1. HEADER 2

2. CLASS SUMMARY 3

3. CLASS DIAGRAMS 3

3.1. CLASS:RECORDER (THE CLASS TO CONTAIN THE TRANSACTION RECORDS AND HAVE METHODS THAT CREATE SUMMARY REPORTS FOR EACH INDIVIDUAL INVESTOR.) 3

3.1.1. Create a table or UML Class diagram showing the public fields and methods of that type 3

3.1.2. For each public field, describe its purpose and valid range of values ... 4

3.1.3. For each public constructor, describe its parameters 4

3.1.4. For each public method of the class, add a row with the following columns: 4

3.2. CLASS:TRANSACTIONNODE (THE CLASS TO STORE THE INFORMATION OF EACH TRANSACTION.) 5

3.2.1. Create a table or UML Class diagram showing the public fields and methods of that type 5

3.2.2. For each public field, describe its purpose and valid range of values ... 6

3.2.3. For each public constructor, describe its parameters 6

3.2.4. For each public method of the class, add a row with the following columns: 6

3.3. CLASS:INVESTOR (THE CLASS TO STORE EACH INVESTOR'S INFORMATION.) 6

3.3.1. Create a table or UML Class diagram showing the public fields and methods of that type 7

3.3.2. For each public field, describe its purpose and valid range of values ... 7

3.3.3. For each public constructor, describe its parameters 7

3.3.4. For each public method of the class, add a row with the following columns: 7

3.4. CLASS:INVESTMENTTARGET (THE CLASS TO STORE THE INFORMATION OF ONE INVESTMENT TARGET.) 7

3.4.1. Create a table or UML Class diagram showing the public fields and methods of that type 8

3.4.2. For each public field, describe its purpose and valid range of values ... 8

3.4.3. For each public constructor, describe its parameters 8

3.4.1. For each public method of the class, add a row with the following columns: 8

3.5. CLASS:FILEREADER (THE CLASS TO READ EXTERNAL FILES) 8

3.5.1. Create a table or UML Class diagram showing the public fields and methods of that type 8

3.5.2. For each public field, describe its purpose and valid range of values ... 9
3.5.3. For each public constructor, describe its parameters 9
3.5.4. For each public method of the class, add a row with the following columns:..... 9
3.6. CLASS:MAIN (THE CLASS TO SET UP AND ENABLE THE GUI OF THE PROGRAM) 10
3.6.1. Create a table or UML Class diagram showing the public fields and methods of that type..... 10
3.6.2. For each public field, describe its purpose and valid range of values . 10
3.6.3. For each public constructor, describe its parameters 10
3.6.4. For each public method of the class, add a row with the following columns:..... 10
4. OBJECT DIAGRAM - A SKETCH OR LIST OF OBJECTS AFTER DATA IS LOADED . 10
5. GUI LAYOUT SKETCH - AN IMAGE SAVED IN YOUR DESIGN DOCUMENT..... 12

1.Header

student name: Chuang, Yun-Shiuan (Sean)
project name: Best Investment Assistant for Joint Account Holders
project description:
Investment is a key to build up savings. Some investors use a joint investment account for various reasons like reducing account maintenance fee, fulfilling a minimum asset requirement. In such case, it is especially challenging for each individual to keep track of their assets because (1) each person's investment target may overlap with one another, (2) different investors' transactions on the same transaction target might have different unit cost, and (3) the dividends are given to the investors as a group rather than individually. Mainstream brokerage companies (e.g., Charles Schwab, TD Ameritrade) do not provide user interfaces for joint investors to view their individual investment records (e.g., current assets portfolio, change in assets).

With this program, the investors could readily (1) view their assets (e.g., current assets portfolio, return of investment, investment history), and (2) manage their investment accordingly (e.g., rebalancing).

2. Class Summary

enum, interface, class, abstract class	**Name of the type**	**Description of use or purpose of this type**
Class	Recorder	stores the transaction records and has methods that create summary reports for each individual investor.
Class	TransactionNode	stores the information of each transaction.
Class	Investor	stores each investor's information.
Class	InvestmentTarget	stores the information for each investment target
Class	FileReader	reads transaction record files that are downloaded from different brokerage companies
Class	Main extends Application	sets up and enables the GUI of the program

3. Class Diagrams

3.1. Class:Recorder (The class to contain the transaction records and have methods that create summary reports for each individual investor.)

3.1.1. Create a table or UML Class diagram showing the public fields and methods of that type

Recorder

-records:LinkedList<TransactionNode>
//a linked list for transactionNode
-tableInvestmentTarget:Hashtable<String,InvestmentTarget>
//the hash table with keys being the investment target name and values being the object InvestmentTarget (which contains the investment target information)
-tableInvestor:Hashtable<String,Investor>
// the hash table with keys being the investor's name and values being the object Investor (which contains the investor information)
-fileReader:FileReader
//the class that contains static methods for reading in external files

+Recorder() +importData(String):boolean +addTransaction(int,String,String,long,long):boolean +writeSummary(fileName):boolean +updateTargetInfo(String):boolean +setInvestorInfo(String, long):boolean +showTransaction():void +showTransaction(String):void

3.1.2. For each public field, describe its purpose and valid range of values

All fields are private.

3.1.3. For each public constructor, describe its parameters

+void Recorder():No args. //Construct an instance of an empty recorder. Users must use importData() and/or addTransaction() to add records.

3.1.4. For each public method of the class, add a row with the following columns:

return type	**the method name**	**parameter list**	**brief description of the method**
boolean //true if succeeds and false if fails	importData	String fileName	import the file of transactions that is downloaded from brokerage companies
boolean	addTransaction	int date,	manually add in one

//true if succeeds and false if fails		String investorName, String investmentType, String targetName long unitPrice long numUnits	transaction
boolean //true if succeeds and false if fails	writeSummary	String fileName	write the summary report for each investor to a file.
boolean //true if succeeds and false if fails	updateTargetInfo	String fileName	Update the information (e.g., current price, stock/bond type) of each investment target based on the information stored in an external file.
void	showTransaction	No args	Show the transaction records of all of the investors
void	showTransaction	String investorName	Show the transaction records of one investor

3.2. Class:TransactionNode (The class to store the information of each transaction.)

3.2.1. Create a table or UML Class diagram showing the public fields and methods of that type

TransactionNode

-next:TransactionNode
//point to the next node in the linked list
-date:long
//the date of the transaction
-investorName:String
//the name of the investor
-investmentType:String
//”sell”,”buy”,”devidend”
-target:String
//the name of the investment target
-unitPrice:long
//the unit price of the transaction
-numUnits:long
//the number of units of the transaction

+TransactionNode(TransactionNode, long, String, String, long, long)

3.2.2. For each public field, describe its purpose and valid range of values

All fields are private.

3.2.3. For each public constructor, describe its parameters

TransactionNode(TransactionNode, long, String, String, long, long):
- TransactionNode next //point to the next node in the linked list
-String investorName //the name of the investor
- long date //the date of the transaction
- String investmentType //”selling”,”buying”,”devidend”
- String target //the name of the investment target
- long unitPrice //the unit price of the transaction
-long numUnits //the number of units of the transaction

3.2.4. For each public method of the class, add a row with the following columns:

Only contains setters and getters.

3.3. Class:Investor (The class to store each investor’s

information.)

3.3.1. Create a table or UML Class diagram showing the public fields and methods of that type

Investor

-name:String
//the inverstor's name
-targetRatio:long
// the target stock/(stock+bond) ratio (the proportion of the stock value over the total value)
-currentPortfolio:Hashtable<String,long>
//the hash table with keys being the investment target name and values being the number of units that the investor has

+Investor(String, long)

3.3.2. For each public field, describe its purpose and valid range of values

All fields are private.

3.3.3. For each public constructor, describe its parameters

Investor(String, long):
-String name //the inverstor's name
-long targetRatio // the target stock/(stock+bond) ratio (the proportion of the stock value over the total value)

3.3.4. For each public method of the class, add a row with the following columns:

This class only contains setters and getters.

3.4. Class:InvestmentTarget (The class to store the

information of one investment target.)

3.4.1. Create a table or UML Class diagram showing the public fields and methods of that type

Investor

-name:String
//the target name
-currentPrice:long
// the target current price
-type:String
//e.g., ”stock”, “bond”

+InvestmentTarget(String, long, String)

3.4.2. For each public field, describe its purpose and valid range of values

All fields are private.

3.4.3. For each public constructor, describe its parameters

InvestmentTarget(String, long, String):
-String name //the target name
-long currentPrice // the target current pric
-String type //e.g., ”stock”, “bond”

3.4.1. For each public method of the class, add a row with the following columns:

This class only contains setters and getters.

3.5. Class:FileReader (The class to read external files)

3.5.1. Create a table or UML Class diagram showing the public fields and methods of that type

FileReader

None //Note that this class only contains static methods so no fields are required
-FileReader() +readTransactionFile(String,String):boolean +readTargetInfoFile(String):boolean +readInvestorInfoFile(String):boolean

3.5.2. For each public field, describe its purpose and valid range of values

No public fields for this class.

3.5.3. For each public constructor, describe its parameters

No public constructor. Note that this class only contains static methods so I want to enforce noninstantiability by using a private constructor

3.5.4. For each public method of the class, add a row with the following columns:

return type	**the method name**	**parameter list**	**brief description of the method**
boolean //true if succeeds and false if fails	readTransactionFile	String fileName String companyName	import the file of transactions that is downloaded from a specified brokerage company
boolean //true if succeeds and false if fails	readTargetInfoFile	String fileName	read a file that contains the investment target information (e.g., price, type)
boolean //true if succeeds and false if fails	readInvestorInfoFile	String fileName	read a file that contains the information of the investors (e.g., name, target stock/(stock+bond) ratio)

3.6. Class:Main (The class to set up and enable the GUI of the program)

3.6.1. Create a table or UML Class diagram showing the public fields and methods of that type

Main
-myRecorder:Recorder //the recorder that is working under the hood of the GUI
+start(Stage):void +main(String[]):void

3.6.2. For each public field, describe its purpose and valid range of values

No public fields for this class.

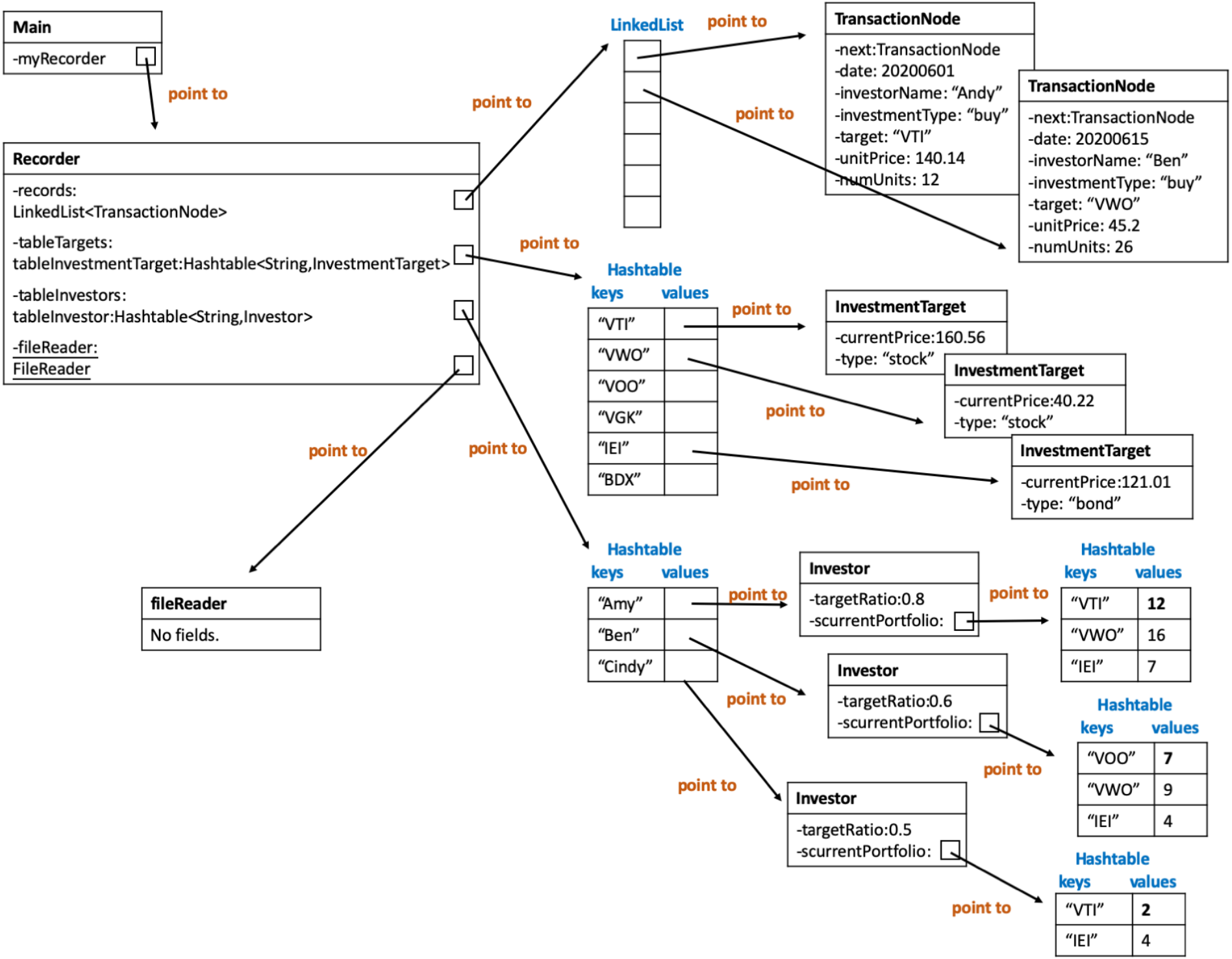
3.6.3. For each public constructor, describe its parameters

No public constructor is needed for this class.

3.6.4. For each public method of the class, add a row with the following columns:

return type	**the method name**	**parameter list**	**brief description of the method**
void	start	Stage primaryStage	set up and enable the GUI.
void	main	String[] args	launch the GUI

4. Object Diagram - a sketch or list of objects after data is loaded


Main
-myRecorder
point to
Recorder
-records:
LinkedList<TransactionNode>
-tableTargets:
tableInvestmentTarget:Hashtable<String,InvestmentTarget>
-tableInvestors:
tableInvestor:Hashtable<String,Investor>
-fileReader:
FileReader
point to
point to
point to
point to
fileReader
No fields.
LinkedList
point to
TransactionNode
-next:TransactionNode
-date: 20200601
-investorName: "Andy"
-investmentType: "buy"
-target: "VTI"
-unitPrice: 140.14
-numUnits: 12
point to
TransactionNode
-next:TransactionNode
-date: 20200615
-investorName: "Ben"
-investmentType: "buy"
-target: "VWO"
-unitPrice: 45.2
-numUnits: 26
Hashtable
keys values
"VTI"
"VWO"
"VOO"
"VGK"
"IEI"
"BDX"
point to
InvestmentTarget
-currentPrice:160.56
-type: "stock"
point to
InvestmentTarget
-currentPrice:40.22
-type: "stock"
point to
InvestmentTarget
-currentPrice:121.01
-type: "bond"
Hashtable
keys values
"Amy"
"Ben"
"Cindy"
point to
Investor
-targetRatio:0.8
-scurrentPortfolio:
point to
Hashtable
keys values
"VTI" 12
"VWO" 16
"IEI" 7
point to
Investor
-targetRatio:0.6
-scurrentPortfolio:
point to
Hashtable
keys values
"VOO" 7
"VWO" 9
"IEI" 4
point to
Investor
-targetRatio:0.5
-scurrentPortfolio:
point to
Hashtable
keys values
"VTI" 2
"IEI" 4

5. GUI Layout Sketch - an image saved in your design document