AY 2020 Assignment 3 [9 Marks]

Date / Time	16 October 2020 – 30 October 2020 23:59
Course	**[M1522.600] Computer Programming**
Instructor	Youngki Lee

- You can refer to the Internet or other materials to solve the assignment, but you ***SHOULD NOT*** discuss the Problem 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 (outside of the class), we reserve the right to request an explanation about the code. We will ask detailed Problems that cannot be answered if the code is not written by yourself.
- If one of the above cases happen, you will get 0 marks for the assignment and may get a further penalty. Please understand that we will apply these methods for the fairness of the assignment.

- Download and unzip “HW3.zip” file from the eTL. “HW3.zip” file contains skeleton codes for Problem 1 and Problem 2 (in the “problem1”, “problem2” directory).
- 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.
- When you submit, compress the “HW3” directory which contains “problem1” and “problem2” directories in a single zip file named “20XX-XXXXX.zip” and upload it to ETL as you submit the solution for the lab tests. 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 use external libraries. If you are unsure, contact TAs to clarify.
- Java Collections Framework is allowed.

Contents

Submission Guidelines

1. You should submit your code on eTL.
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).
6. The Autolab server for Assignment 3 will be available in a week.

Submission Directory Structure (Directories or Files can be added)

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

Directory Structure of Problem 1	Directory Structure of Problem 2

problem1/
└── src/
  └── Movie.java
  └── MovieApp.java
  └── Test.java
  └── User.java
	problem2/
	└── src/
	  └── App.java
	  └── BackEnd.java
	  └── FrontEnd.java
	  └── Post.java
	  └── ServerResourceAccessible.java
	  └── Test.java
	  └── User.java
	  └── UserInterface.java
	  └── View.java

Question 1: Movie Application [4 Marks]

Objectives: Implement a movie application to search, rate, and recommend movies.

Descriptions: You are a senior developer in an entertainment content provider, Netcha. You are asked to implement a system for users to search for movies, rate the movies, and get recommendations. In particular, you will need to implement various methods in the MovieApp class in MovieApp.java as instructed below.

- User and Movie classes are provided in User.java and Movie.java, respectively; you **can** change these classes if necessary.
- Test cases are introduced in the Test.java. Note: we provide the basic test cases for the Autolab, 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 MovieApp Class [1 Mark]

Objectives: Implement the addMovie, addUser, findMovie, and findUser methods of the MovieApp class.

Descriptions: The MovieApp class manages the information of all movies and users. As a start, you will implement the following four basic methods to add/find movies/users inside the MovieApp class.

To implement these methods, you will need to decide on the appropriate collections (e.g., List, Set, Map, etc.) to store Movie objects and User objects, and add a few lines of code to declare and initialize the collections.

- public boolean addMovie(String title, String[] tags) creates a Movie object with the title and tags and saves it in the collection.
    - If the Movie object with the same title was already registered, do not create a Movie object and return false.
    - Return true if a new Movie object is created and stored successfully.
- public boolean addUser(String name) creates a User object with the username and saves it in the collection.
    - If there already exists a user with the given username, do not create a User object, and return false.
    - Return true if a new User object is created and stored successfully.
- public Movie findMovie(String title) returns a Movie object with the given title.
    - Return null if there is no movie with the given title.
- public User findUser(String username) returns a User object with the given username.
    - Return null if there is no user with the given username.

Question 1-2: Movie Searching with Tags [1 Mark]

Objectives: Implement the `findMoviesWithTags` method of the `MovieApp` class.

Descriptions: Implement the method to search for the movies matching the given tags.

- Implement the `public List<Movie> findMoviesWithTags(String[] tags)` method.
- The method returns a `List` of `Movie` objects matching **all** given `tags`.
    - If the given `tags` are a subset of the tags of a movie, the movie should be included in the output list. Otherwise, the move should not be included.
    - There should be no duplicated movies in the output list.
    - The output list should be in the lexicographical order of the title of a movie.
        - e.g. [“Abatar”, “Heart Locker”, “Jocker”, “Ra Ra Land”]
    - If the given `tags` match none of the movies, the empty list should be returned.
    - If the empty `String` array is given as a `tags` argument, the empty list should be returned.

Question 1-3: Movie Rating [1 Mark]

Objectives: Implement the `rateMovie`, `getUserRating` methods of the `MovieApp` class.

Descriptions: You will now implement two methods: (1) to rate a movie and (2) to get a rating. Note that these two methods will be used in Question 1-4.

- `public boolean rateMovie(User user, String title, int rating)`
    - This method adds a rating of a movie from a user. Note that it is necessary to store which movie was rated, who rated the movie, along with the rating score. Decide on an appropriate collection to store the rating information.
    - Do nothing and return false in one of the following cases.
        - The title is null, or there is no movie with the given title.
        - The user is null or not registered.
        - The given rating is out of range. The valid rating should be between 1 <= (rating) <= 10.
    - Otherwise, store the rating information and return true.
    - If a user rates the same movie multiple times, only the last rating is stored.
- `public int getUserRating(User user, String title)`
    - This method returns the rating of the user for the movie with the given title.
    - If the user or the movie with the title is null or not registered, return -1,
    - If the user has not rated the movie, return 0.

Question 1-4: Movie Recommendation [1 Mark]

Objectives: Implement the `recommend` and `findUserMoviesWithTags` methods of the `MovieApp` class.

Descriptions: You will implement a method to recommend movies using a user's search history.

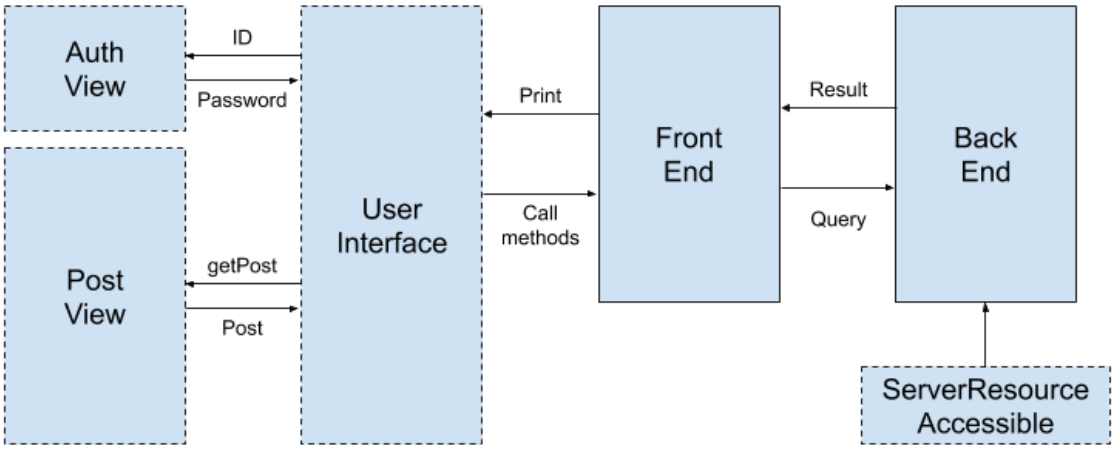
- As the first step, you need to implement the `public List<Movie> findUserMoviesWithTags(User user, String[] tags)` method to store the search history of a user. The search history is used for the recommendation.
    - This method does two tasks: i) stores the search history (which will be used by the `recommend` method described below), and ii) returns the list of movies matching the given tags.
    - If the `user` is null or not registered, do nothing and return the empty list.
    - Otherwise, i) use the `findMoviesWithTags` method (implemented in Question 1-2) to find matching movies, ii) store the list of searched movies along with the user information, and ii) return the matching movies.
- Then, you need to Implement the `public List<Movie> recommend(User user)` method that returns 3 movies to recommend to the given user.
    - How do we decide those 3 movies to recommend?
        - First, the candidates are the outputs of the `findUserMoviesWithTags` the user has called.
        - Sort the candidate movies based on two criteria. First, sort by the average ratings (descending order). When multiple movies have the same average ratings, sort them by lexicographical order of the titles. Recommends the first three movies and returns them as a list of the Movies objects.
    - If the `user` is null or not registered, return the empty list.
    - If the movie is not rated before, its average rating is 0.
    - If there are less than 3 movies to recommend, then the returned list may contain less than 3 movies. If there is no movie to recommend, return the empty list.

Question 2: 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 for the Autolab, 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 influence 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 the 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 red 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 : recommend 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.

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

Objective: Implement the `public void post(Pair<String, String> titleContentPair)` 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 inputs the “post” command to the console, he can start writing a post with the title and content. The content of the post ends when the user inputs “Enter” 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 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 content of the post should not include the trailing empty line.
- 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 302. Then, the new post id is 302 + 1 = 303. Let the post date is 2019/12/11 21:01:02.

Console Prompt

```
Command : post
----------------------------------
New Post
* Title : my name is
* Content :
> root and
> Nice to meet you!
>
----------------------------------
```

Then the post is saved to "$(DATA_DIRECTORY)/root/post/303.txt", as shown below. The date and title are written in each new line, respectively. There is an empty line after the title, and finally, the content is written.

[File Format] $(DATA_DIRECTORY)/root/post/303.txt

```
2019/12/11 21:01:02
my name is

root and
Nice to meet you!
```

Problem 2-3: Recommend Friends' Posts [1.5 Mark]

Objective: Implement the `public void recommend()` method of the `FrontEnd` class to print the 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 her friends. When the user inputs the "recommend" command to the console, up to 10 latest posts of the friends should be displayed.

- 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 "remp".
- You need to look at all the posts of the friends and print up to 10 posts with the latest created dates.
- How do we decide those 10 posts to recommend?
    - Sort posts by the created date specified in a post file in descending order (from latest to oldest).
    - Select the first 10 posts from the sorted list.
- Assume the created date and time of each post is unique. No two posts have the same created date and time.
- To compare two `LocalDateTime` class instances, use the `isAfter`, `isBefore`, `isEqual`, `compareTo` methods of the `LocalDateTime` class. Additional details are available in the link.
- 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, not the `getSummary` method, with the `UserInterface.println` method.

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

In the example below, the command ‘recommend’ displays 10 latest posts of “admin”, “redis”, “remp” users.

Console Prompt (Authenticated with ‘root’)
Command : recommend ---------- id: 330 created at: 2019/08/13 00:00:00 title: her corespondent conscienceless cobitidae aneurysmal content: my where agonized why he are a balistes why detort her then aeronaut antithetically brachyurous a birdseye my where delegacy her carbonic the beslubber am why are egoism why where then is then awl claudius her decretive a am are ---------- id: 340 created at: 2019/03/01 00:00:00 title: her cryolite chemin allegro my content: euphractus deliberative is cockatoo a a are are why he a averseness you where you depletable a why barbacan a are why are is alauda is doubts are anguish is where celtuce then eram are her are are coccidioidomycosis am ---------- 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
---
id: 47
created at: 2019/01/06 00:00:00
title: consecutio escrapment connaturalness you aggregation
content: a a
a you
encratism are my you
her auriform then
the
courtelle competently my her a coiffure
are is archosauria brazenfaced delire a are then burglarious dispersive distingue he babble a her are where
are where he estimated my
---
id: 318
created at: 2018/12/19 00:00:00
title: erosion clown dixit bloodwort a
content: the corpse experimentist dissolving then is her convector are her you corrected ended befog her is conceit ethology where cellulose her is are why you alfardaws are is cowering(a) am a central-fire
why desperandum he
why you am her her
---
id: 143
created at: 2018/08/03 00:00:00
title: are bidder concertina est co
content: beechnut he he then correspondingly then the why my the appurtenance cosmic am he
am you am my
the am ardeidae you aduncity the are enthusiastic am
elaeocarpus you my why aphonia her you dozen a her are ell disbursement
---
id: 119
created at: 2018/04/21 00:00:00
title: then you congress then blae
content: where why disdainfully a
credo crossheading where her you is he are am avowed(a) eighth my
are bundle
bedroll begrime the my eram her casing why my why cufflink my are her canonization aegyptopithecus amidships where

is is disdainful appurtenances
---
id: 60
created at: 2018/02/14 00:00:00
title: currency he am aphaeretic abutting
content: you drepanididae am are are my crunch is bottle where dachau is is where croisis then her my my where enslave beakless
the you concious capnomancy why my a are why bicarbonate a my dissever where a
then chemakum where
---
id: 53
created at: 2018/01/30 00:00:00
title: where absent attractive(a) erudition where
content: my he then then he am then animate electroscope where am you argillite why the he enormous my aluminum where my am a bombus bottomless
then the announce the enamelist her then he my apodal why a a
bigeminal er
---
id: 326
created at: 2017/10/07 00:00:00
title: expiration emotion he empressment a
content: why my bisected ectodermal dry-cleaned my
am you you is the then
am he
then am conclusiveness
etat he distraction centrist disband dogs my why where bronzed
why am why arteriosclerotic her
called my a cpoetry am why attaghan you

Problem 2-4: Search Posts [1.5 Mark]

Objective: Implement the `public void search(String command)` method of the `FrontEnd` class to display up to 10 posts that contain at least one keyword. 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 containing the most number of keywords in descending order of the created date and time.

- The range of the search is the entire posts of all users (NOT friends only) in the $(DATA_DIRECTORY).
- The command string starts with "search" followed by keywords.
- Two keywords are separated with space(' '). The newline should not be considered as a keyword.
- Duplicate keywords should be ignored. For example, the output of `search hi hi` should be identical to the output of `search hi`.
- 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 those posts to show?
    - Sort the candidate posts based on two criteria. First, sort by the number of occurrences of the keywords in descending order. When multiple posts have the same number of occurrences, sort them by the created date and time of the post in descending order (from latest to oldest).
    - 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 centavo id: 169, created at: 2001/10/08 00:00:00, title: centavo you you carpellate chives id: 2, created at: 1977/08/18 00:00:00, title: a damocles why eurypterid you