Assignment 3

Due date

- 11.59 PM EST, July 10th

Git url

- https://classroom.github.com/a/QNRu9iUa

Submit your code as per the provided instructions.

Assignment Goal

Apply the design principles you have learned so far to develop software for the given problem.

Team Work

- You need to work alone on this assignment.
- You cannot discuss with anyone the design patterns(s) to be used for this assignment.

Programming Language

You are required to use Java.

Compiling and Running Commands

- Compilation: Your code should compile on remote.cs.binghamton.edu
- You are required to use ANT as the build tool for this assignment.

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 group. Our code-comparison software can very easily detect similarities.

Post to piazza if you have any questions about the requirements. Do not send emails. Do not post your code asking for help with debugging. However, it is okay to post design/concept questions on programming in Java/C/C++.

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.

Project Description

Replica System for Student Records

- Each Replica System needs to be represented as a tree, wherein each node of the tree corresponds to a Student Record.
- Each Tree needs to be given a unique ID (0, 1, 2, for example). Use a final int in each Tree to save this id. Write a utility class with a static method to return this unique id to be returned whenever it is called.
- Create a class called StudentRecord to serve as nodes for the tree. A student record consists of the following:
    - a bNumber (4 digit positive integer),
    - a String for "firstName",
    - a String for "lastName",

  - a double value for "gpa",
  - a String for "major",
  - and a set of Strings for skills. The max number of skills that will be listed in the input file will be 10.
- Write code for a tree that has the features to insert and search Student Records. You need to write code for the tree by yourself. However, you are allowed to get help from a book or online source (for example the source code for BST, AVL, 2-3, 2-3-4, etc.) If you do so, cite the URL both in your source code and also README. It will be considered CHEATING if you do not cite the source of any code on tree that you use/adapt.
- The three trees (replica_0, replica_1, and replica_2) should be separate instances of the same Tree. Note that you should appropriately use Camel case (as per Java naming convention)) instead of the names I have listed here.
- Each Node of the tree should implement both the SubjectI and the ObserverI interface.
- As you need to modify the code to implement the Observer pattern, you can't use the in-built Observer pattern in Java.
- Populate the tree using the data from an input file provided with the -Dinput commandline option. A line in the input file has the following format:

  ```
  <B_NUMBER>:<FIRST_NAME>,<LAST_NAME>,<GPA>,<MAJOR>,[<SKILL>,[<SKILL>, ...]]
  ```

  Example is shown below.

  ```
  1234:John,Doe,3.9,ComputerScience,Skill3,Skill1,Skill2,Skill5,Skill4,
  2345:John,Doe,3.9,Chemistry,Skill0,Skill7,Skill8,Skill5,Skill3,Skill4,Skill3,Skill9
  1124:Jane,Doe,3.9,Chemistry,Skill9,Skill8,Skill7,Skill6,Skill2,Skill3,Skill4
  ...
  ```

- The input lines may have bNumbers that are repeated. If a node already exists, change the data member values in the Node with the new values from the latest input line. Note that by using a set for the skills, only unique Strings can be added.
- Your TreeHelper should be such that when you create a node (replica_Node_0), you should clone it twice to get two Nodes (let's say replica_Node_1 and replica_Node_2). Setup these three nodes to be replicas of each other. The TreeHelper is a helper that helps build the tree recrusively.
  - So, replica_Node_1 and replica_Node_2 should be observers of replica_Node_0. As replica_Node_0 is the subject in this case, you should store the references of replica_Node_1 and replica_Node_2 in a data structure for listeners (array list of references, for example).
    - replica_Node_0 should be inserted in replica_tree_0.
    - replica_Node_1 should be inserted in replica_tree_1.
    - replica_Node_2 should be inserted in replica_tree_2.
  - replica_Node_0 and replica_Node_1 should be observers of replica_Node_2. Continue with steps corresponding to the ones listed in the 1st bullet.
  - replica_Node_0 and replica_Node_2 should be observers of replica_Node_1. Continue with steps corresponding to the ones listed in the 1st bullet.
- You do NOT need to write code to delete nodes, as the input and modification files should not require deletion of any node.
- The Replica trees should NOT be setup a observers or subjects. The replicas should be setup at only the Node level.
- Process one line at a time from the modification file provided using the -Dmodify commandline option. The file has the following format (first entry corresponds to the replica number, the 2nd to the bNumber, and 3rd to the value that needs to be searched and replaced/modified).

  ```
  <REPLICA_ID>,<B_NUMBER>,<ORIG_VALUE>:<NEW_VALUE>
  ```

  An example is shown below.

```
0,1234,John:John7
1,2345,Chemistry:MathematicalSciences
2,1234,Skill1:Skill9
...
```

- The modification file will not have any request to change the GPA.
- If the request for modified value is empty (for example, "Mathematical Sciences:"), print a meaningful message to error file and proceed to the next line.
- Search for the node with the bNumber in the line from the modification file, and then modify the corresponding attribute value in that Node. If that attribute value does not exist to modify, then ignore and move to the next line. Once the change to the specified replica_Node_X is done, then the change should be sent to both the observer nodes. Note that the updates should be sent before the next line is processed.
- The update(...) call should do the following:
    - determine if the call on the observer is for INSERT or MODIFY.
    - search for the appropriate node using the bNumber.
    - If it is an INSERT based update call, then use the latest values. Note that for skills, insert should only add to the existing set, and not delete any existing skill.
    - If it is a MODIFY based update call, then every occurrence of the origValue should be replaced with the provided modifiedValue.
- From the command line accept the following args:
    - Input file path with the *-Dinput* commandline option.
    - Modification path file with the *-Dmodify* commandline option.
    - Output file path for tree 1 with the *-Dout1* commandline option.
    - Output file path for tree 2 with the *-Dout2* commandline option.
    - Output file path for tree 3 with the *-Dout3* commandline option.
    - Error file path with the *-Derror* commandline option.
    - Debug value with the *-Ddebug* commandline option.
- Do not hardcode the file names in your code. Use the file names provided via command line.
- Add a method to your tree, printNodes(Results r, ...), that stores in Results the list of skills for each student, sorted by the bNumber in ascending order.
- Your driver code should do the following:
    - Read command line arguments.
    - Build the three trees, based on the input file and the observer pattern. It is recommended that you use a TreeHelper to build each tree recursively. The TreeHelper can hold references to the roots of the 3 trees that are being built, and have helper methods to insert, printNodes(...), etc. It is also ok to build your trees in other ways as long as the design will work if the number of replicas is increased/decreased.
    - Apply updates according to modification file.
    - Create a new Results instance and call printNodes(...) on each of the three trees to store the bNumber and list of skills to store in Results. So, each of the three trees will use a different instance of Results.
    - Call a method in Results, via the method in FileDisplayInterface, to write the data stored in Results to three output files, for the three trees. You can run a "diff" on the three output files to make sure your Observer pattern worked correctly.
- When you read in an input file, you should insert the skill into the main tree and in that tree update the node corresponding to the bNumber. If a node with the bNumber exists, you should update that node in the main tree, notifyAll(...) to the two corresponding listener nodes. If no such node with that bNumber exists in the main tree, then do the following:
    - create a new Node (lets call it new node)
    - clone this new node twice, and setup all the listeners and subjects.
    - insert the nodes in their respective trees, recursive insert from the root.
- Note that notifyAll(...) could be used for processing modification file and also input file (for example, when you need to add a skill to an existing bNumber from the input file). So, add enum as a parameter to

the notifyAll(...) call in the SubjectI, also to update(...) in ObserverI, to indicate whether the notifyAll(...) call, which calls update(...) on the observer nodes, is for INSERT or MODIFY.

- Calls made to notifyAll(...) should be propagated to the observers. However, calls to update(...) on a node should not be propagated any further to prevent a cycle (infinite loop).
- In your README.md, briefly describe how the observer pattern has been implemented, and the URLs from where you borrowed ideas/code for the tree implementation.
- For class participation, post interesting, but valid, input and modification files to piazza.
- Create and use your own MyLogger scheme for debugging. Here is an example of how you should use MyLogger.

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-assign3-username
- ./csx42-summer-2020-assign3-username/README.md
- ./csx42-summer-2020-assign3-username/.gitignore
- ./csx42-summer-2020-assign3-username/studentskills/src/build.xml
- ./csx42-summer-2020-assign3-username/studentskills/src/studentskills/driver
- ./csx42-summer-2020-assign3-username/studentskills/src/studentskills/driver/Driver.java
- ./csx42-summer-2020-assign3-username/studentskills/src/studentskills/util
- ./csx42-summer-2020-assign3-username/studentskills/src/studentskills/util/FileProcessor.java
- ./csx42-summer-2020-assign3-username/studentskills/src/studentskills/util/Results.java
- ./csx42-summer-2020-assign3-username/studentskills/src/studentskills/util/FileDisplayInterface.java
- ./csx42-summer-2020-assign3-username/studentskills/src/studentskills/util/StdoutDisplayInterface.java
- ./csx42-summer-2020-assign3-username/studentskills/src/studentskills/util/MyLogger.java
- ./csx42-summer-2020-assign3-username/studentskills/src/studentskills/mytree
- ./csx42-summer-2020-assign3-username/studentskills/src/studentskills/mytree/TreeHelper.java
- ./csx42-summer-2020-assign3-username/studentskills/src/studentskills/mytree/StudentRecord.java
- ./csx42-summer-2020-assign3-username/studentskills/src/studentskills/mytree/SubjectI.java
- ./csx42-summer-2020-assign3-username/studentskills/src/studentskills/mytree/ObserverI.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-assign3-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-assign3-username.
  - tar -cvzf csx42-summer-2020-assign3-username.tar.gz csx42-summer-2020-assign3-username/
  - gzip csx42-summer-2020-assign3-username.tar
- Upload your assignment to Blackboard, assignment-3.

General Requirements

- 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.

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** .

Grading Guidelines

Grading guidelines have been posted here.