Project 3: TextBook

CS121 Computer Science I

Overview

TextBook is a super-retro, text-only, one-user-at-a-time, console-based social media platform. This application allows the current user to add posts, remove posts, comment on posts, display the list of all posts, and select a post to read with all its associated comments. When updates are made, posts and comments are written in files, so the complete set of all posts and comments can be recovered (reloaded from files) the next time the program is run.

When the user runs the program, they will first be asked for their username. Then they will be greeted and presented with a menu. From here users will be able to add new posts, comment on existing posts, delete posts, etc. Below is an example of what the beginning of the program will look like for the user.

```
Enter your name to enter TextBook: PeterPiper
PeterPiper, welcome to TextBook, the totally-text social media site!
---------------------------------------
TextBook Site Menu
---------------------------------------
        (P)rint TextBook posts
        (A)dd a new post
        (D)elete a post
        (C)omment on a post
        (R)ead a post and its comments
        (Q)uit
---------------------------------------
---------------------------------------
Select an option or M for menu:
```

You will write three classes: `Post.java`, `TextBook.java`, and `TextBookDriver.java`. Pay close attention to the details given for each class in the project specifications below. As you are developing your code keep in mind how to break the overall project into smaller pieces.

Objectives

- Develop a program that consists of multiple custom classes that enforce encapsulation, utilize instance variables including collections, and implement multiple methods.
- Use interfaces to develop and test classes with a predefined API.
- Read and write files.

Getting Started

- Create a new Java project in VS Code named "TextBook"
- Download the project support files and import them into the project.
    - `PostInterface.java`
    - `PostUnitTester.java`
    - `TextBookInterface.java`
    - `TextBookUnitTester.java`
- Create a class named `TextBookDriver.java` as a driver for the project that provides the navigation menu structure and console user interface (UI). This class contains the main method.
- Create a class named `TextBook.java` that will manage a collection of `Post` objects.
    - **NOTE**: This class **must** implement the API specified in the provided `TextBookInterface`:
      `public class TextBook implements TextBookInterface`
- Create a class named `Post.java` to represent a post within the `TextBook`.
    - **NOTE**: This class **must** implement the API specified in the provided `PostInterface`:
      `public class Post implements PostInterface`

Development Strategy

Before writing a single line of code, read all of the specifications in this document **and in the provided interface javadocs**. On paper, sketch out all program classes with their instance variables and methods. Draw lines between the classes and identify the relationships between the classes. This drawing does not need to be submitted with the project but helps you visualize the organization and responsibilities of each class in the program.

You are advised to use a "bottom up" development strategy that begins by fully implementing and testing the `Post.java` class, then fully implementing and testing the `TextBook.java` class, and finally the driver class `TextBookDriver.java`. We have provided the `PostUnitTester.java` class to test the functionality of the `Post` class and the `TextBookUnitTester.java` class to test the functionality of the `TextBook` class. These test classes are to help you confirm correct functionality of your program during development and do not need to be submitted with the project.

Project Specification

Quality Requirements (20 points)

1. Project Submission

- Code submitted to the correct cs121 section and with **only** the required files

2. `README.md`
   - `README` file is named properly and has the required sections including your reflection (see Documentation section)
   - Write a two paragraph reflection as a section of your README, describing your experience with this project. Talk about what worked well and what didn't work so well. Did you run into an issue that took some time to figure out? Tell us about it.
3. Javadoc Comments
   - Project should have proper javadoc formatted comments (see Documentation section)
4. Coding Style
   - Code should be properly *indented* and vertically spaced
   - Variable and Method names should follow the *camelCase* naming convention
   - Class names should follow the *TitleCase* naming convention
   - Constants should be in all *CAPITAL_LETTERS*
   - In-line comments to describe what different code sections do

Code Requirements (80 points)

The specifications for each class are listed below. The order that you implement the classes will be determined by your development strategy.

- `Post.java` - This class represents a single post.
  - This class must implement the provided `PostInterface`:

    ```
    public class Post implements PostInterface
    ```

  - Attributes: `author`, `text`, `timestamp`, `postID`, `comments`. Both `text` and `author` are `Strings`; `timestamp` is a java.time.Instant; `postID` is an `int`; and `comments` is an `ArrayList<String>`. Be certain to set the visibility of all instance variables to private to enforce encapsulation. Note that author is expected to be a one-word username, i.e. “AdaLovelace”, not “Ada Lovelace”. This will simplify file parsing later on.

  - Constructors: There are **two** required `Post` constructors, one to create a brand new `Post` and the other to restore a `Post` from data stored in an existing post file.

    - New Post constructor:
      Three parameters are required to create a new Post: `postId`, `author`, and `text`. The `timestamp` should be set to the moment the constructor was called (obtained from `Instant.now()`).
      A file named with format “Post-<postId>.txt” should be created in this

constructor and the first line should be written with the `postID`, `timestamp` in ISO-8601 format (as returned by `Instant`'s `toString()`), `author`, and `text`. Separate values with simple whitespace (space or tab) for ease of reading values from the file later. The ID in the filename should be formatted to be 5 digits long using the `DecimalFormat` class. For example, if the ID is 12, the post file should be named "Post-00012.txt". (This is the exact filename expected from Post's `getFilename()` method.)
First line format:
<postId> <timestamp> <author> <text>
Example first line:
00012 2010-08-30T13:38:23.085Z Sam Pondering the meaning of static.

- Post recovery constructor:
  The only parameter will be the `postID`. This constructor should open the Post-<postID>.txt file corresponding to the given postID (formatted to 5 digits) and set all instance variables to the values as read from the first line of the file.
  - Note that `Instant` has a `parse()` method that returns an `Instant` object reference after parsing a `String` in ISO-8601 date/time format.
  - After reading whitespace separated tokens for ID, timestamp, and username, the entire remainder of the first line is the post text.
  - Each additional line in the file is a comment. Each comment line can be read as a single `String` and stored in the `comments` list.
  - Since this constructor will need to open the file in a `Scanner`, it needs to catch `FileNotFoundException`. When the exception is caught, print a message stating that the file was not able to be opened.

- Methods:
  - `getFilename()` – Returns a `String` in the format of "Post-<postID>.txt", where the postID number is always 5 digits long (which can be accomplished using the `DecimalFormat` class).

- `isValid()` - Return true if none of the instance variables have a `null` value (uninitialized) and a readable file exists with the name returned by `getFilename()`, otherwise it returns false.

- `addComment()` - Takes two `String` parameters: the author and the comment text. This appends the timestamp (the `Instant` this method was called), the author, and the comment text on a single line at the end of the file that has the name returned by `getFilename()`.
  Example file contents after two comments have been added to a post:
  00012 2010-08-30T13:38:23.085Z Sam Pondering the meaning of static.
  2010-08-30T15:21:43.055Z Pat Don't get too existential.
  2010-08-31T09:15:12.118Z Jean Let me know if you get it.

- `toString()` - Return a `String` that contains all `Post` attributes followed by all comments formatted as shown.
  Example:
  Post:
  00012 2010-08-30T13:38:23.085Z Sam Pondering the meaning of static.
  Comments:
  2010-08-30T15:21:43.055Z Pat Don't get too existential.
  2010-08-31T09:15:12.118Z Jean Let me know if you get it.
  Note - The "Comments:" header should be displayed even if there are no comments for the post.

- `toStringPostOnly()` - Return a formatted `String` that contains only `Post` attributes - no comments, and no headers.
  Example:
  00012 2010-08-30T13:38:23.085Z Sam Pondering the meaning of static.

- Getter methods for each instance variable.

- `TextBook.java` - This class manages a collection of `Posts` and provides methods for interacting with this collection.

  - Attributes: `posts` - an `ArrayList<Post>`, and `lastID` - the last/largest known post ID or 0 if no `Posts` exist. Be certain to set visibility of all instance variables to private in order to enforce encapsulation.

  - Constructor: No parameters are required. Initialize `posts` as an empty `ArrayList` of `Post` objects and set `lastID` to 0. Attempt to open the posts file (filename stored in constant POST_LIST_FILE defined in the TextBookInterface) and populate the list with `Posts` corresponding to each ID read from the file. Store the last read ID in `lastID`.

- Methods:
    - `addPost(String author, String text)` - Increments `lastID`. Creates a new `Post` object with the given parameters. Adds the new `Post` to the internal `ArrayList` of `Posts`. Appends the new `Post`'s ID to the POST_LIST_FILE file.
    - `removePost(int index)` - Removes and returns the `Post` at the specified index in the `ArrayList` of `Posts` if the index is in bounds, and rewrites POST_LIST_FILE file to exclude removed ID. If the index is invalid, the method returns `null`.
    - `addComment(int index, String author, String text)` - Adds a comment to the indexed Post. Returns true if the index is valid or false if the index is invalid.
    - `getPostString(int index)` - Returns the full `toString()` String from the `Post` at the specified index (including `Post` and all comments). Returns `null` if the index is out of bounds.
    - `toString()` - Returns a well-formatted String displaying the number of `Posts` and an indexed list of all `Posts`. Utilizes a loop to walk through the posts `ArrayList` and calls the `toStringPostOnly()` method for each `Post` to get its one-line `String` to follow each index. Example:
      TextBook contains 3 posts:
      0 - 00010 2021-11-02T12:20:59.122Z Kathryn Held office hours.
      1 - 00011 2021-11-02T12:45:52.023Z Luke Made another video.
      2 - 00013 2021-11-02T12:50:18.544Z Mason Drank coffee.
    - `getPosts()` - This getter should return a reference to a **copy** of the `posts` list, not the `posts` reference, itself. *This method is intended for testing only and should not be used in `TextBookDriver`.*

      NOTE: This technically still violates encapsulation because we're not performing a 'deep' copy of the `Post` objects themselves. However, this is adequate for our purposes in this project.

- `TextBookDriver.java` - This is the driver for the project, containing the main() method which is the entry point for the application. This class creates a `TextBook` instance and provides console menu logic allowing the user to interact with the `TextBook`. This class should have **no** instance or class variables. Do **not** create an `ArrayList<Post>` locally or duplicate any functionality in the driver class that belongs to the `TextBook` or `Post` classes.

- Welcome the user to your TextBook social media platform and ask them for their one-word login name, which will be used as the author for all subsequent posts and comments.
- Use a while loop, switch statement, and keyboard `Scanner` to build a basic console menu system. The menu UI should display a list of options and allow the user to "(p)rint the textbook (indexed list of posts), (a)dd a post to the TextBook, (d)elete a post from the TextBook, (c)omment on a post, (r)ead a post with comments, or (q)uit the application". In the prompt, let the user know they can repeat the menu by choosing 'm'. Note that **all** menu choices should be case-insensitive.
    - The (d)elete, (c)omment, and (r)ead options should prompt the user for the index of the post they wish to remove, comment on, or read. If the input index is invalid, print a message saying that it is an invalid index.
    - Only valid posts can be (r)ead by the user. If a post is not valid, a message should be displayed stating that the post is unavailable.
    - The (a)dd option should prompt the user to enter the post's text. This text and the author name provided at the beginning are then used to add a new `Post` to the `TextBook`. Likewise, the (c)omment option should prompt for the comment text and use the current author's name when adding a new comment to the chosen `Post`.
    - Any invalid menu choice should result in a message that the choice is invalid.
    - After each action, the user should be prompted to enter another menu option or 'm' to repeat the menu until they select (q)uit.

Example Partial Run of Interaction

```
Please enter your name to enter TextBook: Jerry
Jerry, welcome to TextBook - the totally text social media site!
TextBook Site Menu
-----------------------------------
  (P)rint TextBook posts
  (A)dd a new post
  (D)elete a post
  (C)omment on a post
  (R)ead a post with comments
  (Q)uit
```

```
---------------------------------

Select a menu option or reprint the (M)enu: a
Enter the text for your new post: CS 121 is the best class ever (even
though it is challenging).

Select a menu option or reprint the (M)enu: p
TextBook has 1 post:
0 - 00114 2021-10-25T14:35:31.356Z Jerry CS 121 is the best class
ever (even though it is challenging).

Select a menu option or reprint the (M)enu: c
Enter the index of the post to comment on: 0
Enter your comment: I really enjoy this class!

Select a menu option or reprint the (M)enu: r
Please enter the index of the Post to read: 0
Post:
00114 2021-10-25T14:35:31.356Z Jerry CS 121 is the best class ever
(even though it is challenging).
Comments:
2021-10-25T14:36:01.547Z Jerry I really enjoy this class!

Select a menu option or reprint the (M)enu: Q
Goodbye, Jerry!
```

Testing

The program should be tested continuously throughout the development process to quickly catch errors. Once the project is complete and ready for submission, copy your project code to a folder on onyx. Run the following tests from this folder on onyx and make certain they pass before submitting.

Check that code compiles on onyx.
`javac *.java`
Check that your Post class passes all required functionality tests.
`java PostUnitTester`
Check that your TextBook class passes all required functionality tests.
`java TextBookUnitTester`
Check that your console UI runs and allows the user to add, remove, list and read posts. Then RE-RUN your console UI and confirm that all of the posts and comments created in the previous session have been recovered.
`java TextBookDriver`

Documentation

If you haven't already done so, add **javadoc comments** to each class and all methods.

- Javadoc *class* comments should include both a short description of the class AND an `@author` YOURNAME tag.
    - The class javadoc comment is placed before the class definition
    - Your class comment must include the @author tag at the end of the comment. This will list you as the author of your software when you create your documentation.
- Javadoc *method* comments should be placed before ALL public methods.
    - Methods that are defined by an interface do not require additional comments since they are provided within the interface specification.

Include a plain-text file called **README.md** that describes your program and how to use it. It should also include your reflection on the project. Expected formatting and content are described in README_TEMPLATE.md. See README_EXAMPLE.md for an example.