Assignment #3 [9 Marks]

Due Date	22 October 2022 - 4 November 2022 23:59
Course	**[M1522.000600] Computer Programming**
Instructor	Youngki Lee

- You can refer to the Internet or other materials to solve the assignment, but you ***SHOULD NOT*** discuss the question with anyone else and need to code **ALONE**.
- We will **use the automated copy detector to check the possible plagiarism** of the code between the students. The copy checker is reliable so that it is highly likely to mark a pair of code as the copy even though two students quickly discuss the idea without looking at each other's code. Of course, we will evaluate the similarity of a pair compared to the overall similarity for the entire class.
- We will do the manual inspection of the code. In case we doubt that the code may be written by someone else, we reserve the right to request an explanation about the code. We will ask detailed questions that cannot be answered if the code is not written by yourself.
- For the first event of plagiarism, you will get 0 marks for the specific assignment. You will fail the course and be reported to the dean if you copy others' code more than once.
- Download and unzip "HW3.zip" file from the autolab. "HW3.zip" file contains skeleton codes for Question 1 (in the "problem1" directory) and Question 2 (in the "problem2" directory).
- When you submit, compress the "HW3" directory which contains "problem1" and "problem2" directories in a single zip file named "20XX-XXXXX.zip" (your student ID) and upload it to autolab as you submit the solution for the HW1, HW2. Contact the TA if you are not sure how to submit. Double-check if your final zip file is properly submitted. You will get 0 marks for the wrong submission format.
- Do not modify the overall directory structure after unzipping the file, and fill in the code in appropriate files. It is okay to add new directories or files if needed.
- Java Collections Framework is allowed.
- Do not use any external libraries.
- If you have any questions, feel free to ask any in the Slack channel.

Contents

Submission Guidelines

1. You should submit your code to autolab.
2. After you extract the zip file, you must have a "HW3" directory. The submission directory structure should be as shown in the table below.
3. You can create additional directories or files in each "src" directory.
4. You can add additional methods or classes, but do not remove or change signatures of existing methods.
5. Compress the "HW3" directory and name the file "20XX-XXXXX.zip" (your student ID).

Submission Directory Structure (Directories or Files can be added)

- Inside the "HW3" directory, there should be "problem1" and "problem2" directories.

Directory Structure of Question 1	Directory Structure of Question 2

problem1/
└── src/
    └── Asset.java
    └── Buyer.java
    └── NFTMarket.java
    └── Pair.java
    └── Test.java
	problem2/
	└── src/
	    └── App.java
	    └── BackEnd.java
	    └── FrontEnd.java
	    └── Post.java
	    └── ServerResourceAccessible.java
	    └── Test.java
	    └── User.java
	    └── UserInterface.java
	    └── View.java

Question 1: NFT Market [4 Marks]

Objectives: Implement a market management application to register and trade assets.

Descriptions: You are a manager of an NFT market. You are asked to implement a system for maintaining the status of assets traded at the market. This application can add new assets, search for assets satisfying conditions, register buyers, and support the trade between buyers.

You will need to implement various methods in the NFTMarket.java, Asset.java, and Buyer.java classes in src directory as instructed below.

- Asset and Buyer classes are provided in Asset.java and Buyer.java, respectively; you **can** change these classes if necessary.
- Test cases are introduced in Test.java. Note: we provide the basic test cases and we will use a richer set of test cases for evaluation. We strongly encourage you to add more diverse test cases to make sure your application works as expected.

Question 1-1: Basic Methods for NFTMarket Class [0.5 Marks]

Objectives: Implement the following methods of the NFTMarket class: addAsset, findAsset addBuyer, and findBuyer.

Descriptions: The NFTMarket class manages the information of all the assets and buyers. As a start, you will implement the following four basic methods to add/find assets/buyers inside the NFTMarket class. To implement these methods, you will use Map to store Asset objects and Buyer objects (i.e., idAsset and nameBuyer).

- public boolean addAsset(int id, String item, float price, String artist) creates an Asset object with the id, item, price, and artist and saves it in the corresponding collection.
    - The id of an asset should be unique. If the Asset object with the same id was already registered, do not create an Asset object and return false. Also, the id should have a non-negative value. If a negative id is given, return false.
    - For the initial addition of assets, set the master as the buyer of assets. The master is the given Buyer class attribute of the NFTMarket class. The master manages the initial assets and can trade them with other buyers.
    - The price of an asset, price, is in the range of [0, 100000]. If the given price is out of this range, do not create an asset object and return false.
    - Return true if a new Asset instance is created and stored successfully.

do i have to change it to:
if count = 1, do: ?

- public boolean addBuyer(String buyername) creates a Buyer object with the buyername and saves it in the corresponding collection.
    - If there already exists a buyer with the given buyername, do not create a Buyer object, and return false. If a null or empty string is given for the buyername, return false.
    - Initialize the buyer with the balance of 100000 (or use the existing constructor).

    - Return true if a new Buyer object is created and stored successfully.
- public Asset findAsset(int id) returns an Asset object with the given id.
    - Return null if there is no asset with the given id.
- public Buyer findBuyer(String buyername) returns a Buyer object with the given buyername.
    - Return null if there is no Buyer with the given buyername.

Question 1-2: Asset Searching with Conditions [1 Mark]

Objectives: Implement the findAssetsWithConditions method of the NFTMarket class.

Descriptions: Implement the method to search for the assets matching the given conditions.

- Implement the public List<Asset> findAssetsWithConditions(int minprice, int maxprice, String item, String artist) method.
- The method returns a List of Asset objects matching **all** of the given conditions.
    - A price condition is given as follows. Filter only assets of which price is in the range of [minprice, maxprice]. If the user wants to skip this condition, wildcard option can be used: both minprice = -1 and maxprice = -1. If a wildcard is used on only one side (e.g. minprice = -1 and maxprice = 5000), return an empty list. same as null?
    - An item and artist conditions select assets based on asset name and artist, respectively. Check names case sensitively. Also, the wildcard option can be used in both cases, and the keyword of the wildcard is **All** in string.
    - If there is no asset matching all these conditions, return an empty list.
    - The output list should be sorted in an ascending order of the asset id.
        - e.g. [10, 300, 5000, 5100]

Question 1-3: Asset Trading [1.5 Mark]

Objectives: Implement the trade method of the NFTMarket class. You may want to change and use getAssetPortion method in Buyer class.

Descriptions: You will now implement a method to trade an asset between two buyers.

- public boolean trade(Buyer seller, Buyer client, int id, float portion)
    - This method implements asset trading between two buyers. Note that it can only be used when an asset has at least one owner. (e.g. owner: seller)
    - The seller sells the given portion of his partial (or whole) asset to the client. The Buyer class manages its share of assets using its portfolio.
    - Do nothing and return false in one of the following cases.
        - The asset with id does not exist.
        - The seller or client is given null.
        - The asset is not owned by the seller.

        - The client cannot buy this asset due to its insufficient balance.
    - Otherwise, return true with the following modifications.
        - If the client does not have the asset, add the asset to the portfolio of the client and add the client to the owner list of the asset. If the client already has the asset, update the portion of the asset one holds.
        - If the seller sold all of one's portion, remove the asset from the portfolio of the seller, and update the owner list of the asset.
        - Adjust the balances of the seller and the client.

Question 1-4: Market Fluctuation [1 Marks]

Objectives: Implement the two reflectIssues methods of the NFTMarket class and the getTotalValue method of the Buyer class.

Descriptions: You will implement methods to reflect the value change of an asset, or value changes of assets by a given artist.

- public void reflectIssues(Asset asset, float effectFactor)
    - This method reflects the effect of the issue on an asset. The price value of an asset changes to the current value multiplied by a given effect factor. Note that the total values of buyers who own the asset should be updated.
    - An effectFactor is given as a non-negative float.
- public void reflectIssues(String artistName, float effectFactor)
    - This method reflects the effect of the issue on an artist. The values of assets made by the artist change to the current values multiplied by a given effect factor. Note that the total values of buyers who own the assets by the given artist should be updated.
    - An effectFactor is given as a non-negative float.

what do i update…?

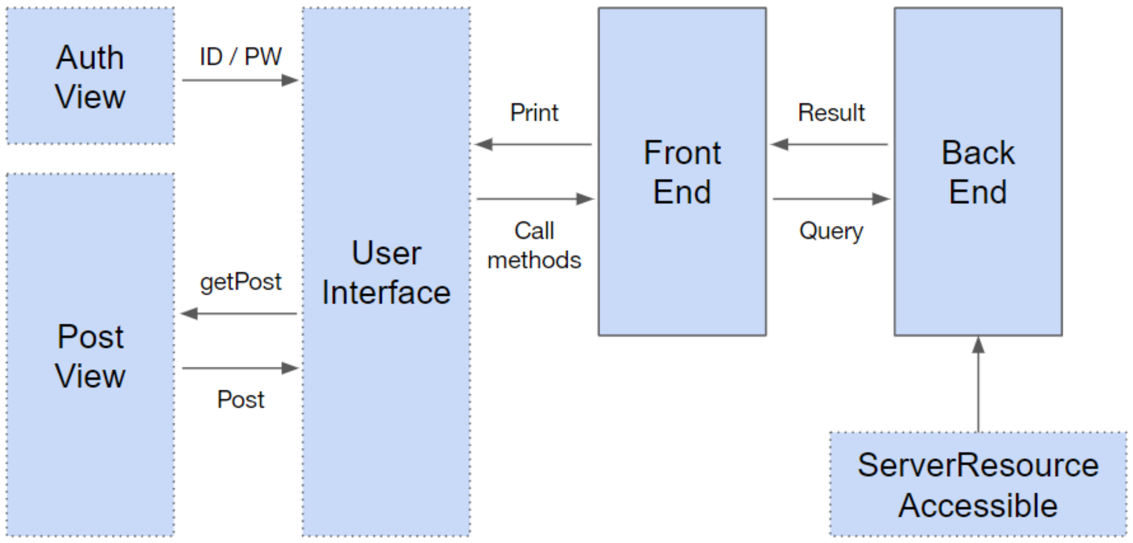
- public float getTotalValue() returns the total value a Buyer holds containing the balance and the values of assets in his or her portfolio.

Question 2: Social Network Service (SNS) [5 Marks]

Objectives: Implement a simple Social Network Service (SNS).

Descriptions: We are going to build a simple console-based SNS system. A user can write posts, search them, and view the recommended friends' posts.

Before you jump into the implementations, you may first want to understand the overall structure of this application. The below figure shows the overall architecture of our SNS application.



The application mainly consists of 3 classes: UserInterface, FrontEnd, and BackEnd (along with a few additional classes supporting these main classes).

- The UserInterface class provides interfaces to users.
  - It does the authentication at the beginning of the application through the AuthView class.
  - After the login, the PostView class gets a command input from the user by String getUserInput(String prompt) method and calls the corresponding method of the FrontEnd class.
  - The results of the commands, provided by the FrontEnd class **should be displayed using the print methods** of UserInterface in an appropriate form.
    - UserInterface.println : Equivalent to System.out.println
    - UserInterface.print : Equivalent to System.out.print
- The FrontEnd class parses user commands/posts, and queries the BackEnd class (by calling the appropriate methods) to process them.
- The BackEnd class receives the queries from the FrontEnd class, processes them, and returns the results.
  - The BackEnd class inherits the ServerResourceAccessible class.

    - The ServerResourceAccessible class has the member attribute named serverStorageDir, which specifies the path to the data directory. All the necessary data (e.g., user information, friends information, user posts) are stored as files under this directory. More details will be explained later.
    - Note that the serverStorageDir value ends with “/”.
    - In subsequent descriptions, we will describe the data directory path as the $(DATA_DIRECTORY).
    - In the given skeleton, the data directory is set to “data/”, but it may be changed to another path in the final testing. Do not assume that the data directory is always “data/”. Use the serverStorageDir attribute to retrieve the data path.
- The program should do nothing in case of any exception (e.g., non-existence of the user id).
- Default test cases specified in this document are provided in Test.java. You can compare your result from Test.java with the expected outputs in this document. Note: we provide the basic test cases, and we will use a richer set of test cases for evaluation. We strongly encourage you to add more diverse test cases to make sure your application works as expected.

Great! You are now ready for the implementation of the cool SNS application.

- You need to change the FrontEnd and BackEnd classes to solve this problem. Please do not modify the rest of the classes.
- Do not modify the signatures of the FrontEnd class methods since we will use them for the final testing.
- There are 4 subproblems that are meant to be solved in order. If you proceed without solving the earlier problems, it may affect the later problems.

Question 2-1: Authenticate [1 Mark]

Objective: Implement the method public boolean auth(String authInfo) of the FrontEnd class to authenticate the user. You may need to make appropriate changes to other parts of the FrontEnd and BackEnd classes as well. In particular, it compares the input password with the password stored in the server to check its validity.

Description: Upon the program start, the console will ask for the user id and the password. In the given skeleton code, a user cannot log in even with a valid password. Now, we want to change it so that login can be possible with a valid password.

- The password of a user is stored at the path $(DATA_DIRECTORY)/(User ID)/password.txt; here, the (User ID) is the id of the user.
- Assume that all the names of the direct child directories of $(DATA_DIRECTORY) are valid user ids.

- Assume that every $(DATA_DIRECTORY)/(User ID) has a password.txt.The format of the password.txt is given in the following example (plain text without a newline). Suppose the password of the user 'root' is 'pivot@eee'.

[File Format] $(DATA_DIRECTORY)/root/password.txt
pivot@eee

- For successful authentication, the input password and the stored password should be **identical** including white spaces.
- If the login fails, the program terminates.

Note that text in blue in the following example indicates the user input:

Console prompt (Login in this case)

```
------ Authentication ------
id : root
passwd : pivot@eee
----------------------------------
root@sns.com
post : Post contents
recommend <number> : recommend <number> interesting posts
search <keyword> : List post entries whose contents contain <keyword>
exit : Terminate this program
----------------------------------
Command :
```

Console prompt (Fails login in this case)

```
------ Authentication ------
id : root
passwd : admin2
Failed Authentication.
```

Question 2-2: Post a User Article [1 Mark]

Objective: Implement the `public void post(List<String> titleContentList)` method of the `FrontEnd` class to store the written post in the server. You may need to make appropriate changes to other parts of the `FrontEnd` and `BackEnd` classes as well.

Description: When a user types in the "post" command to the console, he or she can start writing a post with the 1) title, 2) whether the post is for advertising or not, and 3) content. The content of the post ends when the user inputs "Enter"(\n) twice.

- Store the user's post at the path $(DATA_DIRECTORY)/(User ID)/post/(Post ID).txt. (User ID) is the user's id used for the login, and the (Post ID) is the nonnegative integer assigned uniquely to each and every post in the $(DATA_DIRECTORY). The newly assigned ID should be **1 + the largest post id in the entire posts in $(DATA_DIRECTORY)** or 0 in a case when $(DATA_DIRECTORY)/(User ID)/post/ is empty.
- Assume all the $(DATA_DIRECTORY)/(User ID) has a directory named post, and each post directory has at least one post.
- The format of the post file is given in the examples below.
- The variable "Advertising" is assigned with "yes" or "no".
- The content of the post should not include the trailing empty line.
- The stored post should be initialized and stored with the like-number of 0 (in the form of "like-number 0", refer to the below example.)
- Hint: use the `LocalDateTime` class, and `LocalDateTime.now()` to get the current date and time. Refer to the `Post` class implementation to format the date and time.

Let the user name be 'root', and the largest post id in the entire $(DATA_DIRECTORY) be 399. Then, the new post id is 399 + 1 = 400. Let the post date be 2022/08/29 22:32:00.

Console Prompt

```
Command : post
-----------------------------------
New Post
* Title : my name is
* Advertising (yes/no) : yes
* Content :
> root and
> Nice to meet you!
>
-----------------------------------
```

Then the post is saved to "$(DATA_DIRECTORY)/root/post/400.txt", as shown below. The date, title and advertising input fields are written in each new line, respectively. There is an **empty line after the like number status**, and finally, the content is written.

[File Format] $(DATA_DIRECTORY)/root/post/400.txt

```
2022/08/29 22:32:00
my name is
yes
like-number 0

```

```
root and
Nice to meet you!
```

Question 2-3: Recommend Friends' Posts [1 Mark]

Objective: Implement the public void recommend(int N) method of the FrontEnd class to print the "advertised" and latest posts of the user's friends. You may need to make appropriate changes to other parts of the FrontEnd and BackEnd classes as well.

Description: Our SNS service recommends a user the latest posts of his or her friends. When the user inputs the "recommend <N>" command to the console, up to N latest posts of the friends should be displayed. However, an advertisement post has a higher priority than the latest posts.

- The list of the user's friends is stored at the path "$(DATA_DIRECTORY)/(User ID)/friend.txt". The format of the friend.txt is given as the following example. Suppose the user "root" has 3 friends, "admin", "redis", and "temp".
- You need to look at all the posts of the friends and print up to N posts prioritized by the criteria of 1) advertising and 2) latest creation date.
- How do we decide which N posts to recommend?
  - First, make two types of lists from all the posts of the friends by the advertisement: 1) advertised, and 2) non-advertised.
  - Sort both types of posts by the creation date specified in a post file in descending order (from latest to oldest).
  - Select the first N posts from the sorted list of **advertised posts**. If the number of advertising posts are less than N, select all of them.
  - If not select the N posts yet, select rest from the non-advertised posts.
  - Recommend the selected posts.
- Assume there is no concurrency issue regarding the creation time. Creation time of each post is unique.
- Assume all the friend ids on the friend.txt are valid, and the corresponding folders exist in the $(DATA_DIRECTORY).
- The post should be printed as a format of the toString method of the Post class with the UserInterface.println method, not the getSummary method.

[File Format] $(DATA_DIRECTORY)/root/friend.txt
admin redis temp

In the example below, the command ‘recommend’ displays 1 latest posts of “admin”, “redis”, “temp” users.

Console Prompt (Authenticated with ‘root’)

```
Command : recommend 1
-----------------------------------
id: 314
created at: 2019/02/20 00:00:00
title: then caviar alert are is
content: buteo
then am are cornish is he
he my my am a the adroit are he are you is you he dreamless
where you chevalier then is where her my then the you you creation the cynocephalidae are
why biscuits
alexandria auriga befoul derived cellulosid duet concious dachau disinterestedness essaying
bh anxiousseat acrocarp dividers coleonyx advance(a) dolphin constipate agog deprivation
carambola displeasure exploit attuned assistive baccharis barnacled aegyptopithecus
amplification boogie
bougie eau cresol cajun curmudgeon acrocephalus cocktail catechism bugle carvel-built
coiffure anergy endodontics effluxion expectant bassariscidae curassow alter banister cinclus
apodeme blackhead cocoa chaetal contextual araba amaryllidaceae deceleration attribution
breathe
beaucoup ablepsia coverture beryl cucurbita betula associative accepting catechistic
euproctis
adelomorphous balldress brand-newness catacomb basidiolichen brunt carpet attainder crape
centerpiece
deluge arboraceous agouti chaffe dualistic adherent bolt-hole bannerlike absento contra
badli anguidae embalmer effort anneal drinkable annunciation celestite entice coverall
asio bullfighter boiler chirrup dreggy banzaijapanese appointments eutamias contuse burgle
amaryllidaceae exceptional canape communist cunningman carver abdication bodybuilding
chigoe amid
diapensiaceae aneurysm carnosaur crackajack campaign enough affluxion competition beg
crutched
coffeeberry cogitate blinds crossroads disfigurement abscision ditch diluent devote
androgenesis
dilemma danae clathrus displeasure dissembler contortion bushes coup counterscarp bravo
cancellation eventual droit broker denigration cowering(a) ardently diclinous chaenactis
anglophile
...
```

Question 2-4: Search Posts [1 Mark]

Objective: Implement the public void search(String command) method of the FrontEnd class to display up to 10 posts that contain keywords in the frequency that is greater than or equal to the frequency threshold. You may need to make appropriate changes to other parts of the FrontEnd and BackEnd classes as well.

Description: Our SNS service enables users to search for posts with multiple keywords. When the user inputs the "search" command along with a set of keywords, the console should display up to 10 posts considering the keywords and likes by the users.

- The command string starts with "search" followed by keywords.
- Keywords are separated with space(' '). The newline should not be considered as a keyword.
- The search command should include at least one keyword. If it does not, print "Wrong Command."
- **The last word** of the command string should be an integer representing the frequency threshold of keywords for the target search. If it is not, print "Wrong Command."
- Duplicate keywords should be ignored. For example, the output of search hi hi 5 should be identical to the output of search hi 5.
- The range of the search is the entire posts of all users (NOT friends only) in the $(DATA_DIRECTORY).
- The list of the user's liked-posts is stored at the path "$(DATA_DIRECTORY)/(User ID)/likedposts.txt". The format of the likedposts.txt is given as the following example. In each line, the name of the author and the id of each article is written. And they are separated by one space. (ex. temp 313 323 320 319 305 314)
- You should count the number of occurrences of the **exact** keyword from the title and the content of the post.
- More specific details for the keyword matching:
  - It should be a case-sensitive comparison.
  - You should only count the word that is identical to the provided keyword. You don't need to consider the word that has the given keyword as a substring.
- How do we decide which posts to show?
  - Filter the searched posts with the given keyword threshold number.
  - Sort the candidate posts based on two criteria. Show the user-liked posts in descending order of like-numbers. If the like-numbers are the same, show the recent post first. Show the non-user-liked posts in descending order of like-numbers. If the like-numbers are the same, show the recent post first.
  - Select up to 10 posts from the beginning of the sorted list.
- The posts should be displayed using the format provided by the getSummary method of the Post class, not the toString method.

Console Prompt

```
Command : search classical 1
id: 389, liked 86, created at: 2007/02/13 00:00:00, title: cleveland adapa her boredom you
id: 393, liked 35, created at: 1973/11/26 00:00:00, title: ciliated why daffaire blepharitis are
id: 308, liked 87, created at: 2005/01/22 00:00:00, title: damore is is are cancerous
id: 309, liked 83, created at: 2008/03/26 00:00:00, title: boltonia my are derris canard
id: 202, liked 82, created at: 2009/10/23 00:00:00, title: then her christmas clodpoll
arthritic
id: 221, liked 73, created at: 2007/08/17 00:00:00, title: draft barbital comedietta then is
id: 210, liked 68, created at: 1983/01/28 00:00:00, title: emberizidae carburetor
democritus he bucolic
id: 231, liked 66, created at: 1994/02/14 00:00:00, title: disruptively the my her crocheting
id: 280, liked 52, created at: 2017/09/05 00:00:00, title: abolengo then delinquent bubalus
barracker
id: 296, liked 32, created at: 2001/09/16 00:00:00, title: creativeness coruscate am
adulterated competition
```