CSCE 156/156H
Assignment 3 - Project:
Summary & Detail Report

Spring 2022

1 Introduction

In the previous phase you began initial work to support an asset management system for IFFI. In this iteration, you wicontinue to design classes to support the application. In this phase, you will integrate all of your classes to produce two reports. The first will be a summary report that will report overall figures and totals for each account. The second will report the details of each individual account.

2 Data Files

The data files for persons and assets are attached. An additional data file is used that contains all information relating to accounts. As before, you may assume that all data is valid and properly formatted and all data files are named as specified.

2.1 Account Data File

Data for all accounts is contained in the file `data/Accounts.csv`. The first line contains a single integer indicating the number of records in the file. Each subsequent line contains the following data, separated by commas.

- Account Number – An alphanumeric identifier that uniquely identifies the account

- Type – the type of account, either `P` for Pro accounts or `N` for Noob accounts.
- Owner code – an alphanumeric code corresponding to the person who owns the account
- Manager code – an alphanumeric code corresponding to the person who manages the account
- Beneficiary code – an alphanumeric code corresponding to the person who is designated as the beneficiary. If there is no beneficiary, this will be an empty token.
- Asset list – the remaining data is a list of assets associated with the account. Depending on the type of asset there could be a different number of tokens. You will know the type of asset (property, crypto or stock) based on the first token which will correspond to the asset code identifier in the `Assets.csv` file (if no such entry exists, it should be considered bad data).
  - For **property** assets there will be two tokens tokens after the asset code: the purchase date in the format `2022-01-31` and the original purchase price of the property. For example, a property entry may look like:

    `PA005,2020-03-01,10500`

    The code, `PA005` corresponds to a Honda Fit in the `Assets.csv` file which was purchased on 2020-03-01 (March 1st, 2020) for $10,500.
  - For **cryptocurrency** assets there will be three tokens after the asset code: the purchase date in the format `2022-01-31`, the original exchange rate when purchased and the number of coins owned.

    For example, a cryptocurrency entry may look like:

    `BTC001,2015-03-01,5000,5.25`

    Which corresponds to Bitcoin (asset code `BTC001` in the `Assets.csv` file), for which 5,000 coins were purchased at $5.25 each on March 1st, 2015.
  - For **stock** assets, an account may own the stock outright or it may own either a put or a call option. To indicate which is the case, the second token (after the asset code) will be a `S`, `P` or `C`

    For example, if an entry looks like:

    `GOOG001,S,2020-01-01,928.52,150,0`

    The asset code, `GOOG001` corresponds to Google stock (in the `Assets.csv` file) and that the account owns shares outright. The remaining tokens are the purchase date, purchase share price, number of shares and dividend total. In this case, 150 shares of Google were purchased on January 1st, 2020 at $928.52 per share and the dividend total is $0.

    If an entry looks like:

    `GOOG001,C,2021-01-01,3200,100,10,2021-12-31`

We have the same asset, but this is a call option, the remaining tokens are the purchase date, strike price (per share), share limit premium (per share) and strike date. This example corresponds to a call option purchased on January 1st, 2021 with a strike price of $3,200 per share, a limit of 100 shares, a premium of $10 per share, and a strike date of December 31st, 2021.

If an entry looks like:

```
GOOG001,P,2021-01-01,3200,100,15,2021-12-31
```

We have the same asset, but this is a put option, the remaining tokens are the purchase date, strike price (per share), share limit premium (per share) and strike date. This example corresponds to a call option purchased on January 1st, 2021 with a strike price of $3,200 per share, a limit of 100 shares, a premium of $15 per share, and a strike date of December 31st, 2021.

An example was provided in the same zip file in the previous phase (along with expected output).

3 Requirements

You will be required to design Java classes to model the system and functionality as specified and hold the appropriate data. Which classes you implement and design, what you name them, and how you relate them to each other are design decisions that you must make. However, you should follow good object oriented programming principles. You should make sure that your classes are designed following best practices.

Your program will load the data from all the data files, construct (and relate) instances of your objects, and produce 1) a summary report, and 2) detailed report for each account. Both reports will be output to the standard output.

Formatting details are left up to you but your reports should be readable and communicate enough information to verify that your code is correct and that you've followed all requirements and specifications. An example has been provided (see the ZIP file and/or webgrader) but you are free to make it prettier, use alternate formatting, and are encouraged to communicate even more details if you wish.

A summary report should be printed to the standard output that should include the account number, owner, manager, and totals for fees, return, return percentage, and value. A grand total for fees, returns and asset values should also be reported over all accounts.

After the summary report, the details of each account should be printed. Again, the formatting is up to you, but it should report all information in a clear and human-readable format.

2 Data Files

Entity data is provided in several separate CSV files. A full example of well-formatted input and acceptable output has been provided. However, you will also be required to develop your *own* non-trivial test case. In general, you may assume that all data is valid and properly formatted and all data files are named as specified. You should assume that all data files are located in a directory called `data` and output files should be saved to the same directory.

2.1 Persons Data File

Data pertaining to individual people on the system is stored in a CSV file in `data/Persons.csv`. The format is as follows: the first line will contain a single integer indicating the total number of records. Each subsequent line contains comma delimited data fields:

- Person Code – a unique alpha-numeric designation for the person
- Name – the person's name in a `lastName,firstName` format
- Address – the mailing address of the person. The format is as follows: `STREET,CITY,STATE,ZIP,COUNTRY`
- Email Address(es) – an (optional) list of email addresses. If there are multiple email addresses, they will be delimited by a comma

2.2 Assets File

Data pertaining to assets is stored in a CSV file in `data/Assets.csv`. The format is as follows: the first line will contain the total number of records (an integer). Each subsequent line contains comma delimited data fields depending on the type of asset.

- Property has the format:
  `code,type,label,appraisedValue`
  where `type` will be `P`,
- Stocks have the following format:
  `code,type,label,symbol,sharePrice`
  where `type` will be `S`,
- Cryptocurrencies have the following format:
  `code,type,label,exchangeRate,exchangeFeeRate`
  where `type` will be `C`,

4 Design Process

You have some flexibility in how you design and implement this phase of the project. However, you must use good OOP practices. You should non-trivially demonstrate the proper use of the four major principles: inheritance, abstraction, encapsulation, and polymorphism. When thinking of your design, keep the following in mind.

- What should the public and private interface of each of the classes be? Don't make them simple data containers–what methods (behavior or services) should each class provide?
- Think about the state and methods that are common and/or dissimilar in each of your objects. What would be an appropriate use of inheritance and which methods/state are specific to subclasses? What, if anything should the subclasses define or override?
- What classes should own (via composition or some other method) instances of other classes? How are complex relationships made between objects?
- Think about how to utilize polymorphic behavior to simplify your code. You should not have to handle similar objects in a dissimilar manner if you have properly defined a common public interface.

Object-oriented design is fundamentally different from a top-down procedural style approach that you may be used to. Rather than breaking a problem down into sub-parts, we instead do a bottom-up design. We *first* identify the entities and design classes that can be used as the building blocks to implement a larger application. You are highly encouraged to completely understand the problem statement and have a good prototype design on paper before you write a single line of code.

5 Artifacts – What you need to hand in

- Your program *must* be runnable from a class named `AccountReport.java` which *must* be in the package `com.iffi`

6 Common Errors

Some common errors that students have made on this and similar assignments in the past that you should avoid:

- Design should come *first*–be sure to have a well-thought out design of your objects and how they relate and interact with each other *before* coding.
  - OOP requires more of a bottom-up design: your objects are your building blocks that you combine to create a program (this is in contrast with a procedural style which is top-down)
  - Worry about the design of objects before you worry about how they are created.
  - A good litmus test: if you were to delete your driver class, are your other objects still usable? Is it possible to port them over for some other use or another application and have them still work without knowledge of your driver program? If yes, then it is probably a good design; if no, then you may need to reconsider what you're doing.
- Objects should be created via a constructor (or some other pattern); an object should not be responsible for parsing data files or connecting to a database to build itself (a Factory pattern is much more appropriate for this kind of functionality).
- Classes (at least not all of them) should not be mere data containers: if there is some value of a class that depends on aggregating data based on its state, this should be done in some method, not done outside the class and a field set (violation of encapsulation–grouping of data with the methods that act on it). If a value is based on an object's state and that state is mutable, then the value should be recomputed based on the state it was in at the time that the value was asked for.

- Classes should be designed as stand-alone, reusable objects. Design them so that they could be used if the application was changed to read from a database (rather than a file) or used in a larger Graphical User Interface program, or some other completely different framework.

3 Business Rules

In general, every asset will have a current *value* and a *cost basis*. The cost basis is what the asset cost when it was acquired by the client. These values are calculated differently based on the type of asset and *may differ* from account to account. Full details on each are below, but for example, the value of Google stock will have the same cost-per-share, but one account may own 100 shares while another may have 200 shares and be twice as valuable.

In addition, the *gain* (or loss) of the value of an asset on an account is simply the difference between its cost basis and the current value. The *percentage gain* (or loss) is based on the percentage its value has changed with respect to the original cost basis:

$$\frac{\text{gain}}{\text{cost basis}}$$

In general, all dollar values are rounded to the nearest cent.

3.1 Property

The value of a property asset is based on its *appraised value*. When a property asset is associated with an account, it will also have

- A date of purchase
- Original purchase price (and thus the cost basis)

3.1.1 Examples

Suppose we have a Charizard Pokemon card that is currently appraised at $183,812. Now suppose that it is owned by a client who purchased it in 2001 for $62. The gain on this asset is huge:

$$183,812 - 62 = 183,750$$

and represents a percentage gain of

$$\frac{183,750}{62} = 296,370.97\%$$

As another example, suppose I bought a 1993 Ford Festiva in 1997 for $3,500$ and the current KBB appraised value for this wonderful vehicle is 150. This represents a "gain" (loss) of

$$150 - 3,500 = -3,350$$

and a percentage "gain" of

$$\frac{-3,350}{3,500} = -95.71\%$$

3.2 Cryptocurrency

The value of a cryptocurrency asset is always normalized by converting it to an "equivalent" amount of US dollars. This is done by taking the number of coins (which can be fractional) times the exchange rate. However, each cryptocurrency has an *exchange rate fee* (which is *not* included in the account fees) that needs to be taken into account if/when it is converted to US dollars.

When a cryptocurrency is in a client's account, it also has a purchase date and the exchange rate at which it was purchased (in order to compute the gain).

3.2.1 Examples

Suppose a client purchased 1.25 Bitcoin in July 2020 at $9,072 per coin (for a total of $11,340. Suppose the current value of Bitcoin is $57,930 per coin for an exchange value of $72,412.50. Now suppose there is an exchange rate fee of 1.1%, so the actual value of this asset is

$$1.25 \times 57,930 \times (1 - .011) = \$71,615.96$$

The gain of this asset is thus

$$\$71,615.96 - \$11,340 = \$60,275.96$$

with a percentage of

$$\frac{\$60,275.96}{\$11,340} = +531.53\%$$

3.3 Stocks

Stocks are shares in publicly traded companies. Every stock has a trading symbol and a (current) share price.

When stock is owned by a client, they own a certain number of shares. To compute the gain, the purchase share price is also tracked as well as the date it was purchased. Stocks also may pay the owner *dividends* (profits the company has made and paid to its owners). To keep it simple, the system only tracks the *total* number of dividends the owner has been paid (regardless of the number of shares or how long they have owned them).

3.3.1 Examples

Suppose a client owns 50 shares of GameStop stock (symbol: GME) with a current share price of $201.75. Suppose they bought the stock in July 2020 at $4.01 per share. Thus, the cost basis is

$$\$4.01 \times 50 = \$200.50$$

Further, suppose that the client has been paid a total of $228 in dividends since it was purchased. This makes its current value:

$$\$201.75 \times 50 + \$228 = \$43,728.00$$

with a gain of

$$\frac{\$43,728.00}{\$200.50} = +21,809.48\%$$

3.4 Options

Options (or stock options) are a financial contract the right to buy or sell a (up to) a certain number of shares of a stock (the underlying security) by a certain date. Options can get very complicated, but for simplicity, IFFI offers only two types: Calls and Puts. Both types are from the perspective of the client. Options provide a way for investors to speculate that the price of a certain stock will go up or down and to profit (or lose) depending on whether they are correct or not.[2]

- A *call* is an option for the client to *buy* a stock at a particular *strike price* per share. The client (the buyer of the option) pays the seller of the option a premium (per share) for this option.

  If the current per-share price is greater than the strike price, we assume that the option will be executed because the option allows the client to buy a stock at a discount which the client could then immediately turn around and sell at a profit.

  In reality, the client would need the capital to do so, but because options themselves can be traded (and are fungible), we can assume that the full value of the option is realized. In this case, the value of the of the option is the current share price minus the strike price. This is known as a *short call*.

  If, on the other hand, the current share price is less than or equal to the strike price, the client would *lose* money if they executed the option (because they could always buy the stock for less at the current market price than through the option). In this case, we assume that we will not execute the option and the value is zero. This is known as a *long call*.

[2] "Sounds to me like you guys are a couple of bookies." ... "I told you he'd understand!"

In either case, the premium we paid for the option was the purchase price/cost basis (and is already a sunken cost). The gain (or loss) is the same as other assets.

- A *Put* is an option for the client to *sell* a stock at a particular *strike price* per share. The client (the *seller* or "writer" of the option) is paid by the buyer of the option and is *obligated* to sell a certain number of shares of a given stock but only if the buyer of the option executes the option before the strike date.

  In reality, the client must own the underlying security or buy it at the current share price but only if the buyer executes the option.

  It is assumed that if the current share price is greater than the strike price the buyer *will* exercise the option because they'll realize a profit. In this case, the client will lose (some) money on the option. This is known as a *short put*. The value (to the client) of this option is thus a loss equal to the (strike price minus the share price plus the premium) times the number of shares.

  If, on the other hand, the current share price is less than the strike price, the buyer will *not* execute the option (they'd lose money). The value (to the client) is the premium already paid.

  Because the client is *selling* the option, there is no cost basis at all (the purchase price is zero). If there is a gain, the gain is total (+100%) and if there is a loss, the loss is total (-100%), otherwise it is 0%.

3.4.1 Examples

See Piazza post.

3.5 Account Types

IFFI has to types of accounts it offers clients, a basic "Noob" account and a more exclusive "Pro" account. Each account consists of any number of assets that are managed by IFFI. Each account is owned by a single person and managed by a single person. The owner may also *optionally* designate a beneficiary who will receive ownership of the account upon the owner's death. In addition, each account also has per-asset annual fees associated with it. However, Pro accounts give a 25% discount to the fee total; Noob accounts give no discounts.

3.6 Fees

To make money, IFFI charges annual fees for each account based on the number and type of assets managed.

- A $100 fee is assessed for *each* property regardless of its appraised value
- A $10 fee is assessed for each cryptocurrency asset regardless the number of coins owned by the client
- No fee is assessed for stocks or options