Assignment 2

Due date

- 11.59 PM EST, June 22nd

Git url

- https://classroom.github.com/a/jTL4tq8Y

Submit your code as per the provided instructions.

Assignment Goal

Implement the State Pattern to capture the project requirements.

Team Work

- No team work is allowed. Work individually. You cannot discuss the assignment with ANYONE other than the instructor and TA.

Programming Language

You are required to program using Java.

Compiling and Running Commands

- Compilation: Your code should compile on remote.cs.binghamton.edu with the following command: ant -buildfile src/build.xml all
- Running the code: Your code should run on remote.cs.binghamton.edu with the following command: ant -buildfile src/build.xml run -Dinput="<path/to/inputfile>" -Doutput="<path/to/outputfile>"

Policy on sharing of code

EVERY line of code that you submit in this assignment should be written by you. Do NOT show your code to any other student. Do not copy any code from any online source. Code for File I/O or String operations, if found online, should be clearly cited, and you cannot use more than 5 lines of such online code.

Code downloaded in its entirety from an online repository of code (GitHub, BitBucket, etc.) and submitted as student's own work, even if citied, is considered plagiarism.

Code snippets, for File I/O, if used from an online source should be cited by mentioning it in the README.md and also in the documentation of every source file in which that code appears.

Post to the piazza if you have any questions about the requirements. Sending an email will only get a response requesting you to post on piazza. DO NOT post your code to piazza asking for help with debugging.

Project Description

Design and Implement a program for Youtube to categorize a single channel based on popularity.

- Prior to working on the assignment, please read through the grading instructions to understand the minimum requirement of the assignment.
- A Youtube channel has a popularity score, defined as *The average popularity score of all videos contained in the channel*. Based on its popularity score, a channel can be in one of the following states.
    - UNPOPULAR - This is the starting state of a channel. For a channel to be in this state, its popularity score should in the range [0, 1000].
    - MILDLY_POPULAR - For a channel to be in this state, its popularity score should be in the range (1000, 10000].
    - HIGHLY_POPULAR - For a channel to be in this state, its popularity score should be in the range (10000, 100000].
    - ULTRA_POPULAR - For a channel to be in this state, its popularity score should be in the range (100000, INT_MAX].

    Note: () represents an open interval (not including end points) and [] represents a closed interval (including endpoints). For example, $X \in (10, 20]$ iff $X \in R$ AND $10 < X \leq 20$.
    Question to ask yourself at this point - Is Channel a state or a context? Enumerate the names of the states.

- A channel can hold multiple videos. The popularity score of a video is defined by the following three metrics.
    - Total number of views (≥ 0).
    - Total number of likes (≥ 0).
    - Total number of dislikes (≥ 0).

Using the above information, the popularity score of a video at any point is given by the formula ***#Views + 2 * (#Likes - #Dislikes)*** where # signifies total count so far. The data for number of views, likes and dislikes are provided in the input file.
Question to ask yourself at this point - How and where are videos stored?

- The current state of the channel determines whether advertisement requests are approved or rejected based on how long they are. The following are the rules for the same.
    - When state=UNPOPULAR, advertisements of length in range (1,10] are approved and the rest are rejected.
    - When state=MILDLY_POPULAR, advertisements of length in range (1, 20] are approved and the rest are rejected.
    - When state=HIGHLY_POPULAR, advertisements of length in range (1, 30] are approved and the rest are rejected.
    - When state=ULTRA_POPULAR, advertisements of length in range (1, 40] are approved and the rest are rejected.

  The requests to add advertisements of a certain length are also provided via the input file.
  Question to ask yourself at this point - Where are the advertisement requests processed?

Input Processing and Control flow

1. The input file is processed one line at a time.
2. Each line can correspond to one of the following.
    - Adding a video to the channel. Input format: **ADD_VIDEO::<video name>**.
    - Removing a video from the channel. Input format: **REMOVE_VIDEO::<video name>**.
    - Views, Likes and Dislikes. Input format: **METRICS__<video name>::[VIEWS=<delta in #views>,LIKES=<delta in #likes>,DISLIKES=<delta in #dislikes>]**.
      Note: There are no spaces before or after the comma character.
      Note: Views, Likes and Dislikes MUST be integers.
    - Advertisement requests. Input format: **AD_REQUEST__<video name>::LEN=<length>**.
      Note: Advertisement length MUST be an integer.
3. ADD_VIDEO - If the video already exists or the format of the input is invalid, throw the appropriate exception reporting a meaningful error message and terminate. If the input is valid, then add the video to the collection of videos in

the channel. This new video is assigned an initial popularity score of 0. This operation is performed by the current state.
The processing of this line should result in the string **<current state name>__VIDEO_ADDED::<video name>** being written to Results.

4. REMOVE_VIDEO - If the video does not exist or the format of the input is invalid, throw an appropriate exception reporting a meaningful error message and terminate. If the input is valid, then remove this video from the collection of videos in the channel. The channel's popularity score will need to be updated after the removal. This operation is performed by the current state.
   The processing of this line should result in the string **<current state name>__VIDEO_REMOVED::<video name>** being written to Results.

5. METRICS - If the video does not exist or the format of the input is invalid, throw an appropriate exception reporting a meaningful error message and terminate. If the input is valid, then update the metrics of the corresponding video and recalculate the popularity score of the channel. This operation is performed by the current state. The metric data is provided as delta changes.

   Consider the input line METRICS__testvideo::[VIEWS=10,LIKES=10,DISLIKES=-20]. This means that the total views increased by 10, total likes increased by 10 and the total dislikes decreased by 20, for the video named 'testvideo'.

   The processing of this line should result in the string **<current state name>__POPULARITY_SCORE_UPDATE::<new popularity score of channel>** being written to Results.
   Note: unlike likes and dislikes, the delta in the number of views cannot be negative.
   Note: If there is a decrease in the number of likes or dislikes, it cannot be more than the total number of likes or dislikes received thus far for the video.

6. AD_REQUEST - If the video does not exist or the length is negative, throw an appropriate exception reporting a meaningful error message and terminate. If the input is valid, report if the request is approved or rejected based on whether the length of the advertisement falls within the current state's acceptable ad length range (mentioned above).
   The processing of this should result in the string **<current state name>__AD_REQUEST::<APPROVED / REJECTED>** being written to Results.

7. If any of the lines in the input file are invalid, then the whole input is to be considered invalid.
8. Upon processing of a ADD_VIDEO, REMOVE_VIDEO or METRICS input if the new popularity score of the channel falls in a range acceptable by a state other than the current state then a state change occurs.
9. Once all the lines of the input file have been processed cast the results instance to the appropriate interface and call the necessary method to persist the results to the output file.

INPUT

Your program should accept two files from the commandline - input file and output file. These file names/paths will be provided using the following command-line options.

- ***-Dinput***: Path to the Input file.
- ***-Doutput***: Path to the output file.

EXAMPLES

input

```
ADD_VIDEO::video1
ADD_VIDEO::video2
METRICS__video1::[VIEWS=1000,LIKES=20,DISLIKES=20]
AD_REQUEST__video1::LEN=8
METRICS__video2::[VIEWS=2000,LIKES=400,DISLIKES=20]
METRICS__video1::[VIEWS=20000,LIKES=1000,DISLIKES=-10]
AD_REQUEST__video2::LEN=39
METRICS__video2::[VIEWS=50,LIKES=-50,DISLIKES=0]
REMOVE_VIDEO::video2
ADD_VIDEO::video3
METRICS__video3::[VIEWS=2000,LIKES=100,DISLIKES=20]
METRICS__video1::[VIEWS=0,LIKES=-1000,DISLIKES=500]
ADD_VIDEO::video4
METRICS__video4::[VIEWS=100,LIKES=5,DISLIKES=0]
REMOVE_VIDEO::video1
AD_REQUEST__video3::LEN=15
REMOVE_VIDEO::video3
REMOVE_VIDEO::video4
```

output

```
UNPOPULAR__VIDEO_ADDED::video1
UNPOPULAR__VIDEO_ADDED::video2
UNPOPULAR__POPULARITY_SCORE_UPDATE::500
UNPOPULAR__AD_REQUEST::APPROVED
UNPOPULAR__POPULARITY_SCORE_UPDATE::1880
MILDLY_POPULAR__POPULARITY_SCORE_UPDATE::12890
HIGHLY_POPULAR__AD_REQUEST::REJECTED
HIGHLY_POPULAR__POPULARITY_SCORE_UPDATE::12865
HIGHLY_POPULAR__VIDEO_REMOVED::video2
HIGHLY_POPULAR__VIDEO_ADDED::video3
HIGHLY_POPULAR__POPULARITY_SCORE_UPDATE::12590
HIGHLY_POPULAR__POPULARITY_SCORE_UPDATE::11090
HIGHLY_POPULAR__VIDEO_ADDED::video4
MILDLY_POPULAR__POPULARITY_SCORE_UPDATE::7430
MILDLY_POPULAR__VIDEO_REMOVED::video1
MILDLY_POPULAR__AD_REQUEST::APPROVED
MILDLY_POPULAR__VIDEO_REMOVED::video3
UNPOPULAR__VIDEO_REMOVED::video4
```

NOTES ON GRADING

- Class participation points will be given to students who post good questions on piazza seeking clarification on the assignment or finding ambiguities, and also to those who respond and participate to help make the discussion interesting and informative.
- Class participation points will be given to students who also post sample interesting input examples with corresponding output and metrics.

Sample Input Files sent by students in this course

Please check piazza.

Compiling and Running Java code

- Your submission must include a readme in markdown format with the name **README.md**.
- Your README.md file should have the following information:
    - instructions on how to compile the code
    - instructions on how to run the code
    - justification for the choice of data structures (in terms of time and/or space complexity).
    - citations for external material utilized.

- You should have the following directory structure (replace username with your github username).
- ./csx42-summer-2020-assign2-username
- ./csx42-summer-2020-assign2-username/README.md
- ./csx42-summer-2020-assign2-username/.gitignore
- ./csx42-summer-2020-assign2-username/channelpopularity
- ./csx42-summer-2020-assign2-username/channelpopularity/src
- ./csx42-summer-2020-assign2-username/channelpopularity/src/build.xml
- ./csx42-summer-2020-assign2-username/channelpopularity/src/channelpopularity
- ./csx42-summer-2020-assign2-username/channelpopularity/src/channelpopularity/operation
- ./csx42-summer-2020-assign2-username/channelpopularity/src/channelpopularity/operation/Operation.java
- ./csx42-summer-2020-assign2-username/channelpopularity/src/channelpopularity/context
- ./csx42-summer-2020-assign2-username/channelpopularity/src/channelpopularity/context/ContextI.java
- ./csx42-summer-2020-assign2-username/channelpopularity/src/channelpopularity/state
- ./csx42-summer-2020-assign2-username/channelpopularity/src/channelpopularity/state/StateName.java
- ./csx42-summer-2020-assign2-username/channelpopularity/src/channelpopularity/state/StateI.java
- ./csx42-summer-2020-assign2-username/channelpopularity/src/channelpopularity/state/AbstractState.java
- ./csx42-summer-2020-assign2-username/channelpopularity/src/channelpopularity/state/factory
- ./csx42-summer-2020-assign2-username/channelpopularity/src/channelpopularity/state/factory/SimpleStateFactoryI.java
- ./csx42-summer-2020-assign2-username/channelpopularity/src/channelpopularity/state/factory/SimpleStateFactory.java
- ./csx42-summer-2020-assign2-username/channelpopularity/src/channelpopularity/util
- ./csx42-summer-2020-assign2-username/channelpopularity/src/channelpopularity/util/FileProcessor.java
- ./csx42-summer-2020-assign2-username/channelpopularity/src/channelpopularity/util/Results.java

- ./csx42-summer-2020-assign2-username/channelpopularity/src/channelpopularity/util/FileDisplayInterface.java
- ./csx42-summer-2020-assign2-username/channelpopularity/src/channelpopularity/util/StdoutDisplayInterface.java
- [Other Java files and packages you may need]

Code Organization

- Your directory structure should be EXACTLY as given in the code template. You are free to add additional packages and source files.

Submission

- Make sure all class files, object files (.o files), executables, and backup files are deleted before creating a zip or tarball (use the gitignore file for this). To create a tarball, you need to "tar" and then "gzip" your top level directory. Create a tarball of the directory csx42-summer-2020-assign2-username. We should be able to compile and execute your code using the commands listed above.
- Instructions to create a tarball
    - Make sure you are one level above the directory csx42-summer-2020-assign2-username.
    - tar -cvzf csx42-summer-2020-assign2-username.tar.gz csx42-summer-2020-assign2-username/
- Upload your assignment to Blackboard, assignment-2.

General Requirements

- Upload a picture of a hand-drawn state diagram showing the states and the state changes to piazza and tag it with the "assignment2-state-diagram" folder.
- Start early and avoid panic during the last couple of days.
- Separate out code appropriately into methods, one for each purpose.
- Class names should be nouns. Method names should be verbs.
- The name of the method should tell us what the method is doing, nothing more or nothing less.
- You should document your code. The comments should not exceed 72 columns in width. Use javadoc style comments if you are coding in Java. Include javadoc style documentation. It is acceptable for this assignment to just have the return type described for each method's documentation.
- Do not use "import XYZ.*" in your code. Instead, import each required type individually.

- All objects, in Java, that may be needed for debugging purposes should have the "toString()" method defined. By default, just place a toString() in every class.
- Every class that has data members, should have corresponding accessors and mutators (unless the data member(s) is/are for use just within the method.).
- Java naming conventions MUST be followed.
- Design Guidelines and Considerations
    - The context should use the simple factory design pattern to fetch the required state. Use the following sub points to get further clarity on this.
        1. The constructor of the context accepts an instance of SimpleStateFactory.
        2. The create(...) method of SimpleStateFactory accepts an enum representing the state to be instantiated.
        3. The setState(...) method of the context accepts an enum representing the state to change to. This is provided to the SimpleStateFactory instance to fetch an instance of the respective state.
        4. Code snippet shown below should help.

```
5.              // Assume imports.
6.              public ChannelContext implements ContextI {
7.                  private StateI curState;
8.                  private Map<StateName, StateI>
   availableStates;
9.                  public ChannelContext(SimpleStateFactoryI
   stateFactoryIn, List<StateName> stateNames) {
10.                             // initialize states using
   factory instance and the provided state names.
11.                             // initialize current state.
12.                         }
13.
14.                         // Called by the States based on
   their logic of what the machine state should change to.
15.                         public void
   setCurrentState(StateName nextState) {
16.                             if
   (availableStates.containsKey(nextState)) { // for safety.
17.                                 curState =
   availableStates.get(nextState);
18.                             }
19.                         }
20.                     }
```

    - As the logic for calculating the popularity score for videos, and in turn of the channel, is the same regardless of the current state, this can be put in a method in an abstract class that implements the StateI interface. Note that this would result in the states extending the abstract class instead of directly implementing the StateI interface, which okay.

    - Enumerating the state names and operations follows good design practices.
    - The name of a state is always fixed. Therefore, the state implementations should have their name as a constant member (Question: What access specifiers and modifiers would you use?).
    - Can popularity score be performed in constant time? This would increase the processing throughput.
    - The state should not directly write to the output file. Instead, they should all share access to the same Results instance and store the output strings in it. Once all processing is complete, the driver code casts the results instance to the specific interface and calls the associated method to persist the data either to the output file.
    - Always program to an interface. This applies to java standard library as well. For example, when instantiating an ArrayList, write *List<? extends Object> l = new ArrayList<? extends Object>();*.
- The following rules MUST be followed.
    1. FileProcessor code has been given to you. This should NOT be altered. Use the FileProcessor for reading in the input file line by line. Read the documentation to understand how the FileProcessor works.
    2. The input file should be processed one line at a time.
    3. The program should not read in all the input and store it in a data structure.
    4. Use enums to represent operations to be performed.
    5. Use enums to represent states.
    6. State change, if any, has to happen after performing respective operation and not before performing the operation.
    7. The output format has to be exactly the same as that shown. Failure to generate output adhering to the given format will result in loss of points.

Late Submissions

- The policy for late submissions is that you will lose 10% of the grade for each day that your submission is delayed. **There is NO difference in penalty for assignments that are submitted 1 second late or 23 hours late** .
- If you are using slack days, please mention it in the README file.

Grading Guidelines

Grading guidelines have been posted here.

Grading Instructions

Total Points (none of the below deductions are made): 100 points.

- gzip has been NOT created correctly: -2 points
- gitignore file not used to exclude executables, .class, backup, and jar files from the project: -2 points.
- No hand-drawn state diagram uploaded to piazza: -8 points.
- The following command-line argument validations have not been made.
    1. Incorrect number of commandline arguments: -2 point.
    2. Missing input file for the argument -Dinput: -1 point.
- The following input validations have not been made.
    1. Input file is empty: -2 point
    2. Line in the input file does not follow the specified formats: -2 point
    3. Video added is already present: -2 points.
    4. Video being asked to remove does not exist: -2 points.
    5. Negative value for number of views in an input line: -2 points.
    6. Decrease in likes or dislikes is more than the total number of likes or dislikes, respectively, for the video: -2 points.
    7. Video associated with an advertisement request does not exist: -2 points.
    8. Values for views, likes, dislikes or advertisement length are not integers: -2 points.
- Code does not compile correctly with ANT according to the instructions in the README: -2 points.
- Code compiles but does not execute correctly and abruptly terminates: -2 points.
- Incorrect user of git.
    - Less than 6 commits in total: -2 points
    - ≤ 3 commits 48 hours before the submission deadline: -2 points.
    - commit messages not meaningful and do not explain the functionality that has been added compared to the previous version of the code: -2 points.
    - More than 50% of the number of lines of code is committed in the last 24 hours: -4 points

    - More than 50% of the number of lines of code is committed in the last 48 hours: -2 points
- Code does not generate correct output.
    - Incorrect output for ADD_VIDEO operation: -2 points.
    - Incorrect output for REMOVE_VIDEO operation: -3 points.
    - Incorrect output on processing video METRICS: -3 points.
    - Incorrect output on processing AD_REQUEST: -2 points.
- State Pattern not implemented correctly.
    0. No context interface: -2.5 points
    1. Context does not hold interface reference of current state: -2.5 points
    2. Context determines which state to change to instead of the state determining the same: -2.5 points
    3. Functionality of state is performed by the context: -2.5 points
    4. State change happens before a line processed: -3 points.
- States directly write to output file without using Results instance: -5 points.
- Not programming to an interface: -3 points.
- Using import xyz.*: -3 points.
- Not following java naming conventions: -3 points
- Generic exceptions thrown or caught (*throw new Exception(...)* or *catch (Exception e)*): -3 points
- Program does not produce meaningful error messages, if any: -3 points
- Simple Factory design pattern not used for instantiating states: -5 points.
- State names not stored as constant members in each state implementation: -2 points.
- Enums not used to represent state names and operation names: -3 points.

Back to Programming Design Patterns: Assignment 2