyy");
149                         try {
150                             date = format.parse("00/00/
    0000");
151                         } catch (ParseException ee) {
152                             ee.printStackTrace();
153                         }
154                         e.addEquity(e.tickerSymbol, e.
    name, e.shares, e.acquisitionPrice, "N/A", e.
    marketAverage, false, p);
155                     } else if (string.length > 6) {
156
157                         ArrayList<String>
    marketAverageList = new ArrayList<String>();
158                         e.tickerSymbol = string[2].
    replaceAll("^\"|\"$", "");
159                         e.shares = Integer.parseInt(
    string[1].replaceAll("^\"|\"$", ""));
160                         e.acquisitionPrice = (Double.
    parseDouble(string[3].replace("\"", "").replaceAll("^\"|
    \"$", ""))) / e.shares;
161                         if (string[5].contains("Inc")){
162                             e.name = string[4] += ", Inc
    .";
163                         }else{
164                             e.name = string[4];
165                         }
166                         int x = 5;
167                         while (x < string.length && !
```

```
167 string[x].equals("")) {
168                             if (!string[x].contains("Inc"
    )) {
169                                 marketAverageList.add(
    string[x].replaceAll("^\"|\"$", ""));
170                                 x++;
171                             }else{
172                                 x++;
173                             }
174                         }
175                         e.marketAverage =
    marketAverageList;
176                         e.MA = e.marketAverage.toString()
    .replace("[", "").replace("]", "");
177                         e.addEquity(e.tickerSymbol, e.
    name, e.shares, e.acquisitionPrice, "N/A", e.
    marketAverage, false, p);
178                     }
179                 } else if (string[0].equals("cashacc")) {
180                     CashAccount c = new CashAccount("N/A"
    , "N/A", 0.0, "", null);
181                     c.name = string[2];
182                     c.balance = Double.parseDouble(string
    [1].replace("\"", ""));
183                     //c.dateCreated = string[3];
184 
185                     if (!p.cashAccList.contains(c)) {
186                         c.addCashAccount(c.name, c.
    balance, c.dateCreated, p);
187                     }
188                 } else if (string[0].equals("transaction"
    )) {
189                     if (string[1].equals("EQUITY") ||
    string[1].equals("equity")) {
190                         Equity c = new Equity("", "", 0,
    0, "", null);
191                         History caH = new History("N/A",
    "N/A", c, "", 0, 0);
192 
193                         if (string.length == 5) {
194                             caH.type = string[1];
195                             caH.equityName = string[2];
196                             caH.transferAmount1 = Double.
    parseDouble(string[3]);
197                             caH.date1 = string[4];
```

```
198                         }  else{
199                             caH.type = string[1];
200                             caH.equityName = string[2] +=
     ", Inc.";
201                             caH.transferAmount1 = Double.
    parseDouble(string[4]);
202                             caH.date1 = string[5];
203                         }
204 
205                         p.equityHistory.add(caH);
206                     } else if (string[1].equals("
    CASHACCOUNT") || string[1].equals("cash account")) {
207                         CashAccount c = new CashAccount(
    "", "", 0, "", null);
208                         History caH = new History("N/A",
    "N/A", c, "", 0);
209 
210                         caH.type = string[1];
211                         caH.cashAccName = string[2];
212                         caH.transferAmount2 = Double.
    parseDouble(string[3]);
213                         caH.date2 = string[4];
214 
215                         p.cashAccountHistory.add(caH);
216                     }
217                 } else if (string[0].equals("
    watchlistItem")) {
218                     Watchlist w = new Watchlist("", "",
    "", 0, "", "", "");
219 
220                     w.tickerSymbol = string[1];
221                     w.lowtrigger = string[2];
222                     w.hightrigger = string[3];
223 
224                     p.watchlists.add(w);
225                 } else if (string[0].equals("
    marketAverage")) {
226 
227                     String tmp = string[1];
228                     List<Equity> curMarAvgs = this.
    marketAverages.stream().filter(u -> u.name.equals(tmp)).
    collect(Collectors.toList());
229 
230                     if (!(curMarAvgs.get(0).name.equals(
    string[1]))){
```

```
231                         Equity c = new Equity("", "", 0,
    0, "", null);
232                         c.MA = string[1];
233                         c.shares = Integer.parseInt(
    string[2]);
234                         c.acquisitionPrice = Double.
    parseDouble(string[3]);
235                         p.marketAverages.add(c);
236                     }else{
237                         curMarAvgs.get(0).shares +=
    Integer.parseInt(string[2]);
238                     }
239
240
241                     /**
242                     Equity c = new Equity("", "", 0, 0,
    "", null);
243                     c.MA = string[1];
244                     c.shares = Integer.parseInt(string[2
    ]);
245                     c.acquisitionPrice = Double.
    parseDouble(string[3]);
246                     p.marketAverages.add(c); */
247                 }
248
249             }
250             }catch(FileNotFoundException ee){
251                 ee.printStackTrace();
252             }catch(IOException ee){
253                 ee.printStackTrace();
254             }finally{
255                 if (br != null) {
256                     try {
257                         br.close();
258                     } catch (IOException ee) {
259                         ee.printStackTrace();
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             for (Equity equity : p.equityList1){
272                 writer.append("equity");
273                 writer.append(',');
274                 writer.append(equity.shares + "");
275                 writer.append(',');
276                 writer.append(equity.tickerSymbol);
277                 writer.append(',');
278                 writer.append(equity.getTotal() + "");
279                 writer.append(',');
280                 writer.append(equity.name);
281                 equity.marketAverage.forEach((SI) -> {
282                     try {
283                         writer.append(",");
284                         writer.append(SI.toString());
285                     }
286                     catch (IOException IO){
287                         IO.printStackTrace();
288                     } catch (NullPointerException e){
289                         e.printStackTrace();
290                     }
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                 writer.append(',');
327                 writer.append(h.date1);
328                 writer.append("\n");
329             }
330 
331             for (Watchlist w : p.watchlists){
332                 writer.append("watchlistItem");
333                 writer.append(',');
334                 writer.append(w.getTickerSymbol());
335                 writer.append(',');
336                 writer.append(w.lowtrigger);
337                 writer.append(',');
338                 writer.append(w.hightrigger);
339                 writer.append("\n");
340             }
341 
342             for(Equity ma : p.marketAverages){
343                 writer.append("marketAverage");
344                 writer.append(',');
345                 writer.append(ma.MA.replace("[","").
    replace("]",""));
346                 writer.append(',');
347                 writer.append(Integer.toString(ma.shares)
    );
348                 writer.append(',');
349                 writer.append(Double.toString(ma.
    acquisitionPrice));
350                 writer.append("\n");
351             }
352 
```

```
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(
    "myfile");
368             ObjectInputStream in = new ObjectInputStream(
    fileIn);
369             port = (ArrayList<Portfolio>) in.readObject()
    ;
370             in.close();
371             fileIn.close();
372         } catch (ClassNotFoundException ee) {
373             ee.printStackTrace();
374         } catch (FileNotFoundException ee) {
375             ee.printStackTrace();
376         } catch (IOException ee) {
377             ee.printStackTrace();
378         }
379         port.forEach(iter -> {
380             iter.marketAverages = p.marketAverages;
381         });
382 
383         try
384         {
385             FileOutputStream fileOut =
386                     new FileOutputStream("myfile");
387             ObjectOutputStream out = new
    ObjectOutputStream(fileOut);
388             out.writeObject(port);
389             out.close();
390             fileOut.close();
391         }catch(IOException i)
392         {
393             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,
   WatchlistVisitor  {
 9     public String tickerSymbol;
10     public String lowtrigger;
11     public String hightrigger;
12     public double pershareprice;
13
14     public String compare;
15     public String lowtriggered;
16     public String hightriggered;
17
18     /**
19      * Constructor for an Equity object.
20      * @param tickerSymbol: A String representing the
   market symbol associated with this equity.
21      * @param low : A string representing the low trigger.
22      * @param high : A string representing the hightrigger
   .
23      * @param price : A double representing the orginal
   price pad for this equity, per share.
24      * @param compare : A String representing that a
   trigger was tripped
25      * @param lowtriggered : A string representing that
   the low trigger has been tripped.
26      * @param hightriggered : A string representing that
   the high trigger has been tripped.
27      */
28     public Watchlist(String tickerSymbol, String low,
   String high, double price, String compare, String
   lowtriggered, String hightriggered){
29         this.tickerSymbol = tickerSymbol;
30         this.lowtrigger = low;
31         this.hightrigger = high;
32         this.pershareprice = price;
33         this.compare = compare;
34         this.lowtriggered = lowtriggered;
35         this.hightriggered = hightriggered;
36     }
```

```
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
    .lowtrigger)) {
44                      w.compare = "Low Trigger";
45                      w.lowtriggered = "Was Tripped";
46                  }
47              }
48              if(!w.hightrigger.equals("")) {
49                  if (w.pershareprice > Double.parseDouble(w
    .hightrigger)) {
50                      w.compare = "High Trigger";
51                      w.hightriggered = "Was Tripped";
52                  }
53              }
54          }
55      }
56
57      //A String representing the market symbol associated
    with this equity.
58      public String getTickerSymbol(){return this.
    tickerSymbol;}
59
60      //A string representing the low trigger.
61      public String getLowtrigger(){return this.lowtrigger;}
62
63      //A string representing the hightrigger.
64      public String getHightrigger(){return this.hightrigger
    ;}
65
66      //A double representing the orginal price pad for this
     equity, per share.
67      public double getPershareprice(){return this.
    pershareprice;}
68
69      //A String representing that a trigger was tripped
70      public String getCompare(){return this.compare;}
71
72      //A string representing that the low trigger has been
    tripped.
73      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
```

```java
package trunk.Model;

/**
 * An interface to handle various types of logging.
 */

public interface LogHistory {
    public void log(Portfolio portfolio, History history);
}

```

```
 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
   ArrayDeque<>();
32     public static ArrayDeque<UndoAction> redo = new
   ArrayDeque<>();
33
34
35     public UndoAction(String action, String objectType,
   CashAccount c1, CashAccount c2, double num,Equity eq,
   String file) {
36         this.actionType = action;
37         this.obj = c1;
38         this.objType = objectType;
39         this.transfer = c2;
40         this.transferAmount = num;
41         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,
    FPTS fpts, EquitiesView equitiesView, CAView caView) {
52          Iterator undoIterator = createUndoIterator();
53
54          for( Iterator iterate = undoIterator; iterate.
    hasNext();) {
55              UndoAction iter = (UndoAction) iterate.next();
56              if (iter.actionType.equals("Add") && iter.
    objType.equals("CashAccount") ) {
57                  UndoAddCashAcc(iter.obj, this.p);
58                  redo.addFirst(iter);
59                  undo.remove(iter);
60                  Scene scene = caView.CashAccounts(
    primaryStage,p, fpts);
61                  fpts.stage.setScene(scene);
62                  break;
63              }
64              else if (iter.actionType.equals("Remove") &&
    iter.objType.equals("CashAccount") ) {
65                  UndoRemoveCashAcc(iter.obj, this.p);
66                  redo.addFirst(iter);
67                  undo.remove(iter);
68                  Scene scene = caView.CashAccounts(
    primaryStage,p, fpts);
69                  fpts.stage.setScene(scene);
70                  break;
71              }
72
73              else if(iter.actionType.equals("Transfer") &&
    iter.objType.equals("CashAccount")){
74                  UndoTransfer(iter.obj, iter.transfer, iter
    .transferAmount, this.p);
75                  redo.addFirst(iter);
76                  undo.remove(iter);
77                  Scene scene = caView.CashAccounts(
    primaryStage,p, fpts);
```

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

```
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,
     EquitiesView equitiesView, CAView caView){
128         Iterator undoIterator = createRedoIterator();
129         for( Iterator iterate = undoIterator; iterate.
    hasNext();) {
130             UndoAction iter = (UndoAction) iterate.next()
    ;
131             if (iter.actionType.equals("Add") && iter.
    objType.equals("CashAccount") ) {
132                 UndoRemoveCashAcc(iter.obj, this.p);
133                 addToQueue(iter);
134                 redo.remove(iter);
135                 Scene scene = caView.CashAccounts(
    primaryStage,p, fpts);
136                 fpts.stage.setScene(scene);
137                 break;
138             }
139             else if (iter.actionType.equals("Remove") &&
    iter.objType.equals("CashAccount") ) {
140                 UndoAddCashAcc(iter.obj, this.p);
141                 addToQueue(iter);
142                 redo.remove(iter);
143                 Scene scene = caView.CashAccounts(
    primaryStage,p, fpts);
144                 fpts.stage.setScene(scene);
145                 break;
146             }
147             else if(iter.actionType.equals("Transfer") &&
     iter.objType.equals("CashAccount")){
148                 UndoTransfer(iter.transfer,iter.obj, iter
    .transferAmount, this.p);
149                 addToQueue(iter);
150                 redo.remove(iter);
```

```
151                 Scene scene = caView.CashAccounts(
    primaryStage,p, fpts);
152                 fpts.stage.setScene(scene);
153                 break;
154             }
155 
156             else if (iter.actionType.equals("Add") &&
    iter.objType.equals("Equity") ) {
157                 UndoRemoveEq(iter.item, this.p);
158                 addToQueue(iter);
159                 redo.remove(iter);
160                 Scene scene = equitiesView.ViewEquities(
    primaryStage,p, fpts);
161                 fpts.stage.setScene(scene);
162                 break;
163             }
164             else if (iter.actionType.equals("Remove") &&
    iter.objType.equals("Equity") ) {
165                 UndoAddEq(iter.item, this.p);
166                 addToQueue(iter);
167                 redo.remove(iter);
168                 Scene scene = equitiesView.ViewEquities(
    primaryStage,p, fpts);
169                 fpts.stage.setScene(scene);
170                 break;
171             }
172 
173             else if (iter.actionType.equals("Import") &&
    iter.objType.equals("CashAccount") ) {
174                 CashAccount c = new CashAccount("","",0,
    "",p);
175                 c.importCA(iter.file,p);
176 
177                 addToQueue(iter);
178                 redo.remove(iter);
179                 Scene scene = caView.CashAccounts(
    primaryStage,p, fpts);
180                 fpts.stage.setScene(scene);
181                 break;
182             }
183             else if (iter.actionType.equals("Import") &&
    iter.objType.equals("Equity") ) {
184                 Equity c = new Equity("","",0,0,null,null
    );
185 
```

```
186                 c.importEQ(iter.file,p);
187 
188                 addToQueue(iter);
189                 redo.remove(iter);
190                 Scene scene = equitiesView.ViewEquities(
    primaryStage,p, fpts);
191                 fpts.stage.setScene(scene);
192                 break;
193             }
194         }
195     }
196     public void UndoAddCashAcc(CashAccount obj, Portfolio
     p) {
197         obj.deleteCashAccount(obj.getName(), p); //
    deletes
198     }
199 
200     public void UndoRemoveCashAcc(CashAccount obj,
    Portfolio p) { // need Redo class
201         obj.addCashAccount(obj.name,obj.balance,obj.
    getDate(),p);
202     }
203 
204     public void UndoAddEq(Equity obj, Portfolio p) {
205         obj.removeEquity(obj.tickerSymbol,"None",p);
206     }
207 
208 
209 
210     public void UndoRemoveEq(Equity obj, Portfolio p) {
211         obj.addEquity(obj.tickerSymbol,obj.name,obj.
    shares,obj.acquisitionPrice,obj.acquisitionDate,obj.
    marketAverage,false,p);
212     }
213 
214     public void UndoTransfer(CashAccount c1, CashAccount
    c2, double num, Portfolio p) {
215         CashAccount c = new CashAccount("","",0,"",null);
216         c.transfer(c2,c1,num,p);
217     }
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(",(?=([^\"]*\"[^\"]*\"
    )*[^\"]*$)", -1);;
232 
233                 for(CashAccount c : p.cashAccList){
234                     if(c.name.equals(ca[2])){
235                         toRemove.add(c);
236                     }
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(",(?=([^\"]*\"[^\"]*
    \")*[^\"]*$)", -1);;
268 
269                 for(Equity c : p.equityList2){
270                     if(c.tickerSymbol.equals(equity[0].
    replaceAll("^\"|\"$", ""))){
271                         toRemove.add(c);
272                     }
273                 }
274                 for(Equity c : toRemove){
275                     p.equityList2.remove(c);
276                     //Resetting Market Averages
277                     for(String m : c.marketAverage){
278                         ArrayList<String> avgArray = new
    ArrayList<>();
279                         avgArray.add(m);
280                         Equity marketAvg = new Equity("",
     "", 0, 0, null, avgArray);
281                         double price = 0;
282                         int count = 0;
283                         for(Equity q : p.equityList2){
284                             if(q.marketAverage.contains(m
    )){
285                                 price += q.
    acquisitionPrice;
286                                 count += 1;
287                             }
288                         }
289                         p.equityList2.remove(marketAvg);
290                         if(!m.equals("DOW")) {
291                             marketAvg.acquisitionPrice =
    Math.round((price / count) * 100.0) / 100.0;
292                         } else{
293                             marketAvg.acquisitionPrice =
    Math.round(((price / count) / .14602128057775) * 100.0) /
     100.0;
294                         }
295                         p.equityList2.add(marketAvg);
296                     }
297                 }
298             }
299 
300         } catch (FileNotFoundException ee) {
301             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
    ArrayList<>(p.equityList2);
313             HashSet<String> existingAvgs = new HashSet<>(
    );
314             for(Equity e : equityListCopy){
315                 if(!e.tickerSymbol.equals("")){
316                     existingAvgs.addAll(e.marketAverage);
317                 }
318             }
319             for(Equity e : equityListCopy){
320                 if(e.tickerSymbol.equals("") && !
    existingAvgs.contains(e.marketAverage.get(0))){
321                     p.equityList2.remove(e);
322                 }
323             }
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,
   interacting
 6  * with equities, and interacting with a Portfolio.
 7  */
 8
 9 import trunk.Control.HistoryInvoker;
10
11 import java.io.*;
12 import java.text.DateFormat;
13 import java.text.DecimalFormat;
14 import java.text.SimpleDateFormat;
15 import java.util.ArrayList;
16 import java.util.Date;
17
18 public class CashAccount implements Serializable {
19     public String portfolioID;
20     public String name;
21     public double balance;
22     public String dateCreated;
23     Portfolio portfolio;
24
25     /**
26      * Constructor for a CashAccount Object.
27      * @param portfolioID : A string representing the
   portfolioID this account belongs to.
28      * @param name : A string representing the name of
   this account.
29      * @param dateCreated : A String representing the date
    the account was created.
30      * @param portfolio : The Portfolio object this
   account belongs to.
31      */
32     public CashAccount(String portfolioID, String name,
33                        double initAmount, String
   dateCreated, trunk.Model.Portfolio portfolio){
34
35         this.portfolioID = portfolioID;
36         this.name = name;
37         this.dateCreated = dateCreated;
38         DecimalFormat df = new DecimalFormat("#.00");
39         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
   account.
46       * @return : A string representing the portfolioID.
47       */
48      public String getPortfolioID(){
49          return this.portfolioID;
50      }
51
52      /**
53       * A getter for the name given to this account.
54       * @return : A string representing the account's name.
55       */
56      public String getName() {
57          return this.name;
58      }
59
60      /**
61       * A getter for the date associated with this account'
   s creation.
62       * @return : A string representing the creation date.
63       */
64      public String getDate() {
65          return this.dateCreated;
66      }
67
68      /**
69       * A getter for balance associated with this account
70       * @return : The double amount representing current
   cash amount in this account
71       */
72      public double getBalance(){return this.balance;}
73
74      /**
75       * Transfer funds from one account to another.
76       * @param amount : An int representing the amount to
   transfer.
77       * @param receiver : A CashAccount representing the
   account
78       *                   receiving the amount.
```

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

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

```
151                 if(!p.cashAccList.contains(c)) {
152                     addCashAccount(c.name, c.balance, c.
    dateCreated, p);
153                 } else{
154                     existingAccts.add(c);
155                 }
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     /**
175      * Export Cash Account
176      */
177     public void exportCA(String eFileName, Portfolio p) {
178         try{
179             FileWriter writer = new FileWriter(eFileName)
    ;
180 
181             for (CashAccount ca : p.cashAccList){
182                 writer.append("cashacc");
183                 writer.append(',');
184                 writer.append(ca.balance + "");
185                 writer.append(',');
186                 writer.append(ca.getName());
187                 writer.append(',');
188                 writer.append(ca.dateCreated);
189                 writer.append('\n');
190             }
191 
192             writer.flush();
193             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
    HistoryInvoker();
233                     test.setLog(new LogCAHistory(), p,
    new History("Update", "CASHACCOUNT", account,dateFormat.
    format(date),amount));
234                     test.logAction();
235                 }
236                 else {
237                     account.balance -= amount;
238                     DateFormat dateFormat = new
    SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
239                     Date date = new Date();
240
241                     HistoryInvoker test = new
    HistoryInvoker();
242                     test.setLog(new LogCAHistory(), p,
    new History("Update", "CASHACCOUNT", account,dateFormat.
    format(date),-amount));
243                     test.logAction();
244                 }
245             }
246         });
247     }
248
249     /*
250     * Transfer money between two cash accounts
251     */
252     public void transfer(CashAccount c1, CashAccount c2,
    Double num, Portfolio p){
253         DateFormat dateFormat = new SimpleDateFormat("
    yyyy/MM/dd HH:mm:ss");
254         Date date = new Date();
255         for(CashAccount c : p.cashAccList){
256             if(c.name.equals(c2.name)){
257                 c.balance -= num;
258
259
260                 HistoryInvoker test = new HistoryInvoker(
    );
261                 test.setLog(new LogCAHistory(), p, new
    History("Update", "CASHACCOUNT", c,dateFormat.format(date
    ),-num));
262                 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                     test.setLog(new LogCAHistory(), p, new
    History("Update", "CASHACCOUNT", c,dateFormat.format(date
    ),num));
272                     test.logAction();
273                     break;
274                 }
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)
    ){
284                     return true;
285                 }
286             }
287             return false;
288         }
289 }
290 
```

```java
package trunk.Model;

/**
 * A LogHistory class that represents Cash Account History.
 */


public class LogCAHistory implements LogHistory {
    @Override
    public void log(Portfolio portfolio, History history) {
        portfolio.cashAccountHistory.add(history);
    }
}

```

```
 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,
   FPTS fpts, EquitiesView equitiesView, CAView caView);
13 }
14
```

```java
package trunk.Model;

/**
 * A LogHistory class that represents Equity History.
 */
public class LogEquityHistory implements LogHistory {
    @Override
    public void log(Portfolio portfolio, History history)
  {
        portfolio.equityHistory.add(history);
    }
}

```

```java
package trunk.Model;

/**
 * Created by sadaf345 on 4/14/2016.
 */
public interface WatchlistVisitor {
    public void visit(Portfolio p);
}

```

```java
package trunk.Control;

import trunk.Model.*;

/**
 * With this simulation, the equity prices never change. This class implements
 * the FutureSimulation interface, and acts as a concrete strategy within the
 * Strategy design pattern.
 */
public class NoGrowth  implements FutureSimulation {

    /**
     * The method to run the simulation.
     * @param portfolio : A Portfolio object that will run the simulation.
     * @param percentage : A double that represents a per annum percentage.
     * @param interval : An int that represents a period of time.
     * @param steps : An int that represents the number of intervals to run.
     * @return : A double representing the equity price after the simulation.
     */
    public double[] simulate(Portfolio portfolio, double percentage, int interval, int steps) {
        steps += 1;
        double[] finalValues = new double[steps];
        double finalValue = 0;
        double cashValue = 0;

        for (int i = 0; i < portfolio.equityList1.size(); i++) {
            Equity e = portfolio.equityList1.get(i);
            finalValue += e.shares * e.acquisitionPrice;
        }

        for (int i = 0; i < portfolio.cashAccList.size(); i++) {
            CashAccount ca = portfolio.cashAccList.get(i);
            cashValue += ca.balance;
        }

```

```
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 initValue = e.shares * e.acquisitionPrice;
48          vals[0] = initValue;
49
50          for (int i = 1; i < steps; i++) {
51              vals[i] = initValue;
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,
   ArrayList<Equity> equities) throws Exception {
12         YahooGET get = new YahooGET();
13         String yahooXML = get.yahooCall(get.formatSymbols(
   equities));
14
15         YahooParse parse = new YahooParse();
16         ArrayList<Double> askPrice = parse.parseXML(
   yahooXML, portfolio);
17
18
19         for (int i = 0; i < equities.size(); i++) {
20             equities.get(i).acquisitionPrice = askPrice.
   get(i);
21         }
22
23     }
24 }
```

```java
package trunk.Control;

import trunk.Model.Equity;
import trunk.Model.Portfolio;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

import java.net.HttpURLConnection;
import java.net.URL;
import java.util.ArrayList;

public class YahooGET {

    String urlCall = "";

    public String yahooCall(String symbolString) throws IOException {

        String url =
                String.format("http://query.yahooapis.com/v1/public/yql?q=select%%20*%%20from%%20yahoo.finance.quotes%%20where%%20symbol%%20in%%20(%s)&env=store://datatables.org/alltableswithkeys", symbolString);

        URL YahooURL = new URL(url);
        HttpURLConnection con = (HttpURLConnection) YahooURL.openConnection();

        con.setRequestMethod("GET");

        BufferedReader in = new BufferedReader(
                new InputStreamReader(con.getInputStream()));
        String inputLine;
        StringBuilder response = new StringBuilder();

        while ((inputLine = in.readLine()) != null) {
            response.append(inputLine);
        }

        in.close();

        return response.toString();
```

```
40      }
41
42      public String formatSymbols(ArrayList<Equity> equities
    ){
43          int i = 0;
44
45          while( i < equities.size() ){
46              if (i == equities.size() - 1) {
47                  urlCall += "%22";
48                  urlCall += equities.get(i).tickerSymbol;
49                  urlCall += "%22";
50              } else {
51                  urlCall += "%22";
52                  urlCall += equities.get(i).tickerSymbol;
53                  urlCall += "%22";
54                  urlCall += "%2C";
55                  urlCall += "%20";
56              }
57              i++;
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
   annum percentage that each
 7  * equity will decrease. This class implements the
   FutureSimulation interface, and
 8  * acts as a concrete strategy within the Strategy design
   pattern.
 9  */
10
11 public class BearMarket  implements FutureSimulation {
12
13     /**
14      * The method to run the simulation.
15      * @param portfolio : A Portfolio object that will run
    the simulation.
16      * @param percentage : A double that represents a per
   annum percentage.
17      * @param interval : An int that represents a period
   of time.
18      * @param steps : An int that represents the number of
    intervals to run.
19      * @return : A double representing the equity price
   after the simulation.
20      */
21     public double[] simulate(Portfolio portfolio, double
   percentage, int interval, int steps) {
22
23         steps += 1;
24         double[] finalValues = new double[steps];
25         double initValue = 0;
26         double finalValue;
27         double cashValue = 0;
28         double totChange = 0;
29
30         for (int i = 0; i < portfolio.equityList1.size();
   i++) {
31             Equity e = portfolio.equityList1.get(i);
32             initValue += e.shares * e.acquisitionPrice;
33         }
34
35
```

```
36          for (int i = 0; i < portfolio.cashAccList.size();
    i++) {
37              CashAccount ca = portfolio.cashAccList.get(i);
38              cashValue += ca.balance;
39          }
40
41          initValue += cashValue;
42          finalValues[0] = initValue;
43          double sub = initValue * (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 initValue = e.shares * e.acquisitionPrice;
59          vals[0] = initValue;
60
61          for (int i = 1; i < steps; i++) {
62              initValue -= initValue * (percentage / 100);
63              if (initValue >= 0)
64                  vals[i] = initValue;
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
   annum percentage that each
 7  * equity will increase. This class implements the
   FutureSimulation interface, and
 8  * acts as a concrete strategy within the Strategy design
   pattern.
 9  */
10 public class BullMarket implements FutureSimulation {
11
12     /**
13      * The method to run the simulation.
14      * @param portfolio : A Portfolio object that will run
    the simulation.
15      * @param percentage : A double that represents a per
   annum percentage.
16      * @param interval : An int that represents a period
   of time.
17      * @param steps : An int that represents the number of
    intervals to run.
18      * @return : A double representing the equity price
   after the simulation.
19      */
20     public double[] simulate(Portfolio portfolio, double
   percentage, int interval, int steps) {
21         steps += 1;
22         double[] finalValues = new double[steps];
23         double initValue = 0;
24         double finalValue = 0;
25         double cashValue = 0;
26
27         for (int i = 0; i < portfolio.equityList1.size();
   i++) {
28             Equity e = portfolio.equityList1.get(i);
29             initValue += e.shares * e.acquisitionPrice;
30         }
31
32         for (int i = 0; i < portfolio.cashAccList.size();
   i++) {
33             CashAccount ca = portfolio.cashAccList.get(i);
34             cashValue += ca.balance;
```

```
35          }
36
37          initValue += cashValue;
38          finalValues[0] = initValue;
39          double add = initValue * (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,
    int interval, int steps) {
49          double[] vals = new double[steps];
50          double initValue = e.shares * e.acquisitionPrice;
51          vals[0] = initValue;
52
53          for (int i = 1; i < steps; i++) {
54              initValue += initValue * (percentage / 100);
55              vals[i] = initValue;
56          }
57
58          return vals;
59      }
60  }
61
```

```java
package trunk.Control;

import org.w3c.dom.*;
import org.xml.sax.InputSource;
import trunk.Model.Equity;
import trunk.Model.Portfolio;

import javax.sound.sampled.Port;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import java.io.StringReader;
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;

public class YahooParse {
    public ArrayList<Double> parseXML(String xmlRecords, Portfolio portfolio) throws Exception{

        ArrayList<Double> askingPrices = new ArrayList<Double>();
        DocumentBuilder db = DocumentBuilderFactory.newInstance().newDocumentBuilder();
        InputSource is = new InputSource();
        is.setCharacterStream(new StringReader(xmlRecords));

        Document doc = db.parse(is);
        NodeList nodes = doc.getElementsByTagName("quote");

        for (int i = 0; i < nodes.getLength(); i++) {
            Element element = (Element) nodes.item(i);

            NodeList askPrice = element.getElementsByTagName("Ask");
            Element line1 = (Element) askPrice.item(0);

            NodeList tickerSym = element.getElementsByTagName("Symbol");
            Element line2 = (Element) tickerSym.item(0);

            String sym = getCharacterDataFromElement(line2);

```

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

```java
/**
 * A class that acts as the Invoker in the command design pattern.
 * Gets a request from a Portfolio or a Cash Account and has LogHistory
 * run the appropriate command.
 */
package trunk.Control;

import trunk.Model.*;

public class HistoryInvoker {
    private LogHistory event;
    private trunk.Model.Portfolio portfolio;
    private History history;

    /**
     * A constructor for the HistoryInvoker.
     * Instantiates which type of log should be used.
     * @param event : A LogHistory object representing the type of log to be used.
     * @param portfolio : The Portfolio involved with the events occurring.
     * @param history : A History object representing the event that occurred.
     */
    public void setLog(LogHistory event, trunk.Model.Portfolio portfolio, History history){
        this.event = event;
        this.portfolio = portfolio;
        this.history = history;
    }

    /**
     * Sends the portfolio involved with the event that occurred and the history object
     * representing that event to the appropriate log type.
     */
    public void logAction(){
        event.log(portfolio, history);
    }
}

```

```
 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
```