Assignment 2

Due date

- 11.59 PM EST on October ~~6th~~ 7th

Submit your code as per the provided instructions.

Updates

- Sun Sep 30 16:56:42 EDT 2018: updated the assignment structure.

Assignment Goal

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

Github link

The git link is https://classroom.github.com/a/QWMEJOcS

Team Work

- You are required to work alone on this assignment.

Programming Language

You are required to use Java.

Compilation Method

- You are required to use ANT to compile the code.

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

Project Description

Backup System for Student Course Assignments

- Create a class Node to store the B-Number (3 digit int) and an arraylist of course names (strings). The possible course names are "A" to "Z" (include all the letters of the alphabet).
- Write code for a tree that has the features to insert and search. You are not required to implement deletion of nodes. 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 of where you got the code from both in your source code and also README.md. It will be considered CHEATING if you do not cite the source of any code on tree that you use/adapt.
- As you need to modify the code to implement the Observer pattern, you can't just use an in-built tree in Java. Each Node of the tree should implement both the Subject and the Observer interface.
- Do not use the built-in Observer classes/interfaces in Java.
- Populate the tree using the data from an input file, input.txt, that has the following format:

```
123:C
234:D
124:A
122:D
125:E
235:F
342:Z
...
```

- For class participation, post to piazza interesting input.txt and delete.txt so that I can post it to the course assignment page. The first 10 sample files will get class participation points.
- The input lines may not be unique. In such cases, ignore the repeated lines.
- Assume that the input.txt and delete.txt will be well formatted.
- Your tree builder should be such that when you create a node (nodeOrig as the variable name), you should clone it twice to get two Nodes (let's say backupNode1 and backupNode2 as the variables holding the references). Setup backupNode1 and backupNode2 as observers of nodeOrig. nodeOrig, backupNode1, and backupNode2, should be instances of the same Node class. As nodeOrig is the subject, you should store the references of backupNode1 and backupNode2 in a data structure for listeners in nodeOrig (array list of references, for example).
- The notifyAllObservers(...) method in the SubjectI interface and the update(...) method in the ObserverI interface should take two parameters: String and an enum. The String should have the course name. The enum should indicate whether the update is being called to insert a new course in an existing node, or to delete a course from an existing node.
- Apply the delete operations, processing line at a time, from the file delete.txt. The file has the following format:

```
189:C
845:D
634:A
...
```

- Search for the node with the B_Number in the line, and then delete the corresponding course in that Node. If that course does not exist to delete, then ignore and move to the next line. If a node does not exist with that BNumber, then ignore and move to the next line. Once the changes to the nodeOrig are done, the changes should be updated to both the corresponding nodes (call notifyAllObservers(...)). Note that the updates for a line in delete.txt should be sent, before the next line is processed.
- From the command line accept the following args: input.txt, delete.txt, output1.txt, output2.txt, output3.txt.
- Add a method to your tree, printNodes(Results r, ...), that stores in Results the list of courses for each student, sorted by the B_Number.
- Your driver code should do the following:
  - Read command line arguments.
  - Build the three trees, based on input.txt and the observer pattern.
  - Apply updates according to delete.txt
  - Create a new Results instance and call printNodes(...) on each of the three trees to store the BNumber and list of courses 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 output1.txt, output2.txt, and output3.txt, for the three trees. You should run a "diff" on the three

  - output files to make sure your Observer pattern worked correctly.
  - In your README.md, briefly describe how the observer pattern has been implemented.
  - Boundary conditions:
    - Validate that the command line arguments are as expected.
    - If the input file is empty, the program should not crash. The output files should be empty.
    - Command to check for missing command line input will be:
      ant -buildfile studentCoursesBackup/src/build.xml run -Darg0=input_file.txt
    - Command to run the code correctly will be:
      ant -buildfile studentCoursesBackup/src/build.xml run -Darg0=input_file.txt -Darg1=delete_file.txt -Darg2=output1_file.txt -Darg3=output2_file.txt -Darg4=output3_file.txt"
    - The correct command is mentioned in README.md in git.

Code Organization

- Your directory structure should be the following:

```
firstName_lastName_assign_2
----README.md
----studentCoursesBackup
--------src
------------build.xml
------------studentCoursesBackup
----------------driver
--------------------Driver.java
----------------util
--------------------TreeBuilder.java [Optional]
--------------------Results.java
--------------------FileProcessor.java
--------------------FileDisplayInterface.java
--------------------StdoutDisplayInterface.java
----------------myTree
--------------------Node.java
--------------------SubjectI.java
--------------------ObserverI.java
----------------other packages and classes that you need
```

Submission

- Same as assignment-1.

General Requirements

- Start early and avoid panic during the last couple of days.
- Separate out code appropriately into methods, one for each purpose.
- You should document your code. The comments should not exceed 72 coloums 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.).

Late Submissions

- The policy for late submissions is that you will lose 10% of the grade for each day that your submission is delayed.

Grading Guidelines

Grading Guidelines.

mgovinda at cs dot binghamton dot edu
Back to Programming Design Patterns