```
package controller;

import gui.FPTS;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;

import java.util.ArrayList;
import java.util.Observable;

/**
 * Defines view and controller to obtain any number of numerical inputs, validates
 * such inputs, and notifies observers.
 *
 * @author Eric Epstein
 */
public class AmountInput extends Observable {

    /*
    * ArrayList of amounts, often just one element
    */
    ArrayList<Double> amounts;

    /*
    * context data
    */
    private FPTS theFPTS;

    /**
     * Stores context data in construction
     *
     * @param theFPTS - FPTS
     * @param amounts - ArrayList<Double> - often empty
     */
    public AmountInput(FPTS theFPTS, ArrayList<Double> amounts) {

        this.theFPTS = theFPTS;
        this.amounts = amounts;

        theFPTS.getStage().setScene(getAmountInputScene());
    }
```

```
/**
 * Constructs Scene to process input
 *
 * @return Scene
 */
public Scene getAmountInputScene() {
    VBox split = new VBox();

    HBox aField = new HBox();
    TextField inputAmount = new TextField();
    Label aLabel = new Label("Input amount: ");
    aField.getChildren().addAll(aLabel, inputAmount);

    ArrayList<TextField> inputAmounts = new ArrayList<TextField>();
    inputAmounts.add(inputAmount);

    Button submitBtn = new Button();
    submitBtn.setText("Submit");

    /*
    * Defines action event when the user presses "Submit"
    */
    submitBtn.setOnAction(new EventHandler<ActionEvent>() {
        @Override
        public void handle(ActionEvent e) {
            boolean isValid = true;

            /**
             * For each input amount from the user, all must be true
             */
            for (TextField inputAmount : inputAmounts) {
                isValid = isValid && isValid(inputAmount);
            }

            /*
            * If each input is a valid numerical value, add to ArrayList of
            * amounts.
            */
            if (isValid) {
                for (TextField inputAmount : inputAmounts) {
                    double anAmount = Double.parseDouble(inputAmount.getText());
                    amounts.add(anAmount);
                }
                setChanged();
                notifyObservers();
            /*
            * Else, do not proceed and inform user of invalid input.
```

```
            */
            } else {
                inputAmount.setText("INVALID");
            }

        }
    });

    VBox inputArea = new VBox();
    inputArea.getChildren().addAll(aField, submitBtn);
    split.getChildren().addAll(theFPTS.getNav(), inputArea);
    return new Scene(split, theFPTS.getWidth(), theFPTS.getHeight());

}

/**
 * Validates user input
 *
 * @param inputAmount
 * @return boolean
 */
public boolean isValid(TextField inputAmount) {
    if (inputAmount.getText() == null || inputAmount.getText().equals("")) {
        return false;
    }

    String inputAmountString = inputAmount.getText();

    try {
        Double.parseDouble(inputAmountString);
    } catch (Exception e) {
        return false;
    }

    Double inputAmountDouble = Double.parseDouble(inputAmountString);

    if (inputAmountDouble < 0) {
        return false;
    }

    return true;

}

}

package model;
```

```
import gui.FPTS;

import java.util.ArrayList;

/**
 * Created by Brockway on 3/12/16.
 */
public class BearSimulator implements Simulator {
    public static String name = "Bear Market Simulator";

    private ArrayList<Holding> holdings;//TODO: find out how to get the
holdings******
    private String interval;
    private boolean hasSteps;
    private int numSteps;
    private double pricePerYear;
    private double currentPercentDecrease;
    private int stepNumber;

    /**
     * Bear Market Constructor. Input values are converted
     * to their appropriate types in the SimulationController and
     * then this constructor is called.
     *
     * @param numSteps
     * @param interval
     * @param hasSteps
     * @param pricePerYearPercentage
     */
    public BearSimulator(int numSteps, String interval, boolean hasSteps, double
pricePerYearPercentage) {
        this.interval = interval;
        this.hasSteps = hasSteps;
        this.numSteps = numSteps;
        this.pricePerYear = pricePerYearPercentage;
        this.holdings = FPTS.getSelf().getPortfolio().getHoldings();
        this.stepNumber = 0;
    }


    //TODO: CHECK IF IT HAS STEPS.

    /**
     * DECREASE
     *
     * @return
```

```
     */
    @Override
    public double simulate(int numberOfSteps) {
        double valueCount = 0;
        if (interval.equals("Day")) {
            currentPercentDecrease = pricePerYear / 365;
        } else if (interval.equals("Month")) {
            currentPercentDecrease = pricePerYear / 12;
        } else {
            currentPercentDecrease = pricePerYear;
        }
        for (int i = 0; i < numberOfSteps; i++) {
            for (Holding h : holdings) {
                valueCount -= currentPercentDecrease * h.getValue();
            }
        }
        stepNumber += numberOfSteps;
        return valueCount;
    }

    @Override
    public int getCurrentStep() {
        return stepNumber;
    }

    @Override
    public int getTotalSteps() {
        return numSteps;
    }


}

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package controller;

import model.*;

import java.util.ArrayList;
import java.util.Date;

/**
 * @author ericepstein
```

```
 *       <p>
 *       Extends HoldingAlgorithm
 */
public class BuyHoldingAlgorithm extends HoldingAlgorithm {

    private Portfolio p;
    private ArrayList<Searchable> toBeSearched;

    @Override
    public void establishContext() {
        p = theFPTS.getPortfolio();
        toBeSearched = p.getEquityComponentSearchables();
    }

    public ArrayList<Searchable> getToBeSearched() {
        return toBeSearched;
    }

    /*
    *
    *    precondition - cashAccountOfInterest is already set up

    */

    public void processInsideFPTS() {

        String aTickerSymbol = equityOfInterest.getTickerSymbol();
        Holding e;
        if (p.getHolding(aTickerSymbol) != null) {
            e = p.getHolding(aTickerSymbol);
        } else {
            e = new Holding(equityOfInterest.getTickerSymbol(),
equityOfInterest.getHoldingName(), equityOfInterest.getValuePerShare(),
numOfShares, new Date(), equityOfInterest.getSectors(),
equityOfInterest.getIndices());
        }

        double accountVal = cashAccountOfInterest.getValue();

        if (accountVal >= (numOfShares * pricePerShare)) {
            CashAccount aC =
theFPTS.getPortfolio().getCashAccount(cashAccountOfInterest);
            Transaction t = new Withdrawal(aC, numOfShares * pricePerShare);
            p.add(t);
            e.addShares(numOfShares);

            theStage.setScene(theFPTS.getConfirmationScene());
```

```
        } else {
            mainInput.setText("INVALID");
        }

    }

    public void processOutsideFPTS() {

        String keyword = mainInput.getText();

        for (Searchable s : toBeSearched) {
            if (keyword.equals(s.getDisplayName())) {

                //If the holding exists in the collection, increase # of shares
                if (p.getHolding(keyword) != null) {
                    Holding e = p.getHolding(keyword);
                    e.addShares(numOfShares);
                    //If equity does not exist in the collection, create a new Holding &
add to collection
                } else {
                    Holding e = new Holding(equityOfInterest.getTickerSymbol(),
equityOfInterest.getHoldingName(), equityOfInterest.getValuePerShare(),
numOfShares, new Date(), equityOfInterest.getSectors(),
equityOfInterest.getIndices());
                    p.add(e);
                }

                theStage.setScene(theFPTS.getConfirmationScene());

            } else {
                mainInput.setText("INVALID");
            }
        }

    }

}

package model;

import gui.FPTS;

import java.util.ArrayList;

/**
 * Created by Brockway on 3/12/16.
 */
```

```
public class BullSimulator implements Simulator {
    public static String name = "Bull Market Simulator";

    private ArrayList<Holding> holdings;//TODO: prices increase
    private String interval;
    private boolean hasSteps;
    private int numSteps;
    private double pricePerYear;
    private double currentPercentIncrease;
    private int stepNumber;

    /**
     * @param numSteps
     * @param interval
     * @param hasSteps
     * @param pricePerYearPercentage
     */
    public BullSimulator(int numSteps, String interval, boolean hasSteps, double
pricePerYearPercentage) {
        this.interval = interval;
        this.hasSteps = hasSteps;
        this.numSteps = numSteps;
        this.pricePerYear = pricePerYearPercentage;
        this.holdings = FPTS.getSelf().getPortfolio().getHoldings();
        this.stepNumber = 0;
    }


    //TODO: CHECK IF IT HAS STEPS.

    /**
     * @return
     */
    @Override
    public double simulate(int numberOfSteps) {
        double valueCount = 0;
        if (interval.equals("Day")) {
            currentPercentIncrease = pricePerYear / 365;
        } else if (interval.equals("Month")) {
            currentPercentIncrease = pricePerYear / 12;
        } else {
            currentPercentIncrease = pricePerYear;
        }
        for (int i = 0; i < numberOfSteps; i++) {
            for (Holding h : holdings) {
                valueCount += currentPercentIncrease * h.getValue();
            }
```

```
        }
        stepNumber += numberOfSteps;
        return valueCount;
    }

    @Override
    public int getCurrentStep() {
        return stepNumber;
    }

    @Override
    public int getTotalSteps() {
        return numSteps;
    }
}
```

```
package controller;

import gui.FPTS;
import model.CashAccount;

import java.util.Observable;
import java.util.Observer;

/**
 * Defines the general steps of managing CashAccount. Implements the
 * first step of obtaining a CashAccount object.
 *
 * @author Eric Epstein
 */
abstract public class CashAccountAlgorithm implements Observer {

    /*
    * CashAccount of interest, to be modified in further steps
    */
    protected CashAccount c;

    /*
    * context data
    */
    protected FPTS theFPTS;

    /**
     * Stalls algorithm until a CashAccount is obtained from CashAccountFinder
     *
     * @param theFPTS - FPTS
     */
```

```
    public void process(FPTS theFPTS) {
        c = new CashAccount("", 0, null);
        this.theFPTS = theFPTS;
        CashAccountFinder caFinder = new CashAccountFinder(theFPTS, c);
        caFinder.addObserver(this);
    }

    /**
     * Upon update from CashAccountFinder (in this context), delegates
     * next step to child algorithms
     *
     * @param o
     * @param args
     */
    public void update(Observable o, Object args) {
        action();
    }

    /*
    * Abstract method that is proceeded once the current algorithm is notified
    * of an update that assigns a CashAccount of interest.
    */
    abstract void action();

}

package model;

import java.util.ArrayList;
import java.util.Date;

/**
 * Holds a user's cash account details, including the account name,
 * creation date and total value that may change depending on various
operations.
 *
 * @author Eric Epstein and Kaitlyn Brockway
 */
public class CashAccount implements Searchable {
    private String accountName;
    private double currentValue;
    private Date dateAdded;
    public static ArrayList<CashAccount> cashList = getCashList();

    /**
     * getCashList() returns cash accounts
     */
```

```
    public static ArrayList<CashAccount> getCashList() {
        return cashList;
    }

    /**
     * The system shall allow the user to specify a new cash account.
     * A user defines a cash account by specifying an account name,
     * initial amount, and the date it was added.
     *
     * @param AccountName   - String
     * @param initialAmount - double
     * @param dateAdded     - Date
     */
    public CashAccount(String AccountName, double initialAmount, Date
dateAdded) {
        this.accountName = AccountName;
        currentValue = initialAmount;
        this.dateAdded = dateAdded;
    }

    /**
     * Returns the account name.
     * <p>
     * returns: String accountName
     */
    public String getAccountName() {
        return accountName;
    }

    /**
     * returns account name to override Searchable interface
     *
     * @return String
     */
    @Override
    public String getDisplayName() {
        return accountName;
    }

    /**
     * Returns value for display
     *
     * @return String
     */
    public String toString() {
        return "The cash account \"" + accountName + "\" contains $" +
currentValue;
```

```
}

/**
 * Returns empty string indicating lack of symbol
 *
 * @return String
 */
public String getSymbol() {
    return "";
}

/**
 * returns current value
 *
 * @return double
 */
public double getValue() {
    return currentValue;
}

/**
 * overrides default equals() method
 * <p>
 * Precondition: Object obj can be casted to CashAccount
 *
 * @param obj
 * @return boolean
 */
@Override
public boolean equals(java.lang.Object obj) {
    CashAccount c = (CashAccount) obj;
    return (accountName.equals(c.getAccountName()));
}

/**
 * Returns date the cash account was added
 *
 * @return Date
 */
public Date getDateAdded() {
    return dateAdded;
}


/**
 * subtracts current value by a specified amount
 * <p>
```

```
     * Precondition: the amount cannot exceed the total value of the cash account
     * and the amount is non-negative
     *
     * @param amount - double
     */
    public void withdraw(double amount) {
        currentValue -= amount;
    }


    /**
     * adds amount to current value
     * <p>
     * Precondition: the amount is a non-negative number
     *
     * @param amount - double
     */
    public void deposit(double amount) {
        currentValue += amount;
    }

    /*
    * mimics a copy function to maintain reference to the same cash account
    *
    * @param c - CashAccount
    */
    public void overwrite(CashAccount c) {
        this.accountName = c.getAccountName();
        this.currentValue = c.getValue();
        this.dateAdded = c.getDateAdded();
    }


}

package controller;

import gui.FPTS;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.DatePicker;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
```

```
import model.CashAccount;

import java.time.LocalDate;
import java.time.ZoneId;
import java.util.Date;

/**
 * Implements view and control to create a new CashAccount for a current
portfolio.
 *
 * @author Eric Epstein
 */
public class CashAccountCreator {

    /*
    * context data
    */
    private FPTS theFPTS;

    /**
     * Establishes context data and calls for scene construction
     *
     * @param theFPTS - FPTS
     */
    public CashAccountCreator(FPTS theFPTS) {
        this.theFPTS = theFPTS;
        theFPTS.getStage().setScene(getCashAccountCreatorScene());
    }

    /**
     * Constructs scene with specified fields and input controls.
     *
     * @return Scene
     */
    public Scene getCashAccountCreatorScene() {
        VBox split = new VBox();

        /*
        * Defines the search fields
        */
        HBox aField = new HBox();
        TextField nameInputField = new TextField();
        Label mainInput = new Label("Account name: ");
        aField.getChildren().addAll(mainInput, nameInputField);

        TextField amountInputField = new TextField();
        Label aLabel = new Label("Amount: ");
```

```
        aField.getChildren().addAll(aLabel, amountInputField);

        DatePicker dateField = new DatePicker(LocalDate.now());
        aLabel = new Label("Date: ");
        aField.getChildren().addAll(aLabel, dateField);

        Button submitBtn = new Button();
        submitBtn.setText("Submit");

        /*
        * Processes input
        */
        submitBtn.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent e) {

                boolean isValid = isValidAccountName(nameInputField) &&
isValidDouble(amountInputField);

                /*
                * Converts dateField to Date object
                */
                Date theDate =
Date.from(dateField.getValue().atStartOfDay(ZoneId.systemDefault()).toInstant())
;

                if (isValid) {
                    CashAccount c = new CashAccount(nameInputField.getText(),
Double.parseDouble(amountInputField.getText()), theDate);
                    theFPTS.getPortfolio().add(c);

                    /*
                    * Refers to confirmation scene
                    */
                    theFPTS.getStage().setScene(theFPTS.getConfirmationScene());
                } else {
                    nameInputField.setText("INVALID");
                }
            }
        });

        VBox inputArea = new VBox();
        inputArea.getChildren().addAll(aField, submitBtn);
        split.getChildren().addAll(theFPTS.getNav(), inputArea);
        return new Scene(split, theFPTS.getWidth(), theFPTS.getHeight());

    }
```

```
    /**
     * Helper logic function to validate account name input
     *
     * @param inputAccountName - TextField
     * @return boolean
     */
    private boolean isValidAccountName(TextField inputAccountName) {
        if (inputAccountName.getText() == null ||
inputAccountName.getText().equals("")) {
            return false;
        }

        String inputAccountString = inputAccountName.getText();

        for (CashAccount c : theFPTS.getPortfolio().getCashAccounts()) {
            if (c.getAccountName().equals(inputAccountString)) {
                return false;
            }
        }

        return true;
    }

    /**
     * Helper logic function to validate numerical input
     *
     * @param inputAmount - TextField
     * @return boolean
     */
    private boolean isValidDouble(TextField inputAmount) {

        /*
        * Determine whether the input amount is not empty
        */
        if (inputAmount.getText() == null || inputAmount.getText().equals("")) {
            return false;
        }

        String inputAmountString = inputAmount.getText();

        /*
        * Determine whether the input amount can be parsed to a double
        */
        try {
            Double.parseDouble(inputAmountString);
        } catch (Exception e) {
```

```
            return false;
        }

        Double inputAmountDouble = Double.parseDouble(inputAmountString);

        /*
        * Return whether the input amount is greater than 0
        */
        return (inputAmountDouble >= 0);

    }

}

package controller;

import gui.FPTS;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.ComboBox;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
import model.*;

import java.util.ArrayList;
import java.util.Observable;
import java.util.Observer;

/**
 * Defines the view and control for selecting CashAccount. Observes
 * ashAccountSearcher, updates display, then notifies algorithms when a
 * CashAccount has been selected.
 *
 * @author Eric Epstein
 */
public class CashAccountFinder extends Observable implements Observer {

    /*
    * display of matches
    */
```

```
    VBox matchDisplay;

    /*
    * context data
    */
    FPTS theFPTS;

    /*
    * text field that user types in
    */
    TextField mainInput;

    /*
    * searcher algorithm
    */
    Searcher s;

    /*
    * CashAccount being found
    */
    CashAccount c;

    /**
     * Establishes context data, adds itself as observer of CashAccountSearcher,
     * then calls for scene construction.
     *
     * @param theFPTS
     * @param c
     */
    public CashAccountFinder(FPTS theFPTS, CashAccount c) {
        mainInput = new TextField();
        matchDisplay = new VBox();
        s = new CashAccountSearcher();
        this.s.addObserver(this);

        this.c = c;
        this.theFPTS = theFPTS;
        Portfolio p = theFPTS.getPortfolio();
        Stage theStage = theFPTS.getStage();
        matchDisplay.getChildren().clear();
        mainInput.setText("");

        Scene searchScene = getSearchScene(p.getCashAccountSearchables(),
getCashAccountQueries());
        theStage.setScene(searchScene);

    }
```

```
    /**
     * Constructs view and control for searching and selecting CashAccount.
     *
     * @param toBeSearched
     * @param queries
     * @return
     */
    public Scene getSearchScene(ArrayList<Searchable> toBeSearched, VBox
queries) {

        VBox splitPage = new VBox();
        VBox searchPane = new VBox();

        Button actionBtn = new Button();
        actionBtn.setVisible(false);
        actionBtn.setText("Proceed");
        /*
        * Handles event of selecting a match
        */
        actionBtn.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent e) {
                if (mainInput.getText() != null && s.getMatch(mainInput.getText()) !=
null) {
                    CashAccount aCashAccount = (CashAccount)
s.getMatch(mainInput.getText());
                    c.overwrite(aCashAccount);
                    setChanged();
                    notifyObservers();
                }
            }
        });

        /*
        * Handle event of searching for a match
        */
        Button searchBtn = new Button();
        searchBtn.setText("Search");
        searchBtn.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent e) {
                s.search(queries.getChildren(), toBeSearched);
                actionBtn.setVisible(true);
            }
        });
```

```
    HBox forAction = new HBox();
    forAction.getChildren().addAll(queries, actionBtn);
    searchPane.getChildren().addAll(forAction, searchBtn, matchDisplay);
    splitPage.getChildren().addAll(theFPTS.getNav(), searchPane);
    return new Scene(splitPage, theFPTS.getWidth(), theFPTS.getHeight());
}

/*
*
* Define helper methods related to creation for parts of views
*
*/

/**
 * Helper method to format text fields with description.
 *
 * @return HBox
 */
private HBox createInputField(String description, TextField input) {
    HBox aField = new HBox();
    Label descriptionLabel = new Label(description);
    ObservableList<String> attributes =
            FXCollections.observableArrayList(
                    "",
                    "contains",
                    "starts with",
                    "exactly matches"
            );
    ComboBox searchConditions = new ComboBox(attributes);
    searchConditions.getSelectionModel().select(0);
    aField.getChildren().addAll(descriptionLabel, searchConditions, input);
    aField.setSpacing(10);
    return aField;
}

/**
 * Helper method to define fields to be entered
 *
 * @return VBox
 */
private VBox getCashAccountQueries() {
    VBox queries = new VBox();
    queries.getChildren().add(createInputField("Account name: ", mainInput));
    return queries;
}

/*
```

```
    *
    * Defines methods related to the Observer pattern
    *
    */

    /**
     * On update, the display of matches will change to reflect
     * the next matches.
     *
     * @param o   - Observable
     * @param arg - Object
     */
    @Override
    public void update(Observable o, Object arg) {
        displayMatches(s.getMatches());
    }

    /**
     * Displays the matches, one of which may be selected.
     *
     * @param matches
     */
    public void displayMatches(ArrayList<Searchable> matches) {
        matchDisplay.getChildren().clear();
        for (Searchable s : matches) {
            String symbol = s.getDisplayName();
            Button item = new Button(symbol);
            item.setStyle("-fx-background-color: white; -fx-text-fill: black;");
            item.setOnAction(new EventHandler<ActionEvent>() {
                @Override
                public void handle(ActionEvent e) {
                    mainInput.setText(symbol);
                }
            });
            matchDisplay.getChildren().add(item);
        }
    }
}

package controller;

import java.util.ArrayList;
import java.util.Observable;

/**
 * Implements step defined in CashAccountAlgorithm by obtaining amount of
money
```

```
 * from user. Defines next step to be implemented in subclasses and to be
executed
 * upon notification.
 *
 * @author Eric Epstein
 */
abstract public class ChangeCashAccountAlgorithm extends
CashAccountAlgorithm {

    /*
    * context data
    */
    protected ArrayList<Double> amounts;

    /**
     * Implements step defined in CashAccountAlgorithm. Adds itself as observer
     * for AmountInput.
     */
    @Override
    public void action() {
        amounts = new ArrayList<Double>();
        AmountInput amountInput = new AmountInput(theFPTS, amounts);
        amountInput.addObserver(this);
    }

    /**
     * Upon update of selected CashAccount, notifies the superclass.
     * Upon update of a defined input amount, calls next step to be defined in
     * subclasses.
     *
     * @param o    - Observable
     * @param args - Object
     */
    @Override
    public void update(Observable o, Object args) {
        if (amounts != null) {
            performTransaction();
        } else {
            super.update(o, args);
        }
    }

    /**
     * Defines abstract method representing the next step to be implemented in
     * subclasses.
     */
    abstract void performTransaction();
```

```
}

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package model;

import java.util.ArrayList;

/**
 * Defines one step in the Searcher that converts the object being
 * searched into a string representation of a CashAccount.
 *
 * @author Eric Epstein
 */
public class CashAccountSearcher extends Searcher {

    /**
     * Casts the Searchable object into CashAccount and provides information
     * as strings in ArrayList of one-element ArrayLists.
     *
     * @param s - Searchable
     * @return
     */
    public ArrayList<ArrayList<String>> getSearchableStrings(Searchable s) {
        ArrayList<String> searchableStrings = new ArrayList<String>();
        CashAccount ac = (CashAccount) (Object) s;
        ArrayList<ArrayList<String>> anObject = new
ArrayList<ArrayList<String>>();
        ArrayList<String> cashAccountItem = new ArrayList<String>();
        cashAccountItem.add(ac.getAccountName());
        anObject.add(cashAccountItem);
        return anObject;
    }
}

package controller;

import model.CashAccount;
import model.Deposit;
import model.Transaction;

/**
 * Implements final step in CashAccountAlgorithm by creating a Deposit object.
```

```
 *
 * @author Eric Epstein
 */
public class DepositCashAccountAlgorithm extends
ChangeCashAccountAlgorithm {

    /**
     * Creates a Deposit object with validated CashAccount at a validated amount.
     */
    @Override
    public void performTransaction() {
        double amount = amounts.get(0);

        CashAccount aC = theFPTS.getPortfolio().getCashAccount(c);

        Transaction t = new Deposit(aC, amount);
        theFPTS.getPortfolio().add(t);
        /*
        * Transitions to confirmation scene.
        */
        theFPTS.getStage().setScene(theFPTS.getConfirmationScene());
    }

}

package model;

import java.time.LocalDate;
import java.time.ZoneId;
import java.util.Date;

/**
 * Performs a deposit at a given amount on a given
 * CashAccount when called to do so
 *
 * @author Eric Epstein
 */
public class Deposit implements Transaction {
    private CashAccount c;
    private double amount;

    /**
     * Constructs a Deposit command
     *
     * @param c
     * @param amount
     */
```

```
    public Deposit(CashAccount c, double amount) {
        this.c = c;
        this.amount = amount;
    }

    /**
     * Executes the deposit
     */
    public void execute() {
        c.deposit(amount);
    }

    /**
     * returns a String representation for display
     *
     * @return String
     */
    public String toString() {
        Date theDate = c.getDateAdded();
        LocalDate localDate =
theDate.toInstant().atZone(ZoneId.systemDefault()).toLocalDate();
        String theDateString = (localDate.getMonthValue() + "/" +
                localDate.getDayOfMonth() +
                "/" + localDate.getYear());
        return "Deposited " + amount + " to " + c.getAccountName() + " on " +
theDateString;
    }

    /**
     * returns associated CashAccount
     *
     * @return CashAccount
     */
    public CashAccount getCashAccount() {
        return c;
    }

}

package controller;

import gui.FPTS;

/**
 * Defines the interface for displaying contents in a portfolio.
 *
 * @author Eric Epstein
```

```
 */
public interface Displayer {

    /**
     * Displays information given context data.
     *
     * @param theFPTS - the FPTS
     */
    public void display(FPTS theFPTS);

}

package model;

import java.util.ArrayList;

import static model.DataBase.ReadFile.readEquity;

/**
 * Searchable and available to be purchased by the user.
 * Contains information about the ticker symbol, equity name,
 * the value per share, the market sector, and the index.
 *
 * @author Epstein & Ian London
 */
public class Equity implements Searchable, EquityComponent, HoldingUpdatable
{

    /*
    * the identifying symbol
    */
    private String tickerSymbol;

    /*
    * the identifying name
    */
    private String equityName;

    /*
    * price per share
    */
    private double pricePerShare;

    /*
    * collection of indices
    */
    public ArrayList<String> indices;
```

```
    /*
    * collection of sectors
    */
    public ArrayList<String> sectors;

    /*
    * collection of Equity read from input
    */
    public static ArrayList<EquityComponent> EquityList = readEquity();

    /*
    * collection of Equity that matches a search criteria
    */
    private ArrayList<Equity> matches;

    /*
    * getEquityList() returns equities
    */
    public static ArrayList<EquityComponent> getEquityList() {
        return EquityList;
    }

    /*
    * Equity constructor takes 5 parameters
    *
    * @params : tickerSymbol - str
    *           equityName - str
    *           perShareValue - double
    *           indices - ArrayList<String>
    *           sectors - ArrayList<String>
    */
    public Equity(String tickerSymbol, String equityName, double perShareValue,
ArrayList<String> indices, ArrayList<String> sectors) {
        this.tickerSymbol = tickerSymbol;
        this.equityName = equityName;
        this.pricePerShare = perShareValue;
        this.indices = indices;
        this.sectors = sectors;
    }

    /*
    * getDisplayName returns ticker symbol
    *
    * @return tickerSymbol - String
    */
    public String getDisplayName() {
```

```
    return tickerSymbol;
}

/**
 * getValuePerShare returns price per share
 *
 * @return pricePerShare - double
 */
@Override
public double getValuePerShare() {
    return pricePerShare;
}

/**
 * getEquityName returns equity name
 *
 * @return equityName - String
 */
@Override
public String getEquityName() {
    return equityName;
}

/**
 * @return equityName - String
 */
@Override
public String getHoldingName() {
    return equityName;
}

/**
 * @return tickerSymbol - String
 */
@Override
public String getTickerSymbol() {
    return tickerSymbol;
}

/**
 * @return sectors - ArrayList<String>
 */
@Override
public ArrayList<String> getSectors() {
    return sectors;
}
```

```
    /**
     * @return indices - ArrayList<String>
     */
    @Override
    public ArrayList<String> getIndices() {
        return indices;
    }

    /**
     * @param e - EquityComponent
     *          <p>
     *          overrides EquityComponent but does nothing because
     *          Equity is a leaf node
     */
    @Override
    public void add(EquityComponent e) {
    }

    /**
     * @param e - EquityComponent
     *          <p>
     *          overrides EquityComponent but does nothing because
     *          Equity is a leaf node
     */
    @Override
    public void remove(EquityComponent e) {
    }
}

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package model;

import java.util.ArrayList;

/**
 * Defines the interface that manages and accesses the collection
 * of objects of which there are five accessible attributes.
 *
 * @author Eric Epstein
 */
public interface EquityComponent {

    /**
```

```
 * adds child to the composite
 *
 * @param e - EquityComponent
 */
public void add(EquityComponent e);

/**
 * removes child from composite
 *
 * @param e - EquityComponent
 */
public void remove(EquityComponent e);

/**
 * returns ticker symbol
 *
 * @return String
 */
public String getTickerSymbol();

/**
 * returns equity name
 *
 * @return String
 */
public String getEquityName();

/**
 * returns value per share
 *
 * @return double
 */
public double getValuePerShare();

/**
 * returns sectors
 *
 * @return ArrayList<String>
 */
public ArrayList<String> getSectors();

/**
 * returns indices
 *
 * @return ArrayList<String>
 */
public ArrayList<String> getIndices();
```

```
}

package model;

import java.util.ArrayList;
import java.util.List;

/**
 * Searchable and available to be purchased by the user.
 * Stores four attributes of an Equity in addition to an
 * ArrayList of equities from which value is calculated
 * <p>
 * Authors: Eric Epstein and Kaitlin Brockway
 */
public class EquityComposite implements Searchable, EquityComponent,
HoldingUpdatable {

    /*
    * name
    */
    private String name;

    /*
    * type
    */
    private String type;

    /*
    * collection of child Equity objects
    */
    private List<HoldingUpdatable> childEquities;

    /**
     * EquityComposite structure takes two parameters before adding
     * more attributes
     *
     * @params name - String
     * type - String
     */
    public EquityComposite(String name, String type) {
        this.name = name;
        this.type = type;
        childEquities = new ArrayList<HoldingUpdatable>();
    }

    /**
```

```
 * getDisplayName returns name
 *
 * @return name - String
 */
@Override
public String getDisplayName() {
    return name;
}

/**
 * @return name - String
 */
@Override
public String getTickerSymbol() {
    return name;
}

/**
 * @return name - String
 */
@Override
public String getEquityName() {
    return name;
}

/**
 * @return name - String
 */
@Override
public String getHoldingName() {
    return name;
}

/**
 * @return type - String
 */
public String getEquityType() {
    return type;
}

/**
 * Adds a child equity to composite
 *
 * @param e - HoldingUpdateable
 */
public void add(HoldingUpdatable e) {
    childEquities.add(e);
```

```
}

/**
 * getValuePerShare calculates the average price of the composite
 *
 * @return valuePerShare - double
 */
@Override
public double getValuePerShare() {
    double count = 0;
    double curVal;
    for (HoldingUpdatable se : childEquities) {
        curVal = se.getValuePerShare();
        count += curVal;
    }
    return count / childEquities.size();
}


/**
 * Returns empty list because a composite
 * does not have indices
 *
 * @return sectors - ArrayList<String>
 */
@Override
public ArrayList<String> getSectors() {
    return new ArrayList<String>();
}

/**
 * Returns an empty list because a composite
 * does not have indices
 *
 * @return indices - ArrayList<String>
 */
public ArrayList<String> getIndices() {
    return new ArrayList<String>();
}

/**
 * Removes child node from composite
 *
 * @param ec - EquityComponent
 */
@Override
public void remove(EquityComponent ec) {
```

```
        childEquities.remove((HoldingUpdatable) ec);
    }

    /**
     * Adds child node to composite
     *
     * @param ec - EquityComponent
     */
    @Override
    public void add(EquityComponent ec) {
        childEquities.add((HoldingUpdatable) ec);
    }

}

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package gui;

import controller.*;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
import model.DataBase.WriteFile;
import model.Equity;
import model.Portfolio;
import model.Simulator;
import model.User;

import java.io.*;

/**
 * Executes application and refers user to relevant functions.
 *
 * @author Eric Epstein, Kimberly Sookoo, Ian London, Kaitlyn Brockway, Luke
Veilleux
```

```
 */
public class FPTS extends Application {

    /*
    * context data associated with simulation
    */
    private static double simulationValue;
    private static Simulator currentSimulator;

    /*
    * constants that describe dimension of display
    */
    private final int WIDTH = 1200;
    private final int HEIGHT = 600;

    /*
    * stage
    */
    private Stage thestage;

    /*
    * associated portfolio
    */
    public Portfolio p;

    /*
    * current user
    */
    private static User currentUser;

    /*
    * global reference to loginController
    */
    LoginController loginController;

    /*
    * copy of itself
    */
    private static FPTS self;

    /**
     * Starts the application display and loads users
     *
     * @param primaryStage
     * @throws IOException
     */
    @Override
```

```
    public void start(Stage primaryStage) throws IOException {
        self = this;
        thestage = primaryStage;
        //p = new Portfolio();

        /*
        * Fills the User static class
        */
        User.fillUsers();

        // Fills the Equity static class with whats in the equities.csv file
        Equity.getEquityList();

        /*
        * Sets homepage using FXML loader
        */
        Parent root = FXMLLoader.load(getClass().getResource("LoginPage.fxml"));
        Scene loginScene = new Scene(root, WIDTH, HEIGHT);

        try {
            thestage.setScene(createLogInScene());
        } catch (Exception e) {
            //e.printStackTrace();
        }

        self = this;

        thestage.setScene(loginScene);
        thestage.show();

    }


    /**
     * Returns home page
     *
     * @return Scene
     */
    public Scene getHomeScene() {
        Scene scene = null;
        try {
            Parent parent =
FXMLLoader.load(getClass().getResource("HomePage.fxml"));
            scene = new Scene(parent);
        } catch (IOException e) {
            e.printStackTrace();
        }
```

```
    return scene;
}

/**
 * Returns scene indicating confirmation of a user's action
 *
 * @return Scene
 */

public Scene getConfirmationScene() {
    Label confirmation = new Label("Update completed");
    VBox split = new VBox();
    split.getChildren().addAll(getNav(), confirmation);
    return new Scene(split, WIDTH, HEIGHT);
}

/**
 * Returns scene indicating error on the part of the user
 *
 * @return Scene
 */
public Scene getErrorScene() {
    Label confirmation = new Label("Error");
    VBox split = new VBox();
    split.getChildren().addAll(getNav(), confirmation);
    return new Scene(split, WIDTH, HEIGHT);
}

/**
 * Returns login page loaded in FXMLLoader
 *
 * @return Scene
 */
public Scene createLogInScene() throws IOException {
    Parent root = FXMLLoader.load(getClass().getResource("LoginPage.fxml"));
    Scene scene = new Scene(root, WIDTH, HEIGHT);

    thestage.setTitle("Financial Portfolio Tracking System");
    return scene;
}

/**
 * Returns register page loaded in FXMLLoader
 *
 * @return Scene
 */
public Scene createRegisterPage() throws IOException {
```

```
        Parent root =
FXMLLoader.load(getClass().getResource("RegisterPage.fxml"));
        Scene scene = new Scene(root, WIDTH, HEIGHT);

        thestage.setScene(scene);
        thestage.setTitle("Financial Portfolio Tracking System");

        return scene;
    }


    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        if (args.length >= 2) {
            if (args[0].equals("-delete")) {
                String userID = args[1];
                File csv = new File("JavaFXApp/src/model/DataBase/UserData.csv");
                File csvTemp = new File("JavaFXApp/src/model/DataBase/
UserDataTemp.csv");
                String line;
                try {
                    BufferedReader reader = new BufferedReader(new
FileReader(csv));
                    BufferedWriter writer = new BufferedWriter(new
FileWriter(csvTemp));
                    while ((line = reader.readLine()) != null) {
                        String[] split = line.split(",");
                        if (split[0].equals(userID)) {
                            System.out.println("Deleting " + userID);
                            continue;
                        }
                        writer.write(split[0] + "," + split[1]);
                        writer.newLine();
                    }
                    writer.close();
                    reader.close();
                    csvTemp.renameTo(csv);

                    File directory = new File("JavaFXApp/src/model/Database/
Portfolios/" + userID);
                    if (directory.exists()) {
                        System.out.println("Has " + userID + " been deleted?");
                        File transFile = new File(directory, "/Trans.csv");
                        File cashFile = new File(directory, "/Cash.csv");
```

```
                File holdingsFile = new File(directory, "/Holdings.csv");
                transFile.delete();
                cashFile.delete();
                holdingsFile.delete();
                directory.delete();
            }
        } catch (Exception e) {

        }
    }
    launch(args);
}


/**
 * Returns portfolio
 *
 * @return Portfolio
 */
public Portfolio getPortfolio() {
    return p;
}

/**
 * Constructs navigation for relevant subsystems
 *
 * @return HBox
 */
public HBox getNav() {
    HBox nav = new HBox();
    Button aButton;

    /*
    * Button to visit Home
    */
    aButton = new Button();
    aButton.setText("Home");
    aButton.setOnAction(new EventHandler<ActionEvent>() {
        @Override
        public void handle(ActionEvent event) {
            thestage.setScene(getHomeScene());
        }
    });
    nav.getChildren().add(aButton);

    /*
```

```
* Button to buy Equity
*/
aButton = new Button();
aButton.setText("Buy Holding");
aButton.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {
        HoldingAlgorithm eqUpdater = new BuyHoldingAlgorithm();
        eqUpdater.process(self);
    }
});
nav.getChildren().add(aButton);

/*
* Button to sell equity
*/
aButton = new Button();
aButton.setText("Sell Holding");
aButton.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {
        HoldingAlgorithm eqUpdater = new SellHoldingAlgorithm();
        eqUpdater.process(self);
    }
});
nav.getChildren().add(aButton);

/*
* Button to display portfolio
*/
aButton = new Button();
aButton.setText("Display Portfolio");
//TODO:Action to be set
aButton.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {
        Displayer pd = new PortfolioDisplayer();
        pd.display(getSelf());
    }

});
nav.getChildren().add(aButton);

/*
* Button to view Transaction history
*/
aButton = new Button();
```

```
        aButton.setText("History");
        aButton.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                Displayer td = new TransactionDisplayer();
                td.display(getSelf());
            }

        });
        nav.getChildren().add(aButton);

        /*
        * Button to remove CashAccount
        */
        aButton = new Button();
        aButton.setText("Remove Cash Account");
        aButton.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                CashAccountAlgorithm cashAccountAlgorithm = new
RemoveCashAccountAlgorithm();
                cashAccountAlgorithm.process(self);
                //eqUpdater.process(self);
            }
        });
        nav.getChildren().add(aButton);

        /*
        * Button to Deposit CashAccount
        */
        aButton = new Button();
        aButton.setText("Deposit");
        aButton.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                CashAccountAlgorithm cashAccountAlgorithm = new
DepositCashAccountAlgorithm();
                cashAccountAlgorithm.process(self);
                //eqUpdater.process(self);
            }
        });
        nav.getChildren().add(aButton);

        /*
        * Button to withdraw from CashAccout
        */
        aButton = new Button();
```

```
        aButton.setText("Withdraw");
        aButton.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                CashAccountAlgorithm cashAccountAlgorithm = new
WithdrawCashAccountAlgorithm();
                cashAccountAlgorithm.process(self);
                //eqUpdater.process(self);
            }
        });
        nav.getChildren().add(aButton);

        /*
        * Button to transfer from one CashAccount to another
        */
        aButton = new Button();
        aButton.setText("Transfer");
        aButton.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                CashAccountAlgorithm cashAccountAlgorithm = new
TransferCashAccountAlgorithm();
                cashAccountAlgorithm.process(self);
                //eqUpdater.process(self);
            }
        });
        nav.getChildren().add(aButton);

        /*
        * Button to create CashAccount
        */
        aButton = new Button();
        aButton.setText("Add Cash Account");
        aButton.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                CashAccountCreator cashAccountCreator = new
CashAccountCreator(getSelf());
                //eqUpdater.process(self);
            }
        });
        nav.getChildren().add(aButton);

        /*
        The following code is commented out
        because it complies with our current
        design decision that will be revisited
```

```
        in the next release.

        Create/Delete Portfolio Button disabled


        nav.getChildren().add(aButton);

        //Button to add/remove Portfolio

        Button managePortfolio = new Button();
        WriteFile writeFile = new WriteFile();
        currentUser.setMyPortfolio(this.getPortfolio());

        if (writeFile.hasPortfolio(currentUser)) {
            managePortfolio.setText("Remove Portfolio");
        } else {
            managePortfolio.setText("Add Portfolio");
        }
        managePortfolio.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {

                if (writeFile.hasPortfolio(currentUser)) {
                    writeFile.removePortfolioForUser(currentUser);
                    managePortfolio.setText("Add Portfolio");
                } else {
                    writeFile.createPortfolioForUser(currentUser);
                    managePortfolio.setText("Remove Portfolio");
                }
            }
        });
        nav.getChildren().add(managePortfolio);
        */

        /*
        * Button to Logout
        */
        aButton = new Button();
        aButton.setText("Log out");
        //Setting an action for the logout button
        aButton.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent e) {
                try {
                    Stage stage = new Stage();
                    Scene scene = new
    Scene(FXMLLoader.load(getClass().getResource("../gui/LogoutPage.fxml")));
```

```
                stage.setScene(scene);
                stage.show();
            } catch (Exception ex) {
            }
            thestage.show();
        }
    });
    nav.getChildren().add(aButton);

    return nav;
}

/**
 * Sets simulation value
 *
 * @param value
 */
public static void setSimulationValue(double value) {
    simulationValue = value;
}

/**
 * Returns simulation value
 *
 * @return double
 */
public static double getSimulationValue() {
    return simulationValue;
}

/**
 * Sets current simulation
 *
 * @param curSim - Simulator
 */
public static void setCurrentSimulator(Simulator curSim) {
    currentSimulator = curSim;
}

/**
 * Returns current simulation
 *
 * @return Simulator
 */
public static Simulator getCurrentSimulator() {
    return currentSimulator;
}
```

```
    /**
     * Returns current user
     *
     * @return User
     */
    public User getCurrentUser() {
        return currentUser;
    }

    /**
     * Returns login ID of current user
     *
     * @return String
     */
    public static String getCurrentUserID() {
        return currentUser.getLoginID();
    }

    /**
     * Returns self
     *
     * @return FPTS
     */
    public static FPTS getSelf() {
        return self;
    }

    /**
     * Returns indicator of portfolio existence
     *
     * @param user
     * @return boolean
     */
    public boolean hasPortfolio(User user) {
        File directory = new File("JavaFXApp/src/model/Database/Portfolios/" +
user.getLoginID());
        if (directory.exists()) {
            return true;
        }
        return false;
    }

    /**
     * Sets user object to the current user
     *
     * @param user - User
```

```
     */
    public void setCurrentUser(User user) {
        this.currentUser = user;
        WriteFile writeFile = new WriteFile();
        if (!(hasPortfolio(currentUser))) {
            writeFile.createPortfolioForUser(currentUser);
        }
        p = new Portfolio();
    }

    /**
     * Returns height of stage
     *
     * @return int
     */
    public int getHeight() {
        return HEIGHT;
    }

    /**
     * Returns width of stage
     *
     * @return int
     */
    public int getWidth() {
        return WIDTH;
    }

    /**
     * Returns primary stage
     *
     * @return Stage
     */
    public Stage getStage() {
        return thestage;
    }
}
```

```
package model;

import java.util.ArrayList;
import java.util.Date;

import static model.DataBase.ReadFile.readHoldings;
```

```
/**
 * Holds an individual equity acquisition made by the user of the application.
 * Stores the ticker symbol, name, sectors, and indices, value per share,
 * number of shares, and overall value.
 *
 * @author Eric Epstein
 */
public class Holding implements Searchable, HoldingUpdatable {

    /*
    * ticker symbol
    */
    private String tickerSymbol;

    /*
    * holding name
    */
    private String holdingName;

    /*
    * initial price per share
    */
    private double initialPricePerShare;

    /*
    * number of shares
    */
    private int numOfShares;

    /*
    * value per share
    */
    private double valuePerShare;

    /*
    * current value
    */
    private double currentValue;

    /*
    * acquistion date
    */
    private Date acquisitionDate;

    /*
    * indices
    */
```

```
    private ArrayList<String> indices;

    /*
    * sectors
    */
    private ArrayList<String> sectors;

    /*
    *
    */
    public static ArrayList<Holding> holdingArrayList = readHoldings();


    /**
     * Constructor used when a user manually adds a Holding.
     *
     * @author Eric Epstein and Kaitlyn Brockway
     */
    public Holding(String tickerSymbol, String equityName, double valuePerShare,
int numOfShares, Date acquisitionDate, ArrayList<String> indices,
ArrayList<String> sectors) {
        this.tickerSymbol = tickerSymbol;
        this.holdingName = equityName;
        this.numOfShares = numOfShares;
        this.valuePerShare = valuePerShare;
        this.currentValue = numOfShares * valuePerShare;
        this.acquisitionDate = acquisitionDate;
        //this.cashAccount = cashAccount;
        //extras = new ArrayList<String>();
    }

    /**
     * returns value per share
     *
     * @return double
     */
    @Override
    public double getValuePerShare() {
        return valuePerShare;
    }

    /**
     * returns display name
     *
     * @return String
     */
    @Override
```

```
    public String getDisplayName() {
        return tickerSymbol;
    }

    /**
     * returns symbol
     *
     * @return String
     */
    public String getSymbol() {
        return tickerSymbol;
    }

    /**
     * returns symbol, overrides HoldingUpdatable interface
     *
     * @return String
     */
    @Override
    public String getTickerSymbol() {
        return tickerSymbol;
    }

    /**
     * returns String for display
     *
     * @return String
     */
    @Override
    public String toString() {
        return tickerSymbol + ", " + holdingName + ", " + numOfShares + " shares,
$" + valuePerShare + " per share, $" + currentValue + " current value";
    }


    /**
     * returns holding name
     *
     * @return String
     */
    @Override
    public String getHoldingName() {
        return holdingName;
    }

    /**
     * returns number of shares
```

```
     *
     * @return int
     */
    public int getNumOfShares() {
        return numOfShares;
    }

    /**
     * returns overall value
     *
     * @return double
     */
    public double getValue() {
        return currentValue;
    }

    public Date getAcquisitionDate() {
        return acquisitionDate;
    }

    /**
     * returns overall value
     *
     * @return double
     */
    public double getCurrentValue() {
        return currentValue;
    }

    /**
     * returns sectors
     *
     * @return ArrayList<String>
     */
    @Override
    public ArrayList<String> getSectors() {
        return new ArrayList<String>();
    }

    /**
     * returns indices
     *
     * @return ArrayList<String>
     */
    @Override
    public ArrayList<String> getIndices() {
        return new ArrayList<String>();
```

```
    }

    /**
     * subtracts the number of shares by a specified amount
     * and updates value
     *
     * @param numOfSharesAdded
     */
    public void addShares(int numOfSharesAdded) {
        numOfShares += numOfSharesAdded;
        currentValue += (numOfSharesAdded * valuePerShare);
    }

    /**
     * adds the number of shares by a specified amount
     * and updates value
     *
     * @param numOfSharesSubtracted
     */

    public void subtractShares(int numOfSharesSubtracted) {
        numOfShares -= numOfSharesSubtracted;
        currentValue -= (numOfSharesSubtracted * valuePerShare);
    }


}

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package controller;

import gui.FPTS;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.ComboBox;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
```

```
import javafx.stage.Stage;
import model.*;

import java.util.ArrayList;
import java.util.Observable;
import java.util.Observer;

/**
 * Defines steps that are common to buying Equity and selling Holding objects.
 *
 * @author Eric Epstein
 */
abstract public class HoldingAlgorithm implements Observer {

    /*
    * Reference to self
    */
    private HoldingAlgorithm self;

    /*
    * Display of matches
    */
    private VBox matchDisplay;

    /*
    * Input field from the user
    */
    protected TextField mainInput;

    /*
    * Context data and derived information
    */
    protected FPTS theFPTS;
    protected Stage theStage;
    private double HEIGHT;
    private double WIDTH;

    /*
    * Search algorithm
    */
    protected Searcher s;

    /*
    * user inputs to be processed in child algorithms
    */
    protected double pricePerShare;
    protected int numOfShares;
```

```
protected double valuePerShare;

/*
* Computations from user inputs
*/
protected HoldingUpdatable equityOfInterest;
protected CashAccount cashAccountOfInterest;

/*
* Delegates obtaining relevant context data to child algorithms
*/
abstract void establishContext();

/*
* Delegates what should be searched to child algorithms
*/
abstract ArrayList<Searchable> getToBeSearched();

/**
 * Establishes context derived from child algorithms then constructs
 * search scene with specialized context.
 *
 * @param anFPTS - FPTS
 */
public void process(FPTS anFPTS) {
    theFPTS = anFPTS;
    establishContext();
    theStage = theFPTS.getStage();
    self = this;
    mainInput = new TextField();
    matchDisplay = new VBox();
    HEIGHT = theFPTS.getHeight();
    WIDTH = theFPTS.getWidth();
    theStage.setScene(getFirstSearchScene());
    theStage.show();

}

/*
 *
 * Defines abstract methods to be implemented in subclasses
 *
 */

/*
* Delegates method of purchase/sale inside FPTS to subclass
*/
```

```
    abstract void processInsideFPTS();

    /*
    * Delegates method of purchase/sale outside FPTS to sublcass
    */
    abstract void processOutsideFPTS();

    /*
    *
    * Defines views used throughout algorithm
    *
    */

    /**
     * Constructs Scene where the user searches and selects Equity or Holding,
     * depending on context established previously in given subclass.
     *
     * @return
     */
    public Scene getFirstSearchScene() {
        ArrayList<Searchable> toBeSearched = getToBeSearched();
        s = new SearchedHoldingSearcher();
        s.addObserver(self);
        VBox queries = getHoldingQueries();
        return getSearchScene(toBeSearched, queries,
getTransitionAfterHolding());
    }

    /**
     * Constructs Scene where the user searches and selects CashAccount
     *
     * @return Scene
     */
    public Scene getSecondSearchScene() {
        s = new CashAccountSearcher();
        s.addObserver(self);
        matchDisplay.getChildren().clear();
        mainInput.setText("");
        ArrayList<Searchable> toBeSearched =
theFPTS.getPortfolio().getCashAccountSearchables();
        VBox queries = getCashAccountQueries();
        return getSearchScene(toBeSearched, queries,
getTransitionAfterCashAccount());
    }

    /**
     * Helper scene constructor for the user to select and search any given
```

```
     * Searchable type.
     * <p>
     * Precondition: Searcher algorithm is preset
     *
     * @param toBeSearched - ArrayList<Searchable>
     * @param queries      - VBox of text field
     * @param actionBtn    - controller that determines what to do next
     * @return
     */
    private Scene getSearchScene(ArrayList<Searchable> toBeSearched, VBox
queries, Button actionBtn) {
        VBox splitPage = new VBox();

        VBox searchPane = new VBox();

        actionBtn.setVisible(false);

        Button searchBtn = new Button();
        searchBtn.setText("Search");
        searchBtn.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent e) {
                /*
                * Calls Searcher algorithm specified in the step of program execution.
                */
                s.search(queries.getChildren(), toBeSearched);
                actionBtn.setVisible(true);
            }
        });

        HBox forAction = new HBox();
        forAction.getChildren().addAll(queries, actionBtn);
        searchPane.getChildren().addAll(forAction, searchBtn, matchDisplay);
        splitPage.getChildren().addAll(theFPTS.getNav(), searchPane);
        return new Scene(splitPage, WIDTH, HEIGHT);
    }


    /**
     * Constructs Scene to get the following user input:
     * -price per share
     * -number of shares
     * -whether purchase/sale was made outside or inside the FPTS
     *
     * @return Scene
     */
    public Scene getAdditionalInfoScene() {
```

```
        VBox splitPage = new VBox();
        VBox searchPane = new VBox();
        VBox queries = new VBox();
        HBox aField = new HBox();

        /*
        * Defines first field : price per share
        */
        TextField pricePerShareField = new TextField();
        pricePerShareField.setText("" + equityOfInterest.getValuePerShare());
        aField.getChildren().addAll(new Label("Price per share: "),
pricePerShareField);
        queries.getChildren().add(aField);

        /*
        * Defines second field : number of shares
        */
        aField = new HBox();
        TextField numOfSharesField = new TextField();
        aField.getChildren().addAll(new Label("Number of shares: "),
numOfSharesField);
        queries.getChildren().add(aField);

        /*
        * Defines third field : whether or not the sale is made outside or inside
FPTS.
        */
        aField = new HBox();
        ObservableList<String> attributes =
                FXCollections.observableArrayList(
                        "Outside FPTS",
                        "Use existing cash account"
                );
        ComboBox searchConditions = new ComboBox(attributes);
        searchConditions.getSelectionModel().select(0);
        aField.getChildren().addAll(new Label("Target source: "), searchConditions);

        queries.getChildren().add(aField);

        Button submitButton = new Button();
        submitButton.setText("Proceed");
        submitButton.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent e) {

                /*
```

```
            * Validates user input with helper methods
            */
            boolean isValid = isValid(pricePerShareField) &&
isValid(numOfSharesField);

            /*
            * If input is valid, analyze response and determine which step in the
            * algorithm to go.
            */
            if (isValid) {
               pricePerShare = Double.parseDouble(pricePerShareField.getText());
               numOfShares = Integer.parseInt(numOfSharesField.getText());

               switch (searchConditions.getValue().toString()) {
                  case ("Outside FPTS"):
                     /*
                     * Delegates purchase/sale outside FPTS in respective subclass
                     */
                     processOutsideFPTS();
                     theStage.setScene(theFPTS.getConfirmationScene());
                     break;
                  case ("Use existing cash account"):
                     /*
                     * Constructs another scene to get CashAccount if purchase/
sale
                     * is made inside FPTS.
                     */
                     theStage.setScene(getSecondSearchScene());
                     break;
               }
            } else {
               pricePerShareField.setText("INVALID");
            }
         }
      });

      searchPane.getChildren().addAll(queries, submitButton);
      splitPage.getChildren().addAll(theFPTS.getNav(), searchPane);
      Scene additionalInfoScene = new Scene(splitPage, WIDTH, HEIGHT);
      return additionalInfoScene;

   }

   /**
    * Helper method to validate numerical value from user input
    *
    * @param inputAmount
```

```
     * @return
     */
    private boolean isValid(TextField inputAmount) {
        if (inputAmount.getText() == null || inputAmount.getText().equals("")) {
            return false;
        }

        String inputAmountString = inputAmount.getText();

        try {
            Double.parseDouble(inputAmountString);
        } catch (Exception e) {
            return false;
        }

        Double inputAmountDouble = Double.parseDouble(inputAmountString);

        if (inputAmountDouble < 0) {
            return false;
        }

        return true;

    }

    /*
    *
    * Defines Button controls
    *
    */

    /**
     * Defines behavior for submit button after selecting CashAccount
     *
     * @return Button
     */
    public Button getTransitionAfterCashAccount() {
        Button actionBtn = new Button();
        actionBtn.setText("Proceed");

        actionBtn.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent e) {
                /*
                * Validates CashAccount and calls subclass algorithm to process
                * sale/purchase inside FPTS.
                */
```

```
            if (mainInput.getText() != null && s.getMatch(mainInput.getText()) !=
null) {
                cashAccountOfInterest = (CashAccount)
s.getMatch(mainInput.getText());
                processInsideFPTS();
            } else {
                mainInput.setText("INVALID");
            }
        }
    });

    return actionBtn;

}

/**
 * Defines button action to update equityOfInterest, which may reflect
 * Equity or Holding to be bought or sold, respectively.
 *
 * @return Button
 */
public Button getTransitionAfterHolding() {
    Button actionBtn = new Button();
    actionBtn.setText("Proceed");

    actionBtn.setOnAction(new EventHandler<ActionEvent>() {
        @Override
        public void handle(ActionEvent e) {
            /*
            * Validates HoldingUpdatable (Equity or Holding) then transitions
            * to next view.
            */
            if (mainInput.getText() != null && s.getMatch(mainInput.getText()) !=
null) {
                equityOfInterest = (HoldingUpdatable)
s.getMatch(mainInput.getText());
                theStage.setScene(getAdditionalInfoScene());
            } else {
                mainInput.setText("INVALID");
            }
        }
    });

    return actionBtn;
}

/*
```

```
*
* Defines methods related to the Observer pattern
*
*/

/**
 * On update, calls method to display new set of matches.
 *
 * @param o
 * @param arg
 */
@Override
public void update(Observable o, Object arg) {
    displayMatches(s.getMatches());
}

/**
 * Helper method to display matches produced by Searcher algorithm
 *
 * @param matches
 */
private void displayMatches(ArrayList<Searchable> matches) {
    /**
     * For each match in matches, produce button that may populate the input
     * field.
     */
    matchDisplay.getChildren().clear();
    for (Searchable s : matches) {
        String symbol = s.getDisplayName();
        Button item = new Button(symbol);
        item.setStyle("-fx-background-color: white; -fx-text-fill: black;");
        item.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent e) {
                mainInput.setText(symbol);
            }
        });
        matchDisplay.getChildren().add(item);
    }

}

/*
*
* Define methods related to creation for parts of views
*
*/
```

```
    /**
     * Helper method to create uniquely formatted fields for user input.
     * These fields have a description, search options, and a TextField.
     *
     * @param description
     * @param input
     * @return HBox
     */
    private HBox createInputField(String description, TextField input) {
        HBox aField = new HBox();
        Label descriptionLabel = new Label(description);
        /**
         * Search options are "" (ignore), "contains," "starts with," and "exactly
         * matches"
         */
        ObservableList<String> attributes =
                FXCollections.observableArrayList(
                        "",
                        "contains",
                        "starts with",
                        "exactly matches"
                );
        ComboBox searchConditions = new ComboBox(attributes);
        searchConditions.getSelectionModel().select(0);
        aField.getChildren().addAll(descriptionLabel, searchConditions, input);
        aField.setSpacing(10);
        return aField;
    }

    /**
     * This helper method returns queries in relation to searching/selecting an
     * Equity or a Holding.
     *
     * @return VBox
     */
    private VBox getHoldingQueries() {
        VBox queries = new VBox();
        queries.getChildren().add(createInputField("Ticker symbol: ", mainInput));
        queries.getChildren().add(createInputField("Holding name: ", new
TextField()));
        queries.getChildren().add(createInputField("Index: ", new TextField()));
        queries.getChildren().add(createInputField("Sector: ", new TextField()));
        return queries;
    }
```

```
    /**
     * This helper method returns queries in relation to searching/selecting a
     * CashAccount.
     *
     * @return VBox
     */
    private VBox getCashAccountQueries() {
        VBox queries = new VBox();
        queries.getChildren().add(createInputField("Account name: ", mainInput));
        return queries;
    }

}

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package model;

import java.util.ArrayList;

/**
 * @author ericepstein
 */
public interface HoldingUpdatable {
    public String getTickerSymbol();

    public String getHoldingName();

    public double getValuePerShare();

    public ArrayList<String> getIndices();

    public ArrayList<String> getSectors();
    //public void add(HoldingUpdatable e);
}

package controller;

import gui.FPTS;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
```

```
import javafx.scene.Node;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.chart.PieChart;
import javafx.scene.control.Label;
import javafx.stage.Stage;
import model.CashAccount;
import model.Holding;
import model.Portfolio;

import java.io.IOException;
import java.net.URL;
import java.util.ResourceBundle;

/**
 * Controller class for the HomePage. Extends MenuController which holds all of the functionality for the MenuBar
 * on the page. Controls the actions that occur when a button is pressed on the screen.
 * Created by Luke on 3/02/2016.
 */
public class HomeController extends MenuController {

    FPTS fpts = FPTS.getSelf();
    @FXML
    private PieChart valueChart = new PieChart();
    @FXML
    private Label valueLabel = new Label();

    /**
     * Method that is called if the BuyEquities button is pressed on the HomePage. Loads the buy equities page.
     *
     * @param event - ActionEvent - the event that caused this method to run.
     */
    @FXML
    protected void handlePortfolioButtonPressed(ActionEvent event) {
        Displayer pd = new PortfolioDisplayer();
        pd.display(fpts);
    }

    /**
     * Method that is called if the BuyEquities button is pressed on the HomePage. Loads the buy equities page.
     *
     * @param event - ActionEvent - the event that caused this method to run.
     */
```

```
    @FXML
    protected void handleBuyEquityButtonPressed(ActionEvent event) {
        HoldingAlgorithm eqUpdater = new BuyHoldingAlgorithm();
        eqUpdater.process(FPTS.getSelf());
    }

    /**
     * Method that is called when the Simulate button is pressed on the
HomePage. Loads the Simulation page.
     *
     * @param event - ActionEvent - the event that caused this method to run
     * @throws IOException - Exception is thrown if the SimulatePage.fxml is not
found.
     */
    @FXML
    protected void handleSimulateButtonPressed(ActionEvent event) throws
IOException {
        Parent parent = FXMLLoader.load(getClass().getResource("../gui/
SimulatePage.fxml"));
        Scene scene = new Scene(parent);
        Stage app_stage = (Stage) ((Node)
event.getSource()).getScene().getWindow();
        app_stage.setScene(scene);
        app_stage.show();
    }

    /**
     * Method that is called when the Logout button is pressed on the HomePage.
Loads the Logout pop-up window
     * which in turn will logout the user depending on what option they select.
     *
     * @param event - ActionEvent - the event that caused this method to run
     * @throws IOException - Exception is thrown if the LogoutPage.fxml is not
found.
     */
    @FXML
    protected void handleLogoutButtonPressed(ActionEvent event) throws
IOException {
        Stage stage = new Stage();
        Scene scene = new Scene(FXMLLoader.load(getClass().getResource("../
gui/LogoutPage.fxml")));
        stage.setScene(scene);
        stage.show();
    }

    /**
     * Method used to initialize items on the HomePage, mainly used to fill the
```

```
PieChart on the page and
     * put the current value of the portfolio into the label on the page.
     */
    @Override
    public void initialize(URL location, ResourceBundle resources) {
        if (fpts.hasPortfolio(fpts.getCurrentUser())) {
            Portfolio p = fpts.getPortfolio();
            double cashAccountValue = 0.0;
            double equityTotalValue = 0.0;
            try {
                for (CashAccount c : p.getCashAccounts()) {
                    cashAccountValue += c.getValue();
                }
                for (Holding h : p.getHoldings()) {
                    equityTotalValue += h.getValue();
                }
            } catch (NullPointerException e) {
            }
            ObservableList<PieChart.Data> pieChartData =
                    FXCollections.observableArrayList(
                            new PieChart.Data("Holdings", equityTotalValue),
                            new PieChart.Data("Cash Accounts", cashAccountValue));
            valueChart.setData(pieChartData);
            valueChart.setTitle("Portfolio");
            valueLabel.setText("Current Portfolio Value: $" + (cashAccountValue +
equityTotalValue));

        }

    }
}

<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.chart.PieChart?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>
<?import javafx.scene.text.*?>
<AnchorPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity"
minWidth="-Infinity" prefHeight="600.0"
            prefWidth="800.0" xmlns="http://javafx.com/javafx/8" xmlns:fx="http://
javafx.com/fxml/1"
            fx:controller="controller.HomeController">
    <children>
        <MenuBar fx:id="myMenuBar" layoutY="2.0" minHeight="-Infinity"
minWidth="-Infinity" prefHeight="25.0"
                 prefWidth="800.0" AnchorPane.topAnchor="2.0">
```

```
        <menus>
          <Menu mnemonicParsing="false" text="File">
            <items>
              <MenuItem fx:id="home" mnemonicParsing="false"
onAction="#handleHomeMenuItemPressed"
                        text="Home"/>
              <MenuItem fx:id="save" mnemonicParsing="false"
onAction="#handleSaveMenuItemPressed"
                        text="Save"/>
              <MenuItem fx:id="Logout" mnemonicParsing="false"
onAction="#handleLogoutMenuItemPressed"
                        text="Logout"/>
              <SeparatorMenuItem mnemonicParsing="false"/>
              <MenuItem fx:id="Exit" mnemonicParsing="false"
onAction="#handleExitMenuItemPressed"
                        text="Exit"/>
            </items>
          </Menu>
          <Menu mnemonicParsing="false" text="Equities">
            <items>
              <MenuItem fx:id="showPortfolio" mnemonicParsing="false"
                        onAction="#handlePortfolioMenuItemPressed" text="Show
Portfolio"/>
              <MenuItem fx:id="buyEquities" mnemonicParsing="false"
                        onAction="#handleBuyEquitiesMenuItemPressed"
text="Buy Equities"/>
              <MenuItem fx:id="sellEquities" mnemonicParsing="false"
                        onAction="#handleSellEquitiesMenuItemPressed"
text="Sell Equities"/>
            </items>
          </Menu>
          <Menu mnemonicParsing="false" text="Cash Account">
            <items>
              <MenuItem fx:id="createCA" mnemonicParsing="false"
onAction="#handleCreateMenuItemPressed"
                        text="Create New"/>
              <MenuItem fx:id="deposit" mnemonicParsing="false"
onAction="#handleDepositMenuItemPressed"
                        text="Deposit"/>
              <MenuItem fx:id="withdraw" mnemonicParsing="false"
onAction="#handleWithdrawMenuItemPressed"
                        text="Withdraw"/>
              <MenuItem fx:id="transfer" mnemonicParsing="false"
onAction="#handleTransferMenuItemPressed"
                        text="Transfer"/>
              <SeparatorMenuItem mnemonicParsing="false"/>
              <MenuItem fx:id="remove" mnemonicParsing="false"
```

```
onAction="#handleRemoveMenuItemPressed"
                              text="Remove Account"/>
                  </items>
               </Menu>
               <Menu mnemonicParsing="false" text="Help">
                  <items>
                     <MenuItem fx:id="about" mnemonicParsing="false"
onAction="#handleAboutMenuItemPressed"
                              text="About"/>
                  </items>
               </Menu>
            </menus>
         </MenuBar>
         <GridPane layoutX="-3.0" layoutY="29.0" prefHeight="573.0"
prefWidth="802.0">
            <columnConstraints>
               <ColumnConstraints hgrow="SOMETIMES" maxWidth="496.0"
minWidth="10.0" prefWidth="399.33331298828125"/>
               <ColumnConstraints hgrow="SOMETIMES"
maxWidth="402.66668701171875" minWidth="10.0"
                                  prefWidth="402.66668701171875"/>
            </columnConstraints>
            <rowConstraints>
               <RowConstraints maxHeight="317.99998474121094"
minHeight="10.0" prefHeight="118.33333587646484"
                              vgrow="SOMETIMES"/>
               <RowConstraints maxHeight="225.00000762939453"
minHeight="10.0" prefHeight="166.66666412353516"
                              vgrow="SOMETIMES"/>
               <RowConstraints maxHeight="174.33331298828125"
minHeight="10.0" prefHeight="158.33331298828125"
                              vgrow="SOMETIMES"/>
               <RowConstraints maxHeight="132.33334350585938"
minHeight="10.0" prefHeight="132.33334350585938"
                              vgrow="SOMETIMES"/>
            </rowConstraints>
            <children>
               <Label alignment="CENTER" prefHeight="27.0" prefWidth="329.0"
text="Financial Portfolio Tracking System"
                     GridPane.halignment="CENTER"
GridPane.valignment="CENTER">
                  <font>
                     <Font name="System Bold Italic" size="18.0"/>
                  </font>
               </Label>
               <Label fx:id="label" mouseTransparent="true" prefHeight="17.0"
prefWidth="404.0" GridPane.columnSpan="2"
```

```
                GridPane.halignment="CENTER" GridPane.rowIndex="3"/>
            <HBox alignment="CENTER" prefHeight="100.0" prefWidth="200.0"
spacing="75.0" GridPane.columnSpan="2"
                GridPane.halignment="CENTER" GridPane.hgrow="SOMETIMES"
GridPane.rowIndex="3"
                GridPane.valignment="CENTER"
GridPane.vgrow="SOMETIMES">
              <children>
                <Button fx:id="pinfo" mnemonicParsing="false"
onAction="#handlePortfolioButtonPressed"
                    prefHeight="25.0" prefWidth="120.0" text="Portfolio Info">
                  <font>
                    <Font size="14.0"/>
                  </font>
                </Button>
                <Button fx:id="buyEquity" mnemonicParsing="false"
onAction="#handleBuyEquityButtonPressed"
                    prefHeight="25.0" prefWidth="120.0" text="Buy Equities">
                  <font>
                    <Font size="14.0"/>
                  </font>
                </Button>
                <Button fx:id="simulate" mnemonicParsing="false"
onAction="#handleSimulateButtonPressed"
                    prefHeight="25.0" prefWidth="120.0" text="Run Simulation">
                  <font>
                    <Font size="14.0"/>
                  </font>
                </Button>
                <Button fx:id="logout" mnemonicParsing="false"
onAction="#handleLogoutButtonPressed"
                    prefHeight="25.0" prefWidth="120.0" text="Logout">
                  <font>
                    <Font size="14.0"/>
                  </font>
                </Button>
              </children>
            </HBox>
            <PieChart fx:id="valueChart" title="Portfolio"
GridPane.columnIndex="1" GridPane.rowSpan="2"/>
            <Label fx:id="valueLabel" alignment="CENTER" prefHeight="17.0"
prefWidth="255.0"
                text="Portfolio Current Value: " GridPane.halignment="CENTER"
GridPane.rowIndex="1"/>
          </children>
        </GridPane>
    </children>
```

```
</AnchorPane>

package controller;

import gui.FPTS;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Node;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.PasswordField;
import javafx.scene.control.TextField;
import javafx.stage.Stage;
import model.User;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.net.URL;
import java.util.ResourceBundle;

/**
 * This class controls the actions from button presses on the Login and Register
page of this application.
 * Extends LoginMenuController because that class holds the code to handle
events based on clicking items in the Menu.
 * Created by Luke Veilleux on 3/01/2016.
 */
public class LoginController extends LoginMenuController {
    /**
     * Private variables representing the input of the user ID and password on both
login and registration as well
     * as a label that is used to output error messages to the user.
     */
    @FXML
    private Label error;
    @FXML
    private PasswordField password;
    @FXML
    private TextField userid;
    @FXML
    private PasswordField password1;
    FPTS fpts = FPTS.getSelf();
```

```
    /**
     * Controls the program when the Login button is pressed on the Login page of
the application. Validates the user
     * entered correct credentials, and if so logs them in, otherwise one of a few
different errors will appear.
     *
     * @param event - ActionEvent - event that caused this method to be called.
     * @throws IOException - Exception thrown if the HomePage.fxml is not found
where it should be.
     */
    @FXML
    protected void handleLoginButtonPressed(ActionEvent event) throws
IOException {
        if (userid.getText().length() != 0 && password.getText().length() != 0) {
            User u = new User(userid.getText(), password.getText());
            if (u.validateUser()) {
                fpts.setCurrentUser(u);
                error.setText("Logging in...");
                Parent parent = FXMLLoader.load(getClass().getResource("../gui/
HomePage.fxml"));
                Scene scene = new Scene(parent);
                Stage app_stage = (Stage) ((Node)
event.getSource()).getScene().getWindow();

                app_stage.setScene(scene);
                app_stage.show();
            } else {
                password.clear();
                error.setText("Not a valid combination of login ID and password");
            }
        } else {
            error.setText("You have missing fields.");
        }
    }

    /**
     * Controls the programs actions if the register button is pressed on the
Registration page.
     * The user id is checked to ensure it is not already in use in the system, and
the registers a new user.
     * A few different error messages are displayed based on different criteria not
being met.
     *
     * @param event - ActionEvent - event that caused this method to be called.
     * @throws IOException - Exception thrown if the HomePage.fxml is not found
where it should be.
     */
```

```
    @FXML
    public void handleRegistrationButtonPressed(ActionEvent event) throws
IOException {
        if (userid.getText().length() != 0 && password.getText().length() != 0) {
            if (User.ValidLoginID(userid.getText())) {
                if (password.getText().equals(password1.getText())) {
                    User usr = new User(userid.getText(), password.getText());
                    fpts.setCurrentUser(usr);
                    addUser(usr);
                    Parent parent = FXMLLoader.load(getClass().getResource("../gui/
HomePage.fxml"));
                    Scene scene = new Scene(parent);
                    Stage stage = (Stage) ((Node)
event.getSource()).getScene().getWindow();
                    stage.setScene(scene);
                    stage.show();

                } else {
                    error.setText("Password fields do not match. Try your password
again.");
                    password.clear();
                    password1.clear();
                }
            } else {
                error.setText("That User ID is already in use, please pick another
one.");
            }
        } else {
            error.setText("Please Enter both a UserID and a Password");
        }
    }

    /**
     * Controls the program if the clear button is pressed on either of the Login or
Registration pages.
     * Clears the text entered in the userid and password fields.
     *
     * @param event - ActionEvent - event that caused this method to be called.
     */
    @FXML
    protected void handleClearButtonPressed(ActionEvent event) {
        userid.clear();
        password.clear();
        if (password1 != null) password1.clear();
    }

    /**
```

```
    * Controls the program when the back button is clicked on the Registration
page.
    *
    * @param event - ActionEvent - event that caused this method to be called.
    * @throws IOException - Exception thrown if the LoginPage.fxml is not found
where the program expects.
    */
   @FXML
   protected void handleBackButtonPressed(ActionEvent event) throws
IOException {
       goToLoginPage(event);
   }

   /**
    * Controls the program when the register button is clicked on the Login page.
    *
    * @param event - ActionEvent - event that caused this method to be called.
    * @throws IOException - Exception thrown if the RegisterPage.fxml is not
found where the program expects.
    */
   @FXML
   protected void handleRegisterButtonPressed(ActionEvent event) throws
IOException {
       Parent register_parent = FXMLLoader.load(getClass().getResource("../gui/
RegisterPage.fxml"));
       Scene register_scene = new Scene(register_parent);
       Stage app_stage = (Stage) ((Node)
event.getSource()).getScene().getWindow();

       app_stage.setScene(register_scene);
       app_stage.show();
   }

   /**
    * Method used to handle the when the Logout Save and Exit button is clicked
on the Logout window.
    *
    * @param event - ActionEvent - The event that caused this action to occur.
    * @throws IOException - Throws IO exception if the the LoginPage.fxml file is
not in the gui folder.
    */
   @FXML
   protected void handleSaveExitButtonPressed(ActionEvent event) throws
IOException {
       Stage stg = FPTS.getSelf().getStage();
       // TODO Save portfolio.
       stg.setScene(FPTS.getSelf().createLogInScene());
```

```
        Stage stage = (Stage) ((Node) event.getSource()).getScene().getWindow();
        stage.close();
    }

    /**
     * Method used to handle the when the Logout Exit without Saving button is
clicked on the Logout window.
     *
     * @param event - ActionEvent - The event that caused this action to occur.
     * @throws IOException - Throws IO exception if the the LoginPage.fxml file is
not in the gui folder.
     */
    @FXML
    protected void handleExitExitButtonPressed(ActionEvent event) throws
IOException {
        Stage stg = FPTS.getSelf().getStage();
        stg.setScene(FPTS.getSelf().createLogInScene());
        stg.show();
        Stage stage = (Stage) ((Node) event.getSource()).getScene().getWindow();
        stage.close();
    }

    /**
     * Method used to initialize items on the Login page. Currently not in use.
     */
    @Override
    public void initialize(URL location, ResourceBundle resources) {
        //TODO
    }

    /**
     * Adds a new user into the UserData.csv file for use in later uses of this
application.
     *
     * @param usr - User - New user object to be added into the text file.
     */
    private void addUser(User usr) {
        User.addToList(usr);
        FileWriter fileWriter = null;
        BufferedWriter bufferedWriter = null;
        try {
            fileWriter = new FileWriter(new File("JavaFXApp/src/model/Database/
UserData.csv").getAbsolutePath(), true);
            bufferedWriter = new BufferedWriter(fileWriter);
            bufferedWriter.write(usr.getLoginID() + ",");
            bufferedWriter.write(usr.hash(password1.getText()));
            bufferedWriter.newLine();
```

```
            bufferedWriter.close();
        } catch (IOException e1) {
            e1.printStackTrace();
        }
    }
}
package controller;

import javafx.application.Platform;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.control.MenuBar;
import javafx.stage.Stage;

import java.io.IOException;

/**
 * This class is the base implementation of the Menu Bar used in this application.
 * Holds code to go to the LoginPage, Exit the Application, and Open the About
page.
 * Created by Luke Veilleux on 3/11/2016.
 */
public abstract class LoginMenuController implements Initializable {
    /**
     * A Local variable to access the MenuBar located in the FXML documents for
this application.
     */
    @FXML
    MenuBar myMenuBar;

    /**
     * Handler for when the Logout button is pressed in the Menu Bar
     *
     * @param event - ActionEvent - Event that caused this function to be called.
     * @throws IOException - Throws IO Exception if the LoginPage.fxml is not
found by the program.
     */
    public void handleLogoutMenuItemPressed(ActionEvent event) throws
IOException {
        goToLoginPage(event);
    }

    /**
```

```
     * Handler for when the Exit button is pressed in the Menu Bar
     *
     * @param event - ActionEvent - Event that caused this function to be called.
     */
    public void handleExitMenuItemPressed(ActionEvent event) {
        Platform.exit();
        System.exit(0);
    }

    /**
     * Handler for when the About button is pressed in the Menu Bar
     *
     * @param event - ActionEvent - Event that caused this function to be called.
     */
    public void handleAboutMenuItemPressed(ActionEvent event) {

    }

    /**
     * Additional function used in this application to return the application to the
Login Page.
     *
     * @param event - ActionEvent - Event that caused the super function to be
called, used to get the current Stage.
     * @throws IOException - Throws IO Exception if the LoginPage.fxml cannot
be found.
     */
    public void goToLoginPage(ActionEvent event) throws IOException {
        Stage stage;
        try {
            stage = (Stage) ((Node) event.getSource()).getScene().getWindow();
        } catch (ClassCastException c) {
            stage = (Stage) myMenuBar.getScene().getWindow();

        }
        Scene scene = new Scene(FXMLLoader.load(getClass().getResource("../
gui/LoginPage.fxml")));
        stage.setScene(scene);
        stage.show();
    }

}

<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>
```

```
<?import javafx.scene.text.Font?>
<AnchorPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity"
minWidth="-Infinity" prefHeight="600.0"
          prefWidth="800.0" xmlns="http://javafx.com/javafx/8" xmlns:fx="http://
javafx.com/fxml/1"
          fx:controller="controller.LoginController">
   <children>
      <MenuBar fx:id="myMenuBar" layoutY="2.0" minHeight="-Infinity"
minWidth="-Infinity" prefHeight="25.0"
               prefWidth="800.0" AnchorPane.topAnchor="2.0">
         <menus>
            <Menu mnemonicParsing="false" text="File">
               <items>
                  <MenuItem fx:id="logout" mnemonicParsing="false"
onAction="#handleLogoutMenuItemPressed"
                            text="Logout"/>
                  <MenuItem fx:id="exit" mnemonicParsing="false"
onAction="#handleExitMenuItemPressed"
                            text="Exit"/>
               </items>
            </Menu>
            <Menu mnemonicParsing="false" text="Help">
               <items>
                  <MenuItem fx:id="about" mnemonicParsing="false"
onAction="#handleAboutMenuItemPressed"
                            text="About"/>
               </items>
            </Menu>
         </menus>
      </MenuBar>
      <GridPane layoutX="-3.0" layoutY="29.0" prefHeight="573.0"
prefWidth="805.0">
         <columnConstraints>
            <ColumnConstraints hgrow="SOMETIMES" maxWidth="476.0"
minWidth="10.0" prefWidth="456.0"/>
            <ColumnConstraints hgrow="SOMETIMES" maxWidth="349.0"
minWidth="10.0" prefWidth="349.0"/>
         </columnConstraints>
         <rowConstraints>
            <RowConstraints maxHeight="306.00001525878906"
minHeight="10.0" prefHeight="201.0" vgrow="SOMETIMES"/>
            <RowConstraints maxHeight="285.0" minHeight="10.0"
prefHeight="251.66665649414062" vgrow="SOMETIMES"/>
            <RowConstraints maxHeight="131.00003051757812"
minHeight="10.0" prefHeight="116.33334350585938"
                            vgrow="SOMETIMES"/>
         </rowConstraints>
```

```
        <children>
            <Label alignment="CENTER" prefHeight="27.0" prefWidth="344.0"
text="Financial Portfolio Tracking System"
                   GridPane.halignment="CENTER"
GridPane.valignment="CENTER">
                <font>
                    <Font name="System Bold Italic" size="18.0"/>
                </font>
            </Label>
            <GridPane prefHeight="135.0" prefWidth="324.0"
GridPane.rowIndex="1">
                <columnConstraints>
                    <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0"
prefWidth="100.0"/>
                    <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0"
prefWidth="100.0"/>
                </columnConstraints>
                <rowConstraints>
                    <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES"/>
                    <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES"/>
                </rowConstraints>
                <children>
                    <Label prefHeight="20.0" prefWidth="76.0" text="Username: "
GridPane.halignment="CENTER"
                           GridPane.valignment="CENTER">
                        <font>
                            <Font size="14.0"/>
                        </font>
                    </Label>
                    <Label prefHeight="20.0" prefWidth="69.0" text="Password:"
GridPane.halignment="CENTER"
                           GridPane.rowIndex="1">
                        <font>
                            <Font size="14.0"/>
                        </font>
                    </Label>
                    <PasswordField fx:id="password" promptText="Password"
GridPane.columnIndex="1"
                                   GridPane.rowIndex="1"/>
                    <TextField fx:id="userid" prefHeight="25.0" prefWidth="150.0"
promptText="User ID"
                               GridPane.columnIndex="1"/>
                </children>
            </GridPane>
            <Label fx:id="error" mouseTransparent="true" prefHeight="17.0"
```

```
prefWidth="404.0" GridPane.columnSpan="2"
                  GridPane.halignment="CENTER" GridPane.rowIndex="2"/>
            <GridPane prefHeight="268.0" prefWidth="332.0"
GridPane.columnIndex="1" GridPane.rowIndex="1">
               <columnConstraints>
                  <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0"
prefWidth="100.0"/>
               </columnConstraints>
               <rowConstraints>
                  <RowConstraints maxHeight="51.0" minHeight="10.0"
prefHeight="46.0" vgrow="SOMETIMES"/>
                  <RowConstraints maxHeight="107.0" minHeight="10.0"
prefHeight="44.0" vgrow="SOMETIMES"/>
                  <RowConstraints maxHeight="107.0" minHeight="10.0"
prefHeight="50.0" vgrow="SOMETIMES"/>
               </rowConstraints>
               <children>
                  <Button id="login" defaultButton="true" mnemonicParsing="false"
                        onAction="#handleLoginButtonPressed" prefHeight="25.0"
prefWidth="91.0" text="Login"
                        GridPane.halignment="CENTER"/>
                  <Button id="clear" mnemonicParsing="false"
onAction="#handleClearButtonPressed"
                        prefHeight="25.0" prefWidth="91.0" text="Clear"
GridPane.halignment="CENTER"
                        GridPane.rowIndex="1"/>
                  <Button id="register" mnemonicParsing="false"
onAction="#handleRegisterButtonPressed"
                        prefHeight="25.0" prefWidth="91.0" text="Register"
GridPane.halignment="CENTER"
                        GridPane.rowIndex="2"/>
               </children>
            </GridPane>
            <Label text="Login" GridPane.columnIndex="1"
GridPane.halignment="CENTER">
               <font>
                  <Font name="System Bold Italic" size="18.0"/>
               </font>
            </Label>
         </children>
      </GridPane>
   </children>
</AnchorPane>

<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Button?>
```

```
<?import javafx.scene.control.Label?>
<?import javafx.scene.layout.*?>
<?import javafx.scene.text.Font?>
<AnchorPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity"
minWidth="-Infinity" prefHeight="300.0"
          prefWidth="400.0" xmlns="http://javafx.com/javafx/8" xmlns:fx="http://
javafx.com/fxml/1"
          fx:controller="controller.LoginController">
   <children>
      <GridPane layoutX="1.0" prefHeight="300.0" prefWidth="400.0">
         <columnConstraints>
            <ColumnConstraints hgrow="SOMETIMES" maxWidth="228.0"
minWidth="10.0" prefWidth="200.0"/>
            <ColumnConstraints hgrow="SOMETIMES" maxWidth="200.0"
minWidth="10.0" prefWidth="200.0"/>
         </columnConstraints>
         <rowConstraints>
            <RowConstraints maxHeight="178.0" minHeight="10.0"
prefHeight="170.0" vgrow="SOMETIMES"/>
            <RowConstraints maxHeight="145.0" minHeight="10.0"
prefHeight="130.0" vgrow="SOMETIMES"/>
         </rowConstraints>
         <children>
            <Label alignment="CENTER" contentDisplay="CENTER"
prefHeight="85.0" prefWidth="400.0"
                   text="Are you sure you want to logout?    Any unsaved changes
will not persist to when you next login."
                   textAlignment="CENTER" wrapText="true"
GridPane.columnSpan="2" GridPane.halignment="CENTER">
               <font>
                  <Font name="System Bold" size="14.0"/>
               </font>
            </Label>
            <Button fx:id="saveEXIT" mnemonicParsing="false"
onAction="#handleSaveExitButtonPressed"
                    prefHeight="25.0" prefWidth="140.0" text="Save"
GridPane.halignment="CENTER"
                    GridPane.rowIndex="1">
               <font>
                  <Font name="System Italic" size="13.0"/>
               </font>
            </Button>
            <Button fx:id="exitEXIT" mnemonicParsing="false"
onAction="#handleExitExitButtonPressed"
                    prefHeight="25.0" prefWidth="140.0" text="Don't Save"
GridPane.columnIndex="1"
                    GridPane.halignment="CENTER" GridPane.rowIndex="1">
```

```
                <font>
                    <Font name="System Italic" size="13.0"/>
                </font>
            </Button>
        </children>
    </GridPane>
  </children>
</AnchorPane>
```

```
package controller;

import gui.FPTS;
import javafx.application.Platform;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;
import javafx.scene.Node;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.MenuBar;
import javafx.stage.Stage;
import model.DataBase.WriteFile;

import java.io.IOException;

/**
 * This class is the base implementation of the Menu Bar used in this application.
 * Other Controllers will extend this class to gain functionality of the MenuBar in
the application.
 * Created by Luke on 3/01/2016.
 */
public abstract class MenuController implements Initializable {
    /**
     * A Local variable to access the MenuBar located in the FXML documents for
this FPTS application.
     */
    @FXML
    MenuBar myMenuBar;
    FPTS fpts = FPTS.getSelf();

    /**
     * Handler for when the Logout button is pressed in the Menu Bar
     *
     * @param event - ActionEvent - Event that caused this function to be called.
     * @throws IOException - Throws IO Exception if the LoginPage.fxml is not
found by the program.
```

```
     */
    public void handleLogoutMenuItemPressed(ActionEvent event) throws
IOException {
        Stage stage = new Stage();
        Parent parent = FXMLLoader.load(getClass().getResource("../gui/
LogoutPage.fxml"));
        Scene scene = new Scene(parent);
        stage.setScene(scene);
        stage.show();
    }

    /**
     * Handler for when the Exit button is pressed in the Menu Bar
     *
     * @param event - ActionEvent - Event that caused this function to be called.
     */
    public void handleExitMenuItemPressed(ActionEvent event) {
        Platform.exit();
        System.exit(0);
    }

    public void handleSaveMenuItemPressed(ActionEvent event) {
        WriteFile writeFile = new WriteFile();
        writeFile.updatePortfolioForUser(fpts.getCurrentUser());
    }

    /**
     * Handler for when the About button is pressed in the Menu Bar
     *
     * @param event - ActionEvent - Event that caused this function to be called.
     */
    public void handleAboutMenuItemPressed(ActionEvent event) {
        //TODO
    }

    public void handleHomeMenuItemPressed(ActionEvent event) throws
IOException {
        Parent parent = FXMLLoader.load(getClass().getResource("../gui/
HomePage.fxml"));
        Scene scene = new Scene(parent);
        Stage stage = (Stage) myMenuBar.getScene().getWindow();
        stage.setScene(scene);
        stage.show();
    }

    public void handlePortfolioMenuItemPressed(ActionEvent event) throws
IOException {
```

```
        Displayer pd = new PortfolioDisplayer();
        pd.display(FPTS.getSelf());
    }


    public void handleBuyEquitiesMenuItemPressed(ActionEvent event) {
        HoldingAlgorithm eqUpdater = new BuyHoldingAlgorithm();
        eqUpdater.process(FPTS.getSelf());
    }

    public void handleSellEquitiesMenuItemPressed(ActionEvent event) {
        HoldingAlgorithm eqUpdater = new SellHoldingAlgorithm();
        eqUpdater.process(FPTS.getSelf());
    }

    public void handleWithdrawMenuItemPressed(ActionEvent event) {
        CashAccountAlgorithm cashAcctAlgor = new
WithdrawCashAccountAlgorithm();
        cashAcctAlgor.process(FPTS.getSelf());
    }

    public void handleDepositMenuItemPressed(ActionEvent event) {
        CashAccountAlgorithm cashAcctAlgor = new
DepositCashAccountAlgorithm();
        cashAcctAlgor.process(FPTS.getSelf());
    }

    public void handleCreateMenuItemPressed(ActionEvent event) {
        CashAccountCreator cashAcctAlgor = new
CashAccountCreator(FPTS.getSelf());
        cashAcctAlgor.getCashAccountCreatorScene();
    }

    public void handleTransferMenuItemPressed(ActionEvent event) {
        CashAccountAlgorithm cashAcctAlgor = new
TransferCashAccountAlgorithm();
        cashAcctAlgor.process(FPTS.getSelf());
    }

    public void handleRemoveMenuItemPressed(ActionEvent event) {
        CashAccountAlgorithm cashAcctAlgor = new
RemoveCashAccountAlgorithm();
        cashAcctAlgor.process(FPTS.getSelf());
    }

    /**
     * Additional function used in this application to return the application to the
```

```
Login Page.
     *
     * @param event - ActionEvent - Event that caused the super function to be
called, used to get the current Stage.
     * @throws IOException - Throws IO Exception if the LoginPage.fxml cannot
be found.
     */
    public void goToLoginPage(ActionEvent event) throws IOException {
        Parent parent = FXMLLoader.load(getClass().getResource("../gui/
LoginPage.fxml"));
        Stage stage = new Stage();
        try {
            stage = (Stage) ((Node) event.getSource()).getScene().getWindow();
        } catch (ClassCastException c) {
            stage = (Stage) myMenuBar.getScene().getWindow();

        }
        Scene scene = new Scene(parent);
        stage.setScene(scene);
        stage.show();
    }

}

package model;

import gui.FPTS;

import java.util.ArrayList;

/**
 * Created by Brockway on 3/12/16.
 */
public class NoGrowthSimulator implements Simulator {
    public static String name = "No Growth Simulator";

    private ArrayList<Holding> holdings;
    private String interval;
    private boolean hasSteps;
    private int numSteps;
    private int stepNumber;

    /**
     *
     */
    public NoGrowthSimulator(int numSteps, String interval, boolean hasSteps) {
        this.interval = interval;
```

```
        this.hasSteps = hasSteps;
        this.numSteps = numSteps;
        this.holdings = FPTS.getSelf().getPortfolio().getHoldings();
    }

    //TODO: CHECK IF IT HAS STEPS.

    /**
     * @return
     */
    public double simulate(int numberOfSteps) {
        stepNumber += numberOfSteps;
        return 0;
    }

    @Override
    public int getCurrentStep() {
        return stepNumber;
    }

    @Override
    public int getTotalSteps() {
        return numSteps;
    }
}

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package controller;

import gui.FPTS;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.layout.VBox;
import model.Searchable;

import java.util.ArrayList;

/**
 * Displays portfolio elements like Holding and Cash Account in one Scene.
 *
 * @author ericepstein
 */
public class PortfolioDisplayer implements Displayer {
```

```
    /*
     * context data
     */
    FPTS theFPTS;
    ArrayList<Searchable> portfolioElements;

    /**
     * Overrides the Displayer's display method to display specifically
     * the portfolio elements.
     *
     * @param theFPTS
     */
    @Override
    public void display(FPTS theFPTS) {
        this.theFPTS = theFPTS;
        portfolioElements = theFPTS.getPortfolio().getPortfolioElements();
        theFPTS.getStage().setScene(getDisplayScene());
    }

    /**
     * Helper method to construct Scene of display given the elements.
     *
     * @return Scene
     */
    private Scene getDisplayScene() {

        VBox split = new VBox();
        VBox display = new VBox();

        /*
         * Adds each portfolio element in a list of display.
         */
        for (Searchable s : portfolioElements) {
            display.getChildren().add(new Label(s.toString()));
        }
        split.getChildren().addAll(theFPTS.getNav(), display);
        Scene displayScene = new Scene(split, theFPTS.getWidth(),
theFPTS.getHeight());
        return displayScene;
    }
}

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
```

```
 */
package model;

import javafx.scene.control.TextField;
import java.text.ParseException;
import java.util.List;
import java.util.ArrayList;
import java.util.Date;
import java.util.Observable;
import javafx.collections.ObservableList;
import javafx.scene.control.Button;
import javafx.scene.layout.VBox;
import javafx.scene.Node;
import javafx.scene.control.ComboBox;
import javafx.scene.control.TextField;
import javafx.scene.layout.Pane;
import model.DataBase.ReadFile;

import static model.DataBase.ReadFile.readCash;
import static model.DataBase.ReadFile.readHoldings;

/**
 * Maintains multiple holdings in equities, and cash in one or more cash
 * accounts for the user, and maintains a history of all transactions.
 *
 * @author Eric Epstein and Kaitlyn Brockway
 */
public class Portfolio {

    private ArrayList<Searchable> portfolioElements;
    private ArrayList<CashAccount> cashAccounts;
    private ArrayList<Holding> holdings;
    private ArrayList<Transaction> transactions;
    private ArrayList<EquityComponent> equityComponents =
Equity.getEquityList();  // lists what you can buy
    private double currentValue;

    /**
     * When creating a new portfolio, the system shall allow the user to
     * import holdings and transactions to initialize the new portfolio. THIS IS NOT
ALLOWED YET
     *
     * Method called when a user is constructed.
     *
     * @author ericepstein & Kaitlin
     */
    public Portfolio() {
```

```
    portfolioElements = new ArrayList<Searchable>();
    equityComponents = Equity.getEquityList();
    cashAccounts = readCash(); //new ArrayList<CashAccount>(); <--replaced
    holdings = readHoldings(); //new ArrayList<Holding>(); <--replaced
    transactions = new ArrayList<Transaction>();

}

 /**
 * Returns Holding that matches given name
 *
 * @param keyword - String
 * @return Holding
 */
public Holding getHolding(String keyword) {
    for (Holding e : holdings) {
        if (e.getTickerSymbol().equals(keyword)) {
            return e;
        }
    }
    return null;
}


/**
 * Returns a collection of Holding objects that are cast to Searchable
 *
 * @return ArrayList<Searchable>
 */

public ArrayList<Searchable> getHoldingSearchables() {
    ArrayList<Searchable> temp = new ArrayList<Searchable>();
    for (Holding h : holdings) {
        temp.add((Searchable) h);
    }
    return temp;
}

/**
 * Returns collection of EquityComponent objects that are cast to Searchable
 *
 * @return ArrayList<Searchable>
 */
public ArrayList<Searchable> getEquityComponentSearchables() {
    ArrayList<Searchable> temp = new ArrayList<Searchable>();
    for (EquityComponent ec : equityComponents) {
```

```
        temp.add((Searchable) ec);
    }
    return temp;
}

/**
 * Returns collection of CashAccount objects that are cast to Searchable
 *
 * @return ArrayList<Searchable>
 */
public ArrayList<Searchable> getCashAccountSearchables() {
    ArrayList<Searchable> temp = new ArrayList<Searchable>();
    for (CashAccount c : cashAccounts) {
        temp.add((Searchable) c);
    }
    return temp;
}

/**
 * Returns collection of Transaction objects
 *
 * @return ArrayList<Transaction>
 */
public ArrayList<Transaction> getTransactions() {
    return transactions;
}

/**
*
* Returns collection of portfolio elements
*
* @return ArrayList<Searchable>
*/
public ArrayList<Searchable> getPortfolioElements() {
    return portfolioElements;
}

/**
 * Returns collection of CashAccount
 *
 * @return ArrayList<CashAccount>
 */
public ArrayList<CashAccount> getCashAccounts() {
    return cashAccounts;
}

/**
```

```
 * Returns collection of Holding
 *
 * @return ArrayList<Holding>
 */
public ArrayList<Holding> getHoldings() {
    return holdings;
}

/**
 * Returns collection of EquityComponent
 *
 * @return ArrayList<EquityComponent>
 */
public ArrayList<EquityComponent> getEquityComponents() {
    return Equity.getEquityList();
}

/**
 * Adds EquityComponent object to portfolio
 *
 * @param e
 */
public void add(EquityComponent e) {
    equityComponents.add(e);
}

/**
 * Removes EquityCompoonent object from portfolio
 *
 * @param e
 */
public void remove(EquityComponent e) {
    equityComponents.remove(e);
}

/**
 * Adds CashAccount to portfolio
 *
 * @param e - CashAccount
 */
public void add(CashAccount e) {
    portfolioElements.add((Searchable) e);
    transactions.add((Transaction) new Deposit(e, e.getValue()));
    cashAccounts.add(e);
}

/**
```

```
 * Removes CashAccount from portfolio
 *
 * @param e - CashAccount
 */
public void remove(CashAccount e) {
    portfolioElements.remove((Searchable) e);
    cashAccounts.remove(e);
}

/**
 * Executes Transaction and adds to portfolio history
 *
 * @param t - Transaction
 */
public void add(Transaction t) {
    transactions.add(t);
    t.execute();
}

/**
 * Removes Transaction from history list
 *
 * @param t
 */
public void remove(Transaction t) {
    transactions.remove(t);
}


/**
 * Adds Holding to portfolio
 *
 * @param e - Holding
 */
public void add(Holding e) {
    portfolioElements.add((Searchable) e);
    holdings.add(e);
}

/**
 * Removes Holding from portfolio
 *
 * @param e - Holding
 */
public void remove(Holding e) {
    portfolioElements.remove((Searchable) e);
    holdings.remove(e);
```

```
    }

    public CashAccount getCashAccount(CashAccount c) {
        for (CashAccount aC : cashAccounts) {
            if (aC.equals(c)) {
                return aC;
            }
        }
        return null;
    }

}

package model.DataBase;

import model.CashAccount;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;

/**
 * Created by iLondon on 3/13/16.
 */
public class ReadCash {
    protected static ArrayList<String[]> readInFile() {
        return ReadFile.readInUser("/Cash.csv");
    }

    public static ArrayList<CashAccount> allCash;
    private static ArrayList<String[]> splitFile;

    /**
     * Created by Ian
     *
     * @return list of all Holding objects
     */
    protected static ArrayList<CashAccount> read() {
        splitFile = readInFile();
        allCash = new ArrayList<>();

        // iterate through each line representing an Holding
        for (String[] line : splitFile) {
            Date date = null;
            try {
                date = new SimpleDateFormat("EEE MMM dd HH:mm:ss zzz
```

```
yyyy").parse(line[2]);
            } catch (ParseException e) {
                e.printStackTrace();
            }
            // create cash account object form line
            CashAccount curCash = new CashAccount(line[0],
Double.parseDouble(line[1]), date);

            // add cash accounts iteratively
            allCash.add(curCash);
        }
        return allCash;
    }
}

package model.DataBase;

import model.Equity;
import model.EquityComponent;
import model.EquityComposite;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

/**
 * Created by Ian on 3/2/16.
 */
public class ReadEquity {

    protected static ArrayList<String[]> readInFile() {
        return ReadFile.readInFile();
    }

    private static List<String> indexList = new
ArrayList<String>(Arrays.asList("DOW", "NASDAQ100"));
    private static List<String> sectorList = new
ArrayList<String>(Arrays.asList("FINANCE", "TECHNOLOGY", "HEALTH
CARE", "TRANSPORTATION"));
    public static ArrayList<EquityComponent> allEquities = new ArrayList<>();
    private static ArrayList<String[]> splitFile = new ArrayList<String[]>();

    /**
     * Created by Ian
     *
     * @return list of all Equity objects
     */
```

```
    public static ArrayList<EquityComponent> read() {
        splitFile = readInFile();
        ArrayList<EquityComponent> allEquities = new ArrayList<>();
        ArrayList<EquityComposite> CompositeEquities = loadCompositeList();
        allEquities.addAll(CompositeEquities);

        // iterate through each line representing an equity
        for (String[] line : splitFile) {
            ArrayList<String> indices = new ArrayList<String>();
            ArrayList<String> sectors = new ArrayList<String>();
            Equity curEquity = new Equity(line[0], line[1],
Double.parseDouble(line[2]), indices, sectors);
            // iterate through fields of current equity
            for (int i = 3; i < line.length; i++) {
                // finance, technology, health care, transportation
                if (sectorList.contains(line[i])) {
                    sectors.add(line[i]);
                    // add to sector composite
                    try {
                        for (EquityComposite ec : CompositeEquities) {
                            if (ec.getEquityType().equals("Sector") &
ec.getEquityName().equals(line[i])) {
                                ec.add((EquityComponent) curEquity);
                            }
                        }
                    } catch (Exception e) {
                        System.out.println("sector composite object not found! Please try
again.");
                    }
                }
                // dow, nasdaq100
                else if (indexList.contains(line[i])) {
                    indices.add(line[i]);
                    //add to index composite
                    try {
                        for (EquityComposite ec : CompositeEquities) {
                            if (ec.getEquityType().equals("Index") &
ec.getEquityName().equals(line[i])) {
                                ec.add((EquityComponent) curEquity);
                            }
                        }
                    } catch (Exception e) {
                        System.out.println("index composite object not found! Please try
again.");
                    }
                }
            }
```

```
            allEquities.add(curEquity);
        }
        return allEquities;
    }

    /**
     * Created by Ian
     *
     * @return list of composite Equities
     */
    private static ArrayList<EquityComposite> loadCompositeList() {
        ArrayList<EquityComposite> compositeList = new
ArrayList<EquityComposite>();
        // create the bare index composites
        for (String index : indexList) {
            ArrayList<String> indices = new ArrayList<String>();
            ArrayList<String> sectors = new ArrayList<String>();
            compositeList.add(new EquityComposite(index, "Index"));
        }
        // create the bare index composites
        for (String sector : sectorList) {
            ArrayList<String> indices = new ArrayList<String>();
            ArrayList<String> sectors = new ArrayList<String>();
            compositeList.add(new EquityComposite(sector, "Sector"));
        }
        return compositeList;
    }
}

package model.DataBase;

import gui.FPTS;
import model.CashAccount;
import model.EquityComponent;
import model.Holding;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;

/**
 * @author: Ian London
 */
public class ReadFile {
```

```
    protected static ArrayList<String[]> splitFile = new ArrayList<String[]>();

    public static ArrayList<EquityComponent> readEquity() {
        return ReadEquity.read();
    }

    public static ArrayList<Holding> readHoldings() {
        return ReadHoldings.read();
    }

    public static ArrayList<CashAccount> readCash() {
        return ReadCash.read();
    }


    // reads in CSV file
    protected static ArrayList<String[]> readInFile() {
        String csv = "JavaFXApp/src/model/DataBase/equities.csv";
        splitFile = new ArrayList<String[]>();
        BufferedReader reader = null;
        String line;

        try {
            reader = new BufferedReader(new FileReader(csv));
            while ((line = reader.readLine()) != null) {
                line = line.substring(1, line.length() - 1);
                String[] split = line.split("\",\"");
                splitFile.add(split);
            }
        } catch (FileNotFoundException e) {
            System.out.println("JavaFXApp/src/model/DataBase/equities.csv not
found! Please try again.");
            //readInFile();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            if (reader != null) {
                try {
                    reader.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
        return splitFile;
    }
```

```
    // reads in CSV file
    protected static ArrayList<String[]> readInUser(String file) {
        //System.out.println("CURRENT USER ID: " + FPTS.getCurrentUserID());
        String csv = "JavaFXApp/src/model/DataBase/Portfolios/" +
FPTS.getCurrentUserID() + file;
        splitFile = new ArrayList<String[]>();
        BufferedReader reader = null;
        String line;

        try {
            reader = new BufferedReader(new FileReader(csv));
            while ((line = reader.readLine()) != null) {
                line = line.substring(1, line.length() - 1);
                String[] split = line.split("\",\"");
                splitFile.add(split);
            }
        } catch (FileNotFoundException e) {
            System.out.println("JavaFXApp/src/model/DataBase/equities.csv not
found! Please try again.");
            //readInFile();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            if (reader != null) {
                try {
                    reader.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
        return splitFile;
    }
}

package model.DataBase;

import model.Holding;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Date;
import java.util.List;

/**
```

```
 * Created by iLondon on 3/2/16.
 */
public class ReadHoldings {

    protected static ArrayList<String[]> readInFile() {
        return ReadFile.readInUser("/Holdings.csv");
    }

    private static List<String> indexList = new
ArrayList<String>(Arrays.asList("DOW", "NASDAQ100"));
    private static List<String> sectorList = new
ArrayList<String>(Arrays.asList("FINANCE", "TECHNOLOGY", "HEALTH
CARE", "TRANSPORTATION"));
    public static ArrayList<Holding> allHoldings;
    private static ArrayList<String[]> splitFile;

    /**
     * Created by Ian
     *
     * @return list of all Holding objects
     */
    public static ArrayList<Holding> read() {
        splitFile = readInFile();
        allHoldings = new ArrayList<>();

        // iterate through each line representing an Holding
        for (String[] line : splitFile) {
            ArrayList<String> indices = new ArrayList<String>();
            ArrayList<String> sectors = new ArrayList<String>();
            Date date = null;
            try {
                date = new SimpleDateFormat("EEE MMM dd HH:mm:ss zzz
yyyy").parse(line[4]);
            } catch (ParseException e) {
                e.printStackTrace();
            }

            Holding curHolding = new Holding(line[0], line[1],
Double.parseDouble(line[2]), Integer.parseInt(line[3]), date, indices, sectors);
            // iterate through fields of current Holding
            for (int i = 5; i < line.length; i++) {
                // finance, technology, health care, transportation
                if (sectorList.contains(line[i])) {
                    sectors.add(line[i]);
                }
                // dow, nasdaq100
                else if (indexList.contains(line[i])) {
```

```
                indices.add(line[i]);
            }
        }
        allHoldings.add(curHolding);
    }
    return allHoldings;
  }
}

package controller;


/**
 * Implements final step in CashAccountAlgorithm that removes specified
CashAccount.
 *
 * @author Eric Epstein
 */
public class RemoveCashAccountAlgorithm extends CashAccountAlgorithm {

  /**
   * Overrides CashAccountAlgorithm's action() method to remove specified
   * CashAccount
   */
  @Override
  public void action() {
    theFPTS.getPortfolio().remove(c);
    theFPTS.getStage().setScene(theFPTS.getConfirmationScene());
  }

}

<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>
<?import javafx.scene.text.*?>
<AnchorPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity"
minWidth="-Infinity" prefHeight="600.0"
        prefWidth="800.0" xmlns="http://javafx.com/javafx/8" xmlns:fx="http://
javafx.com/fxml/1"
        fx:controller="controller.LoginController">
  <children>
    <MenuBar fx:id="myMenuBar" layoutY="2.0" minHeight="-Infinity"
minWidth="-Infinity" prefHeight="25.0"
            prefWidth="800.0" AnchorPane.topAnchor="2.0">
        <menus>
```

```
            <Menu mnemonicParsing="false" text="File">
              <items>
                <MenuItem fx:id="Logout" mnemonicParsing="false"
onAction="#handleLogoutMenuItemPressed"
                          text="Logout"/>
                <MenuItem fx:id="Exit" mnemonicParsing="false"
onAction="#handleExitMenuItemPressed"
                          text="Exit"/>
              </items>
            </Menu>
            <Menu mnemonicParsing="false" text="Help">
              <items>
                <MenuItem fx:id="about" mnemonicParsing="false"
onAction="#handleAboutMenuItemPressed"
                          text="About"/>
              </items>
            </Menu>
          </menus>
        </MenuBar>
        <GridPane layoutX="-3.0" layoutY="29.0" prefHeight="573.0"
prefWidth="802.0">
          <columnConstraints>
            <ColumnConstraints hgrow="SOMETIMES" maxWidth="496.0"
minWidth="10.0" prefWidth="413.3333435058594"/>
            <ColumnConstraints hgrow="SOMETIMES" maxWidth="332.0"
minWidth="10.0" prefWidth="310.6666564941406"/>
          </columnConstraints>
          <rowConstraints>
            <RowConstraints maxHeight="317.99998474121094"
minHeight="10.0" prefHeight="196.66665649414062"
                            vgrow="SOMETIMES"/>
            <RowConstraints maxHeight="173.33334350585938"
minHeight="10.0" prefHeight="136.66665649414062"
                            vgrow="SOMETIMES"/>
            <RowConstraints maxHeight="132.33334350585938"
minHeight="10.0" prefHeight="132.33334350585938"
                            vgrow="SOMETIMES"/>
          </rowConstraints>
          <children>
            <Label alignment="CENTER" prefHeight="27.0" prefWidth="350.0"
text="Financial Portfolio Tracking System"
                   GridPane.halignment="CENTER"
GridPane.valignment="CENTER">
              <font>
                <Font name="System Bold Italic" size="18.0"/>
              </font>
            </Label>
```

```
            <GridPane prefHeight="135.0" prefWidth="324.0"
GridPane.rowIndex="1">
                <columnConstraints>
                    <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0"
prefWidth="100.0"/>
                    <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0"
prefWidth="100.0"/>
                </columnConstraints>
                <rowConstraints>
                    <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES"/>
                    <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES"/>
                    <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES"/>
                </rowConstraints>
                <children>
                    <Label prefHeight="20.0" prefWidth="76.0" text="Username: "
GridPane.halignment="CENTER"
                           GridPane.valignment="CENTER">
                        <font>
                            <Font size="14.0"/>
                        </font>
                    </Label>
                    <Label prefHeight="20.0" prefWidth="69.0" text="Password:"
GridPane.halignment="CENTER"
                           GridPane.rowIndex="1">
                        <font>
                            <Font size="14.0"/>
                        </font>
                    </Label>
                    <PasswordField fx:id="password" promptText="Password"
GridPane.columnIndex="1"
                                   GridPane.rowIndex="1"/>
                    <TextField fx:id="userid" prefHeight="25.0" prefWidth="150.0"
promptText="User ID"
                               GridPane.columnIndex="1"/>
                    <Label prefHeight="20.0" prefWidth="117.0" text="Confirm
Password:" GridPane.halignment="CENTER"
                           GridPane.rowIndex="2">
                        <font>
                            <Font size="14.0"/>
                        </font>
                    </Label>
                    <PasswordField fx:id="password1" promptText="Password"
GridPane.columnIndex="1"
                                   GridPane.rowIndex="2"/>
```

```
            </children>
         </GridPane>
         <Label fx:id="error" mouseTransparent="true" prefHeight="17.0"
prefWidth="404.0" GridPane.columnSpan="2"
            GridPane.halignment="CENTER" GridPane.rowIndex="2"/>
         <GridPane prefHeight="268.0" prefWidth="332.0"
GridPane.columnIndex="1" GridPane.rowIndex="1">
            <columnConstraints>
               <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0"
prefWidth="100.0"/>
            </columnConstraints>
            <rowConstraints>
               <RowConstraints maxHeight="51.0" minHeight="10.0"
prefHeight="46.0" vgrow="SOMETIMES"/>
               <RowConstraints maxHeight="107.0" minHeight="10.0"
prefHeight="44.0" vgrow="SOMETIMES"/>
               <RowConstraints maxHeight="107.0" minHeight="10.0"
prefHeight="50.0" vgrow="SOMETIMES"/>
            </rowConstraints>
            <children>
               <Button id="clear" fx:id="clear" mnemonicParsing="false"
onAction="#handleClearButtonPressed"
                  prefHeight="25.0" prefWidth="91.0" text="Clear"
GridPane.halignment="CENTER"
                  GridPane.rowIndex="1"/>
               <Button id="register" fx:id="registration" defaultButton="true"
mnemonicParsing="false"
                  onAction="#handleRegistrationButtonPressed"
prefHeight="25.0" prefWidth="91.0"
                  text="Register" GridPane.halignment="CENTER"/>
               <Button fx:id="back" cancelButton="true"
mnemonicParsing="false"
                  onAction="#handleBackButtonPressed" prefHeight="25.0"
prefWidth="91.0" text="Back"
                  GridPane.halignment="CENTER" GridPane.rowIndex="2"/>
            </children>
         </GridPane>
         <Label text="Registration" GridPane.columnIndex="1"
GridPane.halignment="CENTER">
            <font>
               <Font name="System Bold Italic" size="18.0"/>
            </font>
         </Label>
      </children>
   </GridPane>
</children>
</AnchorPane>
```

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package model;

/**
 * Defines an interface that accesses a display name. Allows objects of
 * different types to be processed in the same algorithm.
 *
 * @author Eric Epstein
 */
public interface Searchable {

    /**
     * Returns a displayable name
     *
     * @return String
     */
    public String getDisplayName();

}

package model;

import java.util.ArrayList;

/**
 * Defines one step in the Searcher that converts the object being
 * searched into a string representation of an Equity or Holding, as both
 * are treated the same under the HoldingUpdatable interface.
 *
 * @author Eric Epstein
 */
public class SearchedHoldingSearcher extends Searcher {

    /**
     * Casts Searchable into HoldingUpdatable to get information for either
     * Equity or Holding. The ArrayList contains one-element and multi-
     * element ArrayList, the latter representing indices and sectors.
     *
     * @param s - Searchable
     * @return ArrayList<ArrayList<String>>
     */
    public ArrayList<ArrayList<String>> getSearchableStrings(Searchable s) {
```

```
        HoldingUpdatable eq = (HoldingUpdatable) (Object) s;
        ArrayList<ArrayList<String>> anObject = new
ArrayList<ArrayList<String>>();
        ArrayList<String> tickerSymbolItem = new ArrayList<String>();
        tickerSymbolItem.add(eq.getTickerSymbol());
        anObject.add(tickerSymbolItem);

        ArrayList<String> holdingNameItem = new ArrayList<String>();
        holdingNameItem.add(eq.getHoldingName());
        anObject.add(holdingNameItem);

        ArrayList<String> indices = eq.getIndices();
        anObject.add(indices);
        ArrayList<String> sectors = eq.getSectors();
        anObject.add(sectors);

        return anObject;
    }


}

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package model;

import javafx.collections.ObservableList;
import javafx.scene.Node;
import javafx.scene.control.ComboBox;
import javafx.scene.control.TextField;
import javafx.scene.layout.Pane;

import java.util.ArrayList;
import java.util.Observable;

/**
 * Provides general algorithm to produce matches and notifies observers
 * once the matching algorithm is executed
 *
 * @author Eric Epstein
 */
abstract public class Searcher extends Observable {

    /*
```

```
    * collection of fields with user input
    */
    ObservableList<Node> queries;

    /*
    * collection of Searchables to be traversed
    */
    ArrayList<Searchable> toBeSearched;

    /*
    * collection of Searchables that match the queries
    */
    ArrayList<Searchable> matches;

    /*
    * collection of collection of strings that are converted from queries
    */
    ArrayList<ArrayList<String>> searchableStrings;

    /**
     * Assigns values to global variables before executing the getMatches()
     * algorithm
     *
     * @param queries
     * @param toBeSearched
     */
    public void search(ObservableList<Node> queries, ArrayList<Searchable>
toBeSearched) {
        this.toBeSearched = toBeSearched;
        this.queries = queries;
        matches = new ArrayList<Searchable>();
        generateMatches();
    }

    /**
     * abstract method where subclasses convert the Searchable object
     * in a format such that can be tested against queries
     *
     * @param e
     * @return ArrayList<ArrayList<String>>
     */
    abstract ArrayList<ArrayList<String>> getSearchableStrings(Searchable e);

    /**
     * getMatch is common to all subclasses in that the keyword is used to
     * find the object of interest in the list of toBeSearched
     *
```

```
     * @param keyword
     * @return
     */
    public Searchable getMatch(String keyword) {
        if (toBeSearched != null) {
            for (Searchable s : toBeSearched) {
                if (s.getDisplayName().toUpperCase().equals(keyword.toUpperCase()))
{
                    return s;
                }
            }
        }
        return null;
    }

    /**
     * returns matches
     *
     * @return ArrayList<Searchable>
     */
    public ArrayList<Searchable> getMatches() {
        return matches;
    }

    /**
     * Algorithm to iterate through the list of possible matches and compare
     * each one against the fields that the user filled out where these fields
     * have search conditions like "exactly matches", "starts with", or "contains"
     * <p>
     * The algorithm delegates methods to subclasses when context is required.
     */
    public void generateMatches() {
        //Iterates through the list to be searched
        for (Searchable e : toBeSearched) {
            boolean isMatch = true;

            //Delegates to subclass to obtain string representation of object
            ArrayList<ArrayList<String>> allItems = getSearchableStrings(e);

            //Iterates through each field that a user filled out
            for (int i = 0; i < allItems.size(); i++) {

                ArrayList<String> anItem = allItems.get(i);

                Pane p = (Pane) queries.get(i);

                //Access the search condition for that field
```

```
        ComboBox cond = (ComboBox) p.getChildren().get(1);

        //Access what the user inputted in that field
        TextField content = (TextField) p.getChildren().get(2);

        //Enter inspection if the user inputted something
        if (!cond.getValue().equals("")) {
            boolean tempMatch = false;
            for (int j = 0; j < anItem.size(); j++) {
                String testStr = anItem.get(j);

                //Tests the condition of the search
                boolean oneMatch = false;
                switch (cond.getValue().toString()) {
                    case "exactly matches":
                        oneMatch = strExactlyMatches(content, testStr);
                        break;
                    case "starts with":
                        oneMatch = strStartsWith(content, testStr);
                        break;
                    case "contains":
                        oneMatch = strContains(content, testStr);
                        break;
                    case "":
                        oneMatch = true;
                }
                    /*
                    tempMatch reflects whether there was at least one match
                    for one field
                    */
                tempMatch = tempMatch || oneMatch;

            }
            //If there were no matches for one field, terminate
            isMatch = isMatch && tempMatch;

            if (!isMatch) {
                break;
            }
        }

    }
    //Adds a match to the collection
    if (isMatch) {
        matches.add(e);
    }
}
```

```
        //Notifies observers
        setChanged();
        notifyObservers();
    }

    private boolean strContains(TextField testField, String str) {
        return fieldHasContent(testField) &&
str.toUpperCase().contains(testField.getText().toUpperCase());
    }

    private boolean strStartsWith(TextField testField, String str) {
        return fieldHasContent(testField) &&
str.toUpperCase().startsWith(testField.getText().toUpperCase());
    }

    private boolean strExactlyMatches(TextField testField, String str) {
        return fieldHasContent(testField) &&
str.toUpperCase().equals(testField.getText().toUpperCase());
    }

    //Overloading fieldHasContent for TextField
    private boolean fieldHasContent(TextField aField) {
        return (aField.getText() != null && !aField.getText().isEmpty());
    }

}
```

```
package controller;

import model.*;

import java.util.ArrayList;

/**
 * Extends the Holding Algorithm by defining methods to sell a Holding
 * from outside and inside the FPTS.
 *
 * @author Eric Epstein
 */
public class SellHoldingAlgorithm extends HoldingAlgorithm {

    /*
    * context data
    */
    private Portfolio p;

    /*
```

```
    * collection of interest from which element of interest is identified
    */
    private ArrayList<Searchable> toBeSearched;

    /**
     * defines step in HoldingAlgorithm to establish context
     */
    @Override
    public void establishContext() {
        p = theFPTS.getPortfolio();
        toBeSearched = p.getHoldingSearchables();
    }

    /**
     * returns a list of Holding that is searchable
     * for possible sale
     */
    public ArrayList<Searchable> getToBeSearched() {
        return toBeSearched;
    }

    /**
     * Implements algorithm of a sale that is made inside FPTS.
     * <p>
     * Precondition - the following variables have already been assigned:
     * equityOfInterest
     * numOfShares
     * pricePerShare
     * cashAccountOfInterest
     */
    @Override
    public void processInsideFPTS() {
        Holding e = (Holding) equityOfInterest;

        /*
        * Validates the following conditions:
        *   - number of shares are positive
        *   - price per share is positve
        *   - the number of shares sold is less than or equal to the number of shares
        *     in the Holding
        */
        if (numOfShares > 0 && pricePerShare > 0 && e.getNumOfShares() >=
numOfShares) {
            /*
            * Creates Transaction that adds value to cashAccountOfInterest
            */
            CashAccount aC =
```

```
theFPTS.getPortfolio().getCashAccount(cashAccountOfInterest);
            Transaction t = new Deposit(aC, numOfShares * pricePerShare);
            p.add(t);
            e.subtractShares(numOfShares);
            /*
            * Portfolio removes Holding if the number of shares is equal to zero
            */
            if (e.getNumOfShares() == 0) {
               p.remove(e);
               theStage.setScene(theFPTS.getConfirmationScene());
            }
         } else {
            mainInput.setText("INVALID");
         }
      }

      /**
       * Implements algorithm of a sale that is made outside FPTS.
       */
      @Override
      public void processOutsideFPTS() {
         Holding e = (Holding) equityOfInterest;
         /*
         * Validates that the number of shares subtracted is less than
         * the current number of shares before subtraction.
         */
         if (e.getNumOfShares() > numOfShares) {
            e.subtractShares(numOfShares);
         /*
         * Removes Holding if the number of shares subtracted is equal
         * to the current number of shares.
         */
         } else if (e.getNumOfShares() == numOfShares) {
            p.remove(e);
            theStage.setScene(theFPTS.getConfirmationScene());
         /*
         * Warns the user of an invalid input.
         */
         } else {
            mainInput.setText("INVALID");
         }
      }
}

<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.chart.CategoryAxis?>
```

```
<?import javafx.scene.chart.LineChart?>
<?import javafx.scene.chart.NumberAxis?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>
<?import javafx.scene.text.*?>
<AnchorPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity"
minWidth="-Infinity" prefHeight="600.0"
          prefWidth="800.0" xmlns="http://javafx.com/javafx/8" xmlns:fx="http://
javafx.com/fxml/1"
          fx:controller="controller.SimulationController">
   <children>
      <MenuBar fx:id="myMenuBar" layoutY="2.0" minHeight="-Infinity"
minWidth="-Infinity" prefHeight="25.0"
               prefWidth="800.0" AnchorPane.topAnchor="2.0">
         <menus>
            <Menu mnemonicParsing="false" text="File">
               <items>
                  <MenuItem fx:id="home" mnemonicParsing="false"
onAction="#handleHomeMenuItemPressed"
                            text="Home"/>
                  <MenuItem fx:id="save" mnemonicParsing="false"
onAction="#handleSaveMenuItemPressed"
                            text="Save"/>
                  <MenuItem fx:id="Logout" mnemonicParsing="false"
onAction="#handleLogoutMenuItemPressed"
                            text="Logout"/>
                  <SeparatorMenuItem mnemonicParsing="false"/>
                  <MenuItem fx:id="Exit" mnemonicParsing="false"
onAction="#handleExitMenuItemPressed"
                            text="Exit"/>
               </items>
            </Menu>
            <Menu mnemonicParsing="false" text="Equities">
               <items>
                  <MenuItem fx:id="portfolio" mnemonicParsing="false"
onAction="#handlePortfolioMenuItemPressed"
                            text="Show Portfolio"/>
                  <MenuItem fx:id="buyEquities" mnemonicParsing="false"
                            onAction="#handleBuyEquitiesMenuItemPressed"
text="Buy Equities"/>
                  <MenuItem fx:id="sellEquities" mnemonicParsing="false"
                            onAction="#handleSellEquitiesMenuItemPressed"
text="Sell Equities"/>
               </items>
            </Menu>
            <Menu mnemonicParsing="false" text="Cash Account">
               <items>
```

```
                    <MenuItem fx:id="createCA" mnemonicParsing="false"
onAction="#handleCreateMenuItemPressed"
                              text="Create New"/>
                    <MenuItem fx:id="deposit" mnemonicParsing="false"
onAction="#handleDepositMenuItemPressed"
                              text="Deposit"/>
                    <MenuItem fx:id="withdraw" mnemonicParsing="false"
onAction="#handleWithdrawMenuItemPressed"
                              text="Withdraw"/>
                    <MenuItem fx:id="transfer" mnemonicParsing="false"
onAction="#handleTransferMenuItemPressed"
                              text="Transfer"/>
                    <SeparatorMenuItem mnemonicParsing="false"/>
                    <MenuItem fx:id="remove" mnemonicParsing="false"
onAction="#handleRemoveMenuItemPressed"
                              text="Remove Account"/>
                 </items>
              </Menu>
              <Menu mnemonicParsing="false" text="Help">
                 <items>
                    <MenuItem fx:id="about" mnemonicParsing="false"
onAction="#handleAboutMenuItemPressed"
                              text="About"/>
                 </items>
              </Menu>
           </menus>
        </MenuBar>
        <GridPane layoutX="-3.0" layoutY="29.0" prefHeight="573.0"
prefWidth="802.0">
           <columnConstraints>
              <ColumnConstraints hgrow="SOMETIMES" maxWidth="496.0"
minWidth="10.0" prefWidth="462.0"/>
              <ColumnConstraints hgrow="SOMETIMES"
maxWidth="402.66668701171875" minWidth="10.0" prefWidth="340.0"/>
           </columnConstraints>
           <rowConstraints>
              <RowConstraints maxHeight="317.99998474121094"
minHeight="10.0" prefHeight="118.33333587646484"
                              vgrow="SOMETIMES"/>
              <RowConstraints maxHeight="225.00000762939453"
minHeight="10.0" prefHeight="166.66666412353516"
                              vgrow="SOMETIMES"/>
              <RowConstraints maxHeight="174.33331298828125"
minHeight="10.0" prefHeight="158.33331298828125"
                              vgrow="SOMETIMES"/>
              <RowConstraints maxHeight="132.33334350585938"
minHeight="10.0" prefHeight="132.33334350585938"
```

```
                        vgrow="SOMETIMES"/>
            </rowConstraints>
            <children>
               <Label alignment="CENTER" prefHeight="27.0" prefWidth="329.0"
text="Financial Portfolio Tracking System"
                     GridPane.halignment="CENTER"
GridPane.valignment="CENTER">
                  <font>
                     <Font name="System Bold Italic" size="18.0"/>
                  </font>
               </Label>
               <LineChart fx:id="graph" prefHeight="283.0" prefWidth="409.0"
title="Portfolio Value"
                        GridPane.columnIndex="1" GridPane.rowSpan="2">
                  <xAxis>
                     <CategoryAxis side="BOTTOM"/>
                  </xAxis>
                  <yAxis>
                     <NumberAxis side="LEFT"/>
                  </yAxis>
               </LineChart>
               <GridPane GridPane.rowIndex="1" GridPane.rowSpan="3">
                  <columnConstraints>
                     <ColumnConstraints hgrow="SOMETIMES" maxWidth="226.0"
minWidth="10.0" prefWidth="185.0"/>
                     <ColumnConstraints hgrow="SOMETIMES" maxWidth="277.0"
minWidth="10.0" prefWidth="277.0"/>
                  </columnConstraints>
                  <rowConstraints>
                     <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES"/>
                     <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES"/>
                     <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES"/>
                     <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES"/>
                     <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES"/>
                  </rowConstraints>
                  <children>
                     <Label alignment="CENTER" prefHeight="20.0"
prefWidth="135.0" text="Number of Steps:"
                           GridPane.halignment="CENTER">
                        <font>
                           <Font size="14.0"/>
                        </font>
```

```
                </Label>
                <Label alignment="CENTER" prefHeight="20.0"
prefWidth="135.0" text="Time Interval: "
                    GridPane.halignment="CENTER" GridPane.rowIndex="1">
                  <font>
                    <Font size="14.0"/>
                  </font>
                </Label>
                <Label alignment="CENTER" prefHeight="20.0"
prefWidth="135.0" text="Show Step-by-Step:"
                    GridPane.halignment="CENTER" GridPane.rowIndex="2">
                  <font>
                    <Font size="14.0"/>
                  </font>
                </Label>
                <ChoiceBox fx:id="interval" prefWidth="150.0" value="Day"
GridPane.columnIndex="1"
                      GridPane.halignment="CENTER" GridPane.rowIndex="1"/
>
                <VBox alignment="CENTER" prefHeight="200.0"
prefWidth="100.0" spacing="10.0"
                    GridPane.columnIndex="1" GridPane.rowIndex="2">
                  <children>
                    <RadioButton fx:id="stepYes" mnemonicParsing="false"
                          onAction="#handleStepYesRadioButtonPressed"
selected="true" text="Yes">
                      <toggleGroup>
                        <ToggleGroup fx:id="stepByStep"/>
                      </toggleGroup>
                    </RadioButton>
                    <RadioButton fx:id="stepNo" mnemonicParsing="false"
                          onAction="#handleStepNoRadioButtonPressed"
selected="true" text="No"
                          toggleGroup="$stepByStep"/>
                  </children>
                </VBox>
                <TextField fx:id="numSteps" promptText="Number of Steps"
GridPane.columnIndex="1"/>
                <Label alignment="CENTER" prefHeight="20.0"
prefWidth="135.0" text="Simulation Style"
                    GridPane.halignment="CENTER" GridPane.rowIndex="3">
                  <font>
                    <Font size="14.0"/>
                  </font>
                </Label>
                <HBox alignment="CENTER" prefHeight="100.0"
prefWidth="200.0" spacing="10.0"
```

```
                        GridPane.columnIndex="1" GridPane.halignment="CENTER"
GridPane.rowIndex="3"
                        GridPane.valignment="CENTER">
                       <children>
                         <RadioButton fx:id="bearSim" mnemonicParsing="false"

onAction="#handleBearSimulateRadioButtonPressed" text="Bear"
                                      toggleGroup="$simulateMethod">
                            <toggleGroup>
                               <ToggleGroup fx:id="simulateMethod"/>
                            </toggleGroup>
                         </RadioButton>
                         <RadioButton fx:id="bullSim" mnemonicParsing="false"
                                      onAction="#handleBullSimulateRadioButtonPressed"
text="Bull"
                                      toggleGroup="$simulateMethod"/>
                         <RadioButton fx:id="noSim" mnemonicParsing="false"
                                      onAction="#handleNoSimulateRadioButtonPressed"
selected="true"
                                      text="No Growth" toggleGroup="$simulateMethod"/>
                       </children>
                    </HBox>
                    <Label alignment="CENTER" prefHeight="17.0"
prefWidth="190.0"
                           text="Price per Annum (if applicable): "
GridPane.halignment="CENTER"
                           GridPane.rowIndex="4"/>
                    <TextField fx:id="priceAnnum" promptText="Price-Per-Annum"
GridPane.columnIndex="1"
                               GridPane.rowIndex="4"/>
                 </children>
              </GridPane>
              <Button fx:id="simulate" alignment="CENTER"
contentDisplay="CENTER" defaultButton="true"
                      mnemonicParsing="false"
onAction="#handleSimulateButtonPressed" prefHeight="25.0"
                      prefWidth="90.0" text="Simulate" GridPane.columnIndex="1"
GridPane.halignment="CENTER"
                      GridPane.rowIndex="2"/>
              <Label fx:id="error" alignment="CENTER" contentDisplay="CENTER"
prefHeight="50.0" prefWidth="390.0"
                     GridPane.columnIndex="1" GridPane.halignment="CENTER"
GridPane.rowIndex="3"/>
           </children>
        </GridPane>
     </children>
</AnchorPane>
```

```
package controller;

import gui.FPTS;
import javafx.collections.FXCollections;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Node;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.ChoiceBox;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.stage.Stage;
import model.BearSimulator;
import model.BullSimulator;
import model.NoGrowthSimulator;
import model.Simulator;

import java.io.IOException;
import java.net.URL;
import java.util.ResourceBundle;

/**
 * Created by Brockway on 3/12/16.
 */
public class SimulationController extends MenuController {
    @FXML
    private Label error;
    @FXML
    private TextField numSteps;
    @FXML
    private ChoiceBox<String> interval;
    @FXML
    private Button stepButton;
    @FXML
    private TextField priceAnnum;
    private Simulator currentSimulator;
    @FXML
    private Label pValue;
    @FXML
    private Label stepNumber;
    private String simulation = "NOGROWTH";
    private boolean steps = false;
```

```
    @FXML
    protected void handleBearSimulateRadioButtonPressed(ActionEvent event) {
        simulation = "BEAR";
    }

    @FXML
    protected void handleStepYesRadioButtonPressed(ActionEvent event) {
        steps = true;
    }

    @FXML
    protected void handleStepNoRadioButtonPressed(ActionEvent event) {
        steps = false;
    }

    @FXML
    protected void handleBullSimulateRadioButtonPressed(ActionEvent event) {
        simulation = "BULL";
    }

    @FXML
    protected void handleNoSimulateRadioButtonPressed(ActionEvent event) {
        simulation = "NOGROWTH";
    }


    /**
     * Checks to make sure the number of steps entered is valid.
     * If the simulation is no growth then the simulation will be called,
     * but if the simulation is a bull or bear market simulation the user
     * will be asked to input a percentage for price increase or decrease
     * per year.
     *
     * @param event - ActionEvent - The event that is created when Simulate
button is pressed.
     * @throws java.io.IOException - Exception thrown if the SimulationPage.fxml
is not found.
     */
    public void handleSimulateButtonPressed(ActionEvent event) throws
IOException {
        if (numSteps.getText().length() != 0 && priceAnnum.getText().length() != 0) {
            try {
                int numberOfSteps = Integer.parseInt(numSteps.getText());
                String curInterval = interval.getValue();
                Boolean hasSteps;
                if (steps) {
                    hasSteps = true;
```

```
            } else {
                hasSteps = false;
            }
            if (simulation.equals("NOGROWTH")) {
                currentSimulator = new NoGrowthSimulator(numberOfSteps,
curInterval, hasSteps);
                System.out.println("NOGROWTH");
            } else {
                if (priceAnnum.getText().length() != 0) {
                    String pricePerAnum = priceAnnum.getText();
                    try {
                        double pricePerYearAsDouble =
Double.parseDouble(pricePerAnum);
                        if (pricePerYearAsDouble < 1.00 && pricePerYearAsDouble >
0) {
                            if (simulation.equals("BEAR")) {
                                System.out.println("BEARSIM");
                                currentSimulator = new BearSimulator(numberOfSteps,
curInterval, hasSteps, pricePerYearAsDouble);
                            } else {
                                System.out.println("BULLSIM");
                                currentSimulator = new BullSimulator(numberOfSteps,
curInterval, hasSteps, pricePerYearAsDouble);
                            }
                        } else {
                            error.setText("Please Enter a value between 0 and 1 for the
Price per Annum.");
                        }
                    } catch (NumberFormatException x) {
                        error.setText("Invalid Format. Please enter a percent value for
the number of steps.");
                    }
                } else {
                    error.setText("Please enter a percent value for the Price Per
Annum.");
                }
            }
            FPTS.setCurrentSimulator(currentSimulator);
            if (hasSteps) {
                FPTS.setSimulationValue(currentSimulator.simulate(1));
            } else {

FPTS.setSimulationValue(currentSimulator.simulate(numberOfSteps));
            }
            Parent parent = FXMLLoader.load(getClass().getResource("../gui/
SimulationPage.fxml"));
            Scene scene = new Scene(parent);
```

```
            Stage stage = (Stage) ((Node)
event.getSource()).getScene().getWindow();
            stage.setScene(scene);
            stage.show();
            System.out.println("Value returned = " + FPTS.getSimulationValue());
        } catch (NumberFormatException x) {
            error.setText("Invalid Format. Please enter an integer for the number of
steps.");
        }
    } else if (numSteps.getText().length() != 0) {
        error.setText("Please enter a decimal value for the Price Per Annum.");
    } else {
        error.setText("Please enter an integer for the number of steps.");
    }
}

@FXML
protected void handleStepButtonPressed(ActionEvent event) throws
IOException {
    currentSimulator = FPTS.getCurrentSimulator();
    if (currentSimulator.getCurrentStep() < currentSimulator.getTotalSteps()) {
        FPTS.setSimulationValue(FPTS.getSimulationValue() +
currentSimulator.simulate(1));

        Stage stage = (Stage) ((Node)
event.getSource()).getScene().getWindow();
        stage.setScene(new
Scene(FXMLLoader.load(getClass().getResource("../gui/
SimulationPage.fxml"))));
        stage.show();
    }
}

@FXML
protected void handleResetToStartButtonPressed(ActionEvent event) throws
IOException {
    Stage stage = (Stage) ((Node) event.getSource()).getScene().getWindow();
    stage.setScene(new Scene(FXMLLoader.load(getClass().getResource("../
gui/SimulatePage.fxml"))));
    stage.show();
}

@FXML
protected void handleResetToCurrentPricesButtonPressed(ActionEvent event)
{

}
```

```
    /**
     * Method used to initialize the choiceBox on the SimulatePage and the
simulation value on the SimulationPage.
     */
    @Override
    public void initialize(URL location, ResourceBundle resources) {
        if (interval != null) {
            interval.setItems(FXCollections.observableArrayList( //TODO: CHECK
THIS CALL**
                    "Day", "Month", "Year"
            ));
        }
        if (pValue != null) {
            pValue.setText("$" + FPTS.getSimulationValue());
            stepNumber.setText("" + FPTS.getCurrentSimulator().getCurrentStep());
            currentSimulator = FPTS.getCurrentSimulator();
            if (currentSimulator.getCurrentStep() >= currentSimulator.getTotalSteps())
{
                stepButton.setDisable(true);
            }

        }

    }
}

<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.chart.CategoryAxis?>
<?import javafx.scene.chart.LineChart?>
<?import javafx.scene.chart.NumberAxis?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>
<?import javafx.scene.text.*?>
<AnchorPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity"
minWidth="-Infinity" prefHeight="600.0"
            prefWidth="800.0" xmlns="http://javafx.com/javafx/8" xmlns:fx="http://
javafx.com/fxml/1"
            fx:controller="controller.SimulationController">
    <children>
        <MenuBar fx:id="myMenuBar" layoutY="2.0" minHeight="-Infinity"
minWidth="-Infinity" prefHeight="25.0"
                 prefWidth="800.0" AnchorPane.topAnchor="2.0">
            <menus>
                <Menu mnemonicParsing="false" text="File">
                    <items>
```

```
                <MenuItem fx:id="home" mnemonicParsing="false"
onAction="#handleHomeMenuItemPressed"
                        text="Home"/>
                <MenuItem fx:id="save" mnemonicParsing="false"
onAction="#handleSaveMenuItemPressed"
                        text="Save"/>
                <MenuItem fx:id="Logout" mnemonicParsing="false"
onAction="#handleLogoutMenuItemPressed"
                        text="Logout"/>
                <SeparatorMenuItem mnemonicParsing="false"/>
                <MenuItem fx:id="Exit" mnemonicParsing="false"
onAction="#handleExitMenuItemPressed"
                        text="Exit"/>
            </items>
          </Menu>
          <Menu mnemonicParsing="false" text="Equities">
            <items>
              <MenuItem fx:id="portfolio" mnemonicParsing="false"
onAction="#handlePortfolioMenuItemPressed"
                        text="Show Portfolio"/>
              <MenuItem fx:id="buyEquities" mnemonicParsing="false"
                        onAction="#handleBuyEquitiesMenuItemPressed"
text="Buy Equities"/>
              <MenuItem fx:id="sellEquities" mnemonicParsing="false"
                        onAction="#handleSellEquitiesMenuItemPressed"
text="Sell Equities"/>
            </items>
          </Menu>
          <Menu mnemonicParsing="false" text="Cash Account">
            <items>
              <MenuItem fx:id="createCA" mnemonicParsing="false"
onAction="#handleCreateMenuItemPressed"
                        text="Create New"/>
              <MenuItem fx:id="deposit" mnemonicParsing="false"
onAction="#handleDepositMenuItemPressed"
                        text="Deposit"/>
              <MenuItem fx:id="withdraw" mnemonicParsing="false"
onAction="#handleWithdrawMenuItemPressed"
                        text="Withdraw"/>
              <MenuItem fx:id="transfer" mnemonicParsing="false"
onAction="#handleTransferMenuItemPressed"
                        text="Transfer"/>
              <SeparatorMenuItem mnemonicParsing="false"/>
              <MenuItem fx:id="remove" mnemonicParsing="false"
onAction="#handleRemoveMenuItemPressed"
                        text="Remove Account"/>
            </items>
```

```
            </Menu>
            <Menu mnemonicParsing="false" text="Help">
              <items>
                <MenuItem fx:id="about" mnemonicParsing="false"
onAction="#handleAboutMenuItemPressed"
                          text="About"/>
              </items>
            </Menu>
          </menus>
        </MenuBar>
        <GridPane layoutX="-3.0" layoutY="29.0" prefHeight="573.0"
prefWidth="802.0">
          <columnConstraints>
            <ColumnConstraints hgrow="SOMETIMES" maxWidth="496.0"
minWidth="10.0" prefWidth="399.33331298828125"/>
            <ColumnConstraints hgrow="SOMETIMES"
maxWidth="402.66668701171875" minWidth="10.0"
                               prefWidth="402.66668701171875"/>
          </columnConstraints>
          <rowConstraints>
            <RowConstraints maxHeight="317.99998474121094"
minHeight="10.0" prefHeight="118.33333587646484"
                            vgrow="SOMETIMES"/>
            <RowConstraints maxHeight="225.00000762939453"
minHeight="10.0" prefHeight="166.66666412353516"
                            vgrow="SOMETIMES"/>
            <RowConstraints maxHeight="174.33331298828125"
minHeight="10.0" prefHeight="158.33331298828125"
                            vgrow="SOMETIMES"/>
            <RowConstraints maxHeight="132.33334350585938"
minHeight="10.0" prefHeight="132.33334350585938"
                            vgrow="SOMETIMES"/>
          </rowConstraints>
          <children>
            <Label alignment="CENTER" prefHeight="27.0" prefWidth="329.0"
text="Financial Portfolio Tracking System"
                   GridPane.halignment="CENTER"
GridPane.valignment="CENTER">
              <font>
                <Font name="System Bold Italic" size="18.0"/>
              </font>
            </Label>
            <LineChart fx:id="graph" prefHeight="283.0" prefWidth="409.0"
title="Portfolio Value"
                       GridPane.columnIndex="1" GridPane.rowSpan="2">
              <xAxis>
                <CategoryAxis side="BOTTOM"/>
```

```
            </xAxis>
            <yAxis>
               <NumberAxis side="LEFT"/>
            </yAxis>
         </LineChart>
         <GridPane GridPane.rowIndex="2">
            <columnConstraints>
               <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0"
prefWidth="100.0"/>
               <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0"
prefWidth="100.0"/>
            </columnConstraints>
            <rowConstraints>
               <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES"/>
               <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES"/>
            </rowConstraints>
            <children>
               <Label alignment="CENTER" prefHeight="20.0"
prefWidth="135.0" text="Current Step: "
                  GridPane.halignment="CENTER">
                  <font>
                     <Font size="14.0"/>
                  </font>
               </Label>
               <Label fx:id="stepNumber" prefHeight="53.0" prefWidth="137.0"
GridPane.columnIndex="1"
                  GridPane.halignment="CENTER"/>
               <Label alignment="CENTER" prefHeight="20.0"
prefWidth="135.0" text="Change in Portfolio Value: "
                  GridPane.halignment="CENTER" GridPane.rowIndex="1">
                  <font>
                     <Font size="14.0"/>
                  </font>
               </Label>
               <Label fx:id="pValue" prefHeight="53.0" prefWidth="137.0"
GridPane.columnIndex="1"
                  GridPane.halignment="CENTER" GridPane.rowIndex="1"/>
            </children>
         </GridPane>
         <Label fx:id="errors" alignment="CENTER" contentDisplay="CENTER"
prefHeight="50.0" prefWidth="390.0"
            GridPane.columnIndex="1" GridPane.halignment="CENTER"
GridPane.rowIndex="3"/>
         <Label alignment="CENTER" prefHeight="27.0" prefWidth="329.0"
text="Simulation"
```

```
                  GridPane.halignment="CENTER" GridPane.rowIndex="1">
                <font>
                  <Font name="System Bold Italic" size="18.0"/>
                </font>
              </Label>
              <VBox alignment="CENTER" prefHeight="200.0" prefWidth="100.0"
spacing="20.0" GridPane.columnIndex="1"
                    GridPane.rowIndex="2">
                <children>
                  <Button fx:id="stepButton" alignment="CENTER"
contentDisplay="CENTER" mnemonicParsing="false"
                          prefHeight="25.0" prefWidth="170.0" text="Next Step"
                          onAction="#handleStepButtonPressed">
                    <font>
                      <Font size="13.0"/>
                    </font>
                  </Button>
                  <Button alignment="CENTER" contentDisplay="CENTER"
mnemonicParsing="false"
                          prefHeight="25.0" prefWidth="170.0" text="Reset to Current
Prices"
                          onAction="#handleResetToCurrentPricesButtonPressed">
                    <font>
                      <Font size="13.0"/>
                    </font>
                  </Button>
                  <Button alignment="CENTER" contentDisplay="CENTER"
mnemonicParsing="false"
                          prefHeight="25.0" prefWidth="170.0" text="Reset Simulation"
                          onAction="#handleResetToStartButtonPressed">
                    <font>
                      <Font size="13.0"/>
                    </font>
                  </Button>
                </children>
              </VBox>
            </children>
          </GridPane>
        </children>
</AnchorPane>

package model;

/**
 * Created by Brockway on 3/12/16.
 */
public interface Simulator {
```

```
    public double simulate(int numberOfSteps);

    public int getCurrentStep();

    public int getTotalSteps();
}
```

```
package model;

/**
 * Interface that defines execute behaviors in association with CashAccount
 *
 * @author Eric Epstein
 */
public interface Transaction {

    /**
     * Run operation on object
     */
    public void execute();

    /**
     * returns CashAccount unique to the realization
     *
     * @return CashAccount
     */
    public CashAccount getCashAccount();
}
```

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package controller;

import gui.FPTS;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.DatePicker;
import javafx.scene.control.Label;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import model.Transaction;
```

```
import java.time.ZoneId;
import java.util.ArrayList;
import java.util.Date;

/**
 * Displays Transaction objects in one Scene.
 *
 * @author Eric Epstein
 */
public class TransactionDisplayer implements Displayer {

    /*
    * context data
    */
    FPTS theFPTS;
    ArrayList<Transaction> transactions;
    VBox results;

    /*
    * Establishes context data and overrides Displayer's display method
    * by calling a scene constructor.
    */
    @Override
    public void display(FPTS theFPTS) {
        this.theFPTS = theFPTS;
        transactions = theFPTS.getPortfolio().getTransactions();
        theFPTS.getStage().setScene(getTransactionDisplayScene());
    }

    /**
     * Helper method to construct Scene with controller functionality for
     * start and end dates.
     *
     * @return
     */
    private Scene getTransactionDisplayScene() {

        VBox split = new VBox();
        VBox queries = new VBox();
        HBox aField = new HBox();

        /**
         * Field to select start date
         */
        DatePicker startDate = new DatePicker();
        Label aLabel = new Label("Start date: ");
        aField.getChildren().addAll(aLabel, startDate);
```

```
        queries.getChildren().add(aField);

        /*
        * Field to select end date
        */
        aField = new HBox();
        DatePicker endDate = new DatePicker();
        aLabel = new Label("End date: ");
        aField.getChildren().addAll(aLabel, endDate);

        queries.getChildren().add(aField);

        /*
        * Initially displays all Transaction objects.
        */
        VBox results = new VBox();
        for (Transaction t : transactions) {
            results.getChildren().add(new Label(t.toString()));
        }

        Button submitBtn = new Button();
        submitBtn.setText("Search");
        /*
         * Filters list of Transaction in case user inputs two valid start and end
         * dates.
         */
        submitBtn.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent e) {
                results.getChildren().clear();
                if (startDate.getValue() != null && endDate.getValue() != null) {
                    Date start =
Date.from(startDate.getValue().atStartOfDay(ZoneId.systemDefault()).toInstant())
;
                    Date end =
Date.from(endDate.getValue().atStartOfDay(ZoneId.systemDefault()).toInstant());
                    /*
                    * Update display to represent filtered Transaction objects
                    */
                    for (Transaction t : transactions) {
                        Date aDate = t.getCashAccount().getDateAdded();
                        /*
                        * Add to display if Transaction object is after the start
                        * date and before the end date.
                        */
                        if (aDate.after(start) && aDate.before(end)) {
                            results.getChildren().add(new Label(t.toString()));
```

```
                }
            }
        }
    }
    });
    split.getChildren().addAll(theFPTS.getNav(), queries, submitBtn, results);
    Scene transactionDisplayScene = new Scene(split, theFPTS.getWidth(),
theFPTS.getHeight());
    return transactionDisplayScene;
  }
}

package controller;

import model.CashAccount;
import model.Deposit;
import model.Transaction;
import model.Withdrawal;

import java.util.ArrayList;
import java.util.Observable;

/**
 * Implements final step in CashAccountAlgorithm by specifying amount and
another
 * CashAccount to which the previously specified CashAccount is transfered.
 *
 * @author ericepstein
 */
public class TransferCashAccountAlgorithm extends CashAccountAlgorithm {

  /*
  * context data
  */
  CashAccount c2;
  protected ArrayList<Double> amounts;

  /*
  * number of times notified
  */
  static int numCalled = 0;

  /*
  * Implements the action() step by creating a new CashAccountFinder to
establish
  * the second CashAccount
  */
```

```
public void action() {
    c2 = new CashAccount("", 0, null);
    CashAccountFinder caFinder = new CashAccountFinder(theFPTS, c2);
    caFinder.addObserver(this);
}

/*
* Processes notifications by calling specific methods based on number of
* current notification.
*/
public void update(Observable o, Object args) {

    numCalled++;

    switch (numCalled) {
        /*
        * At first notification, the first CashAccount is already set.
        * Allow superclass to handle update.
        */
        case (1):
            super.update(o, args);
            break;
        /*
        * At second notification, the second CashAccount is already set
          Call method to transition to numerical input page.
        */
        case (2):
            getAmountInput();
            break;
        /*
        * At third notification, the amount input is already set.
        * Call method to perform transaction.
        */
        case (3):
            performTransaction();
            break;
    }
}

/*
* Creates AmountInput to manage user interface and input to
* receive validated numerical value.
*/
public void getAmountInput() {
    amounts = new ArrayList<Double>();
    AmountInput amountInput = new AmountInput(theFPTS, amounts);
    amountInput.addObserver(this);
```

```
    }

    /*
    * Validates amount withdrawn before creating Deposit and Withdraw objects
    * for respective CashAccount objects.
    */
    public void performTransaction() {

        double amount = amounts.get(0);
        /*
        * If the current value exceeds or equals the amount withdrawn, create
        * respective Transaction objects
        */
        if (c.getValue() >= amount) {
            CashAccount aC = theFPTS.getPortfolio().getCashAccount(c);

            Transaction t = new Withdrawal(aC, amount);
            theFPTS.getPortfolio().add(t);

            aC = theFPTS.getPortfolio().getCashAccount(c2);
            t = new Deposit(aC, amount);
            theFPTS.getPortfolio().add(t);
            theFPTS.getStage().setScene(theFPTS.getConfirmationScene());
        } else {
            theFPTS.getStage().setScene(theFPTS.getErrorScene());
        }

        /*
        * Resets the number of notifications to 0 for later invocations.
        */
        numCalled = 0;

    }

}

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package model;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
```

```
import java.util.ArrayList;

/**
 * Stores attributes of User, generates list of Users, and validates User
 *
 * @author Kimberly Sookoo and Ian London
 */
public class User {
    private String loginID;
    private String password;
    private Portfolio myPortfolio;
    private static ArrayList<User> userList = new ArrayList<User>();//Holds All
Registered Users

    /**
     * When creating a new portfolio, the system shall allow the user to
     * import holdings and transactions to initialize the new portfolio. THIS IS NOT
ALLOWED YET
     *
     * @param loginID  - String - Login ID of of the User.
     * @param password - String - Password of the User, stored in a hashed
setting.
     */
    public User(String loginID, String password) {
        this.loginID = loginID;
        this.password = hash(password);
        //this.myPortfolio = new Portfolio(); //TODO: check if user wants to import
holdings and transactions
    }

    /**
     * Ensures encryption by incrementing each password character by 1
     *
     * @param password - String
     * @return String of password
     */
    public String hash(String password) {
        String encryptedPW = "";
        for (int i = 0; i < password.length(); i++) {
            char encryptedChar = (char) (password.charAt(i) + 1);
            encryptedPW += encryptedChar;
        }
        return encryptedPW;
    }

    /**
     * Ensures decryption by decrementing each password character by 1
```

```
     *
     * @param password
     * @return String of password
     */
    public static String unHash(String password) {
        String textPass = "";
        for (int i = 0; i < password.length(); i++) {
            char encryptedChar = (char) (password.charAt(i) - 1);
            textPass += encryptedChar;
        }
        return textPass;
    }

    /**
     * Equality function that matches user and password
     *
     * @param u
     * @return boolean
     */
    public boolean equals(User u) {
        return u.getLoginID().equals(loginID) &&
u.getPassword().equals(password);
    }

    /**
     * Overrides equals() method
     * <p>
     * Precondition - object passed must be a User object
     *
     * @param o
     * @return
     */
    @Override
    public boolean equals(Object o) {
        return equals((User) o);
    }

    /**
     * Returns login ID
     *
     * @return String
     */
    public String getLoginID() {
        return loginID;
    }

    /**
```

```
     * Returns password
     *
     * @return String
     */
    private String getPassword() {
        return password;
    }

    /**
     * Public method used to populate the users ArrayList<User> from the
UserData.csv file.
     */
    public static void fillUsers() {
        String csv = "JavaFXApp/src/model/DataBase/UserData.csv";
        BufferedReader reader = null;
        String line;

        try {
            reader = new BufferedReader(new FileReader(csv));
            while ((line = reader.readLine()) != null) {
                String[] split = line.split(",");
                User newUser = new User(split[0], unHash(split[1]));
                userList.add(newUser);
            }
        } catch (FileNotFoundException e) {
            System.out.println("src/model/DataBase/UserData.csv not found! Please
try again.");
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            if (reader != null) {
                try {
                    reader.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }

    /**
     * Returns validation by testing a User object against a list of existing
     * User objects
     *
     * @return boolean
     */
    public boolean validateUser() {
```

```
        for (User usr : userList) {
            if (this.equals(usr)) {
                return true;
            }
        }
        return false;
    }

    /**
     * Returns whether the login ID exists in a collection of User objects
     *
     * @return boolean
     */
    public static boolean ValidLoginID(String id) {
        for (User usr : userList) {
            if (usr.getLoginID().equals(id)) {
                return false;
            }
        }
        return true;
    }

    /**
     * Adds a User object to list of users
     *
     * @param u
     */
    public static void addToList(User u) {
        userList.add(u);
    }

    /**
     * returns Portfolio
     *
     * @return Portfolio
     */
    public Portfolio getMyPortfolio() {
        return myPortfolio;
    }

    public void setMyPortfolio(Portfolio portfolio) {
        this.myPortfolio = portfolio;
    }

}

package model;
```

```
import java.time.LocalDate;
import java.time.ZoneId;
import java.util.Date;

/**
 * Performs a withdrawal at a given amount on a given CashAccount when
 * called to do so
 *
 * @author Eric Epstein
 */
public class Withdrawal implements Transaction {

    private CashAccount c;
    private double amount;

    /**
     * Constructs a Withdrawal command
     *
     * @param c
     * @param amount
     */
    public Withdrawal(CashAccount c, double amount) {
        this.c = c;
        this.amount = amount;
    }

    /**
     * invokes operation
     */
    public void execute() {
        c.withdraw(amount);
    }

    /**
     * returns a String representation for display
     *
     * @return String
     */
    public String toString() {
        Date theDate = c.getDateAdded();
        LocalDate localDate =
theDate.toInstant().atZone(ZoneId.systemDefault()).toLocalDate();
        String theDateString = (localDate.getMonthValue() + "/" +
                localDate.getDayOfMonth() +
                "/" + localDate.getYear());
        return "Withdrew " + amount + " from " + c.getAccountName() + " on " +
```

```
theDateString;
    }

    /**
     * returns CashAccount associated with Withdrawal
     *
     * @return CashAccount
     */
    public CashAccount getCashAccount() {
        return c;
    }

}

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package controller;

import model.CashAccount;
import model.Transaction;
import model.Withdrawal;

/**
 * Implements final step in CashAccountAlgorithm by creating a Withdraw object.
 *
 * @author Eric Epstein
 */
public class WithdrawCashAccountAlgorithm extends
ChangeCashAccountAlgorithm {

    /**
     * Creates a Withdraw object with validated CashAccount if the amount is
validated.
     */
    @Override
    public void performTransaction() {

        double amount = amounts.get(0);

        /*
        * Validates whether amount withdrawn will lead to negative overall value.
        */
        if (c.getValue() >= amount) {
```

```
            CashAccount aC = theFPTS.getPortfolio().getCashAccount(c);

            Transaction t = new Withdrawal(aC, amount);
            theFPTS.getPortfolio().add(t);

            theFPTS.getStage().setScene(theFPTS.getConfirmationScene());
        } else {
            theFPTS.getStage().setScene(theFPTS.getErrorScene());
        }
    }

}

package model.DataBase;

import gui.FPTS;
import model.CashAccount;
import model.Holding;
import model.Transaction;
import model.User;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.util.ArrayList;

/**
 * Created by Kimberly Sookoo on 3/2/16.
 */
public class WriteFile {

    FPTS fpts = FPTS.getSelf();

    /*
    Public method that checks to see if customer has a portfolio
    */
    public boolean hasPortfolio(User user) {
        File directory = new File("JavaFXApp/src/model/Database/Portfolios/" +
user.getLoginID());
        return directory.exists();
    }

    /*
 Public method that creates portfolio for customer.
 */
    public void createPortfolioForUser(User user) {
        try {
```

```
            File directory = new File("JavaFXApp/src/model/Database/Portfolios/" +
user.getLoginID());
            if (!directory.exists()) {
                directory.mkdir();
            }
            File transFile = new File(directory, "Trans.csv");
            File cashFile = new File(directory, "Cash.csv");
            File holdingsFile = new File(directory, "Holdings.csv");
            transFile.createNewFile();
            cashFile.createNewFile();
            holdingsFile.createNewFile();
            FileWriter writerT = new FileWriter(transFile, true);
            FileWriter writerC = new FileWriter(cashFile, true);
            FileWriter writerH = new FileWriter(holdingsFile, true);

            //this.transactionsWriter(user, writerT);
            this.cashAccountsWriter(writerC);
            this.holdingsWriter(writerH);

        } catch (Exception e1) {
            e1.printStackTrace();
        }

        System.out.println("Created");
    }

    /*
   Public method that removes portfolio for customer.
   */
    public void removePortfolioForUser(User user) {
        File directory = new File("JavaFXApp/src/model/Database/Portfolios/" +
user.getLoginID());
        File transFile = new File(directory, "/Trans.csv");
        File cashFile = new File(directory, "/Cash.csv");
        File holdingsFile = new File(directory, "/Holdings.csv");
        transFile.delete();
        cashFile.delete();
        holdingsFile.delete();
        directory.delete();
    }

    /*
    Public method that updates portfolio for given user.
     */
    public void updatePortfolioForUser(User user) {
        try {
            File directory = new File("JavaFXApp/src/model/Database/Portfolios/" +
```

```
user.getLoginID());
            File transFile = new File(directory, "Trans.csv");
            File cashFile = new File(directory, "Cash.csv");
            File holdingsFile = new File(directory, "Holdings.csv");
            FileWriter writerT = new FileWriter(transFile, true);
            FileWriter writerC = new FileWriter(cashFile, true);
            FileWriter writerH = new FileWriter(holdingsFile, true);

            cashAccountsWriter(writerC);
            holdingsWriter(writerH);

            System.out.println("Has anything extra been written?");

        } catch (Exception e) {
        }
    }

    /*
    Private method for writing down holdings
     */
    private void holdingsWriter(FileWriter writer) {
        try {
            BufferedWriter bufferedWriter = new BufferedWriter(writer);
            ArrayList<Holding> holding = fpts.getPortfolio().getHoldings();
            for (int i = 0; i < holding.size(); i++) {
                bufferedWriter.write("\"" + holding.get(i).getSymbol() + "\",\"" +
holding.get(i).getHoldingName() + "\",\"" +
                        holding.get(i).getValuePerShare() + "\",\"" +
holding.get(i).getNumOfShares() + "\",\"" +
                        holding.get(i).getAcquisitionDate() + "\",\"" +
holding.get(i).getIndices() + "\",\"" +
                        holding.get(i).getSectors() + "\"");
                bufferedWriter.newLine();
            }
            bufferedWriter.close();
        } catch (Exception e) {

        }
    }

    /*
    Private method for writing down cash accounts
     */
    private void cashAccountsWriter(FileWriter writer) {
        try {
            BufferedWriter bufferedWriter = new BufferedWriter(writer);
            ArrayList<CashAccount> cashAccounts =
```

```
fpts.getPortfolio().getCashAccounts();
            for (int i = 0; i < cashAccounts.size(); i++) {
                bufferedWriter.write("\"" + cashAccounts.get(i).getAccountName() + "\",
\"" + cashAccounts.get(i).getValue() +
                        "\",\"" + cashAccounts.get(i).getDateAdded() + "\"");
                bufferedWriter.newLine();
            }
            bufferedWriter.close();
        } catch (Exception e) {

        }
    }

    /*
    Private method for writing down cash accounts
     */
    private void transactionsWriter(User user, FileWriter writer) {
        try {
            BufferedWriter bufferedWriter = new BufferedWriter(writer);
            ArrayList<Transaction> transactions =
user.getMyPortfolio().getTransactions();
            for (int i = 0; i < transactions.size(); i++) {

                bufferedWriter.newLine();
            }
            bufferedWriter.close();
        } catch (Exception e) {

        }
    }
}
```