Homework #3

RELEASE DATE: 05/21/2021

Red Correction Date: 06/06/2021 05:45

Blue Correction Date: 05/31/2021 17:45

DUE DATE: 06/18/2021, BEFORE 10:00 ON GITHUB CLASSROOM

QUESTIONS ARE WELCOMED ON THE NTU COOL FORUM.

You need to upload your submission files to the github repository under the exact guidelines that will be provided on NTU COOL.

Any form of cheating, lying, or plagiarism will not be tolerated. Students can get zero scores and/or fail the class and/or be kicked out of school and/or receive other punishments for those kinds of misconducts.

Discussions on course materials and homework solutions are encouraged. But you should write the final solutions alone and understand them fully. Books, notes, and Internet resources can be consulted, but not copied from.

Since everyone needs to write the final solutions alone, there is absolutely no need to lend your homework solutions and/or source codes to your classmates at any time. In order to maximize the level of fairness in this class, lending and borrowing homework solutions are both regarded as dishonest behaviors and will be punished according to the honesty policy.

You should write your solutions in English. We do not accept solutions written in any other languages.

We hope to remind everyone again that submitting code that cannot be compiled (for whatever reason) or providing no/incorrect commit ID for late submissions cause TAs extra burden in grading and will result in penalties!!!

This homework set comes with 200 points and 20 bonus points. In general, every homework set would come with a full credit of 200 points, with some possible bonus points.

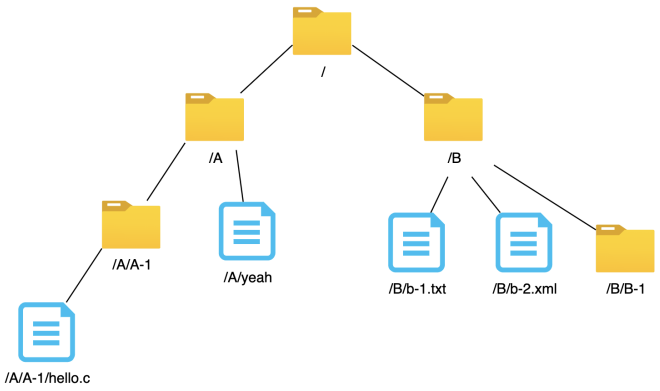
Overview

A file system controls how data is stored and retrieved, and is an essential part of almost every operating system. The operating system often helps us interact with the file system via a command line interface (CLI). In this homework, you are asked to write a simple CLI to interact with the file system.

Command Line Interface to the File System

1. A file system can be represented by a rooted tree. The node of the tree can either be a directory or a file. Each directory is a sub-tree that contains zero or many children nodes—that is, zero or many sub-directories and/or files; each file is a leaf node that stores some contents. The root of the tree is always a directory, which is commonly called the "root directory."

   The following diagram illustrates a file system:



The root directory is commonly named by a slash "`/`". There are five directories (named "`/`", "`A`", "`A-1`", "`B`", "`B-1`") and four files (named "`hello.c`", "`yeah`", "`b-1.txt`", "`b-2.txt`") in the file system. The root directory "`/`" contains two directories, named "`A`" and "`B`"; the directory "`B`" contains two files named "`b-1.txt`" and "`b-2.txt`", and one directory named "`B-1`". We call "`/`" the parent directory of "`B`"; similarly, we call "`B`" the parent directory of "`b-1.txt`".

Based on the hierarchy of the tree, we can write down the unique path from the root directory to the node as a string. For instance, the path of "`A-1`" is "`/A/A-1`"; the path of "`b-1.txt`" is "`/B/b-1.txt`", as shown under every icon in the diagram. Note that the path of the root directory is simply "`/`".

2. The CLI repeatedly accepts the user's commands line by line until the user exits. The user enters the CLI with a root directory with no children as the **current directory**. Then, the CLI executes each command within the current directory. Some commands just read out the state of the file system; other commands can cause the file system to change its state (such as adding a new file) or the CLI to change its state (such as changing to another directory).

3. During the user's interaction with the CLI, it always outputs the path of the current directory before accepting the user's command. It helps the user know the directory within which the command is executed.

Commands (Input Format)

Every line of the input contains a command and its parameters. There are 8 legal commands: `cd`, `mkdir`, `touch`, `rm`, `cat`, `ls`, `search`, `exit`.

1. `cd (<child directory name>|..)`
   Change the current directory to one of its child directory (indicated by the name) or its parent directory ("`..`"). Please note that, unlike the `cd` in your favorite operating system, the `cd` here can be used to change directory to only the children or the parent, not any other nodes of the tree.

   - Illegal parameters: `<child directory name>` does not exist, or is not a directory.
   - Legal parameter (special case): `..` at the root directory, which keeps the user in the root directory without any error.

2. `mkdir <new child directory name>`
   Make a new directory (with no children) as a child of the current directory.

   - Illegal parameters: `<new child directory name>` exists as a file or as a sub-directory within the current directory.

3. `touch <new child file name> <content>`
   Create a new file with a `<new child file name>` as its name and `<content>` as its content. The content parameter is guaranteed to be non-empty.

- Illegal parameters: `<new child file name>` exists as a file or as a sub-directory within the current directory.

4. `rm <child name>`
   Remove a child directory (the whole sub-tree) or a child file indicated by the name.
   - Illegal parameters: `<child name>` does not exist within the current directory.
5. `cat <child file name>`
   Output the content of a child file, indicated by the name, in one line.
   - Illegal parameters: `<child file name>` does not exist, or is a directory.
6. `ls`
   Output the names of all children line by line in increasing alphabetic order, where characters are ordered by their ASCII values.
7. `search <keyword>`
   Search all nodes within the sub-tree of the current directory, including the current directory, and output the name of every node that contains the (case-sensitive) `<keyword>` in its name, line by line. The keyword parameter is guaranteed to be non-empty.

   The search is done recursively in a preorder traversal manner within the current directory. In the preorder traversal, a directory is visited by examining its name (outputting if needed), before any of its children is visited. Then, each children of the directory is visited in an increasing alphabetic order like `ls`.

   For example, in the diagram above, when the current directory is "`/`", `search B` should output

   ```
   B
   B-1
   ```

   and `search 1` should output

   ```
   A-1
   B-1
   b-1.txt
   ```

   and `search /` should output

   ```
   /
   ```

8. `exit`
   Stop the CLI and exit.

Guarantees

- Each `<*name>`, `<content>`, or `<keyword>` is guaranteed to be a non-empty single string composed by alpha-numeric characters (`a-z`, `A-Z`, `0-9`), the dash character (`-`) or the underscore character (_).
- The command part in the input is always guaranteed to be legal, and is guaranteed to be followed by the required number of parameters (i.e. each parameter is non-empty).
- The last command in the input is guaranteed to be `exit`.
- Please check the definitions above for the legal and illegal parameters to each command. Note that illegal parameters may exist in the input and require the CLI to output error messages.

Output Format

- Before reading each input command (including `exit`), output the path of the current directory as:

  `Current path: <current directory path>`

  where `<current directory path>` is the path of the current directory.

- If the parameter to the command is illegal, output a line of:

  `Illegal command.`

  otherwise, for `cat`, `ls`, `search`, output the lines as instructed in the input format.

Sample Input

```
mkdir h1
mkdir h2
mkdir c
cd h1
mkdir h1-1
mkdir h1-2
cd h1-1
mkdir h1-1-1
mkdir ppp
cd ..
mkdir h1-3
mkdir aaa
touch file-h hello
touch nono yeah
ls
cd ..
touch hello HHH
search h
exit
```

Sample Output

```
Current path: /
Current path: /
Current path: /
Current path: /
Current path: /h1
Current path: /h1
Current path: /h1
Current path: /h1/h1-1
Current path: /h1/h1-1
Current path: /h1/h1-1
Current path: /h1
Current path: /h1
Current path: /h1
Current path: /h1
Current path: /h1
aaa
file-h
h1-1
```

```
h1-2
h1-3
nono
Current path: /h1
Current path: /
Current path: /
h1
file-h
h1-1
h1-1-1
h1-2
h1-3
h2
hello
Current path: /
```

Java Programming

The Java Programming environment is exactly like Homework 1. We hope to clarify here that your program will be graded under Linux (though Java is supposed to be cross-platform and there should not be many issues if you developed the program under Windows or Mac). We will use the following command to compile your program.

```
javac -sourcepath src/ -d out/ src/*.java
```

Grading Criteria

- program correctness (each 20%):
  - public test case 1: cd-ls-mkdir
  - public test case 2: cd-ls-mkdir-rm
  - public test case 3: cd-ls-mkdir-rm-touch
  - public test case 4: cd-ls-mkdir-rm-touch-error
  - public test case 5: all
  - public test case 6: bonus
  - private test case 7: private-all-1
  - private test case 8: private-all-2
- Software design (40%)
- Bonus software design (20%)
  - Add a new type of node called the `link`. The link is a special file that stores a string, which stores the path to a possible directory in the file system. The link type needs to at least support the following commands (and we will just test those and check your design report for your bonus points)
    * a new command `ln <directory name> <link name>` that creates the link to any directory name in the file system, outputting

      `Illegal command.`

      if the directory does not exist, or if link name exists as a node in the current directory

      In the test data, the `<directory name>` part is restricted to a name in the current directory. For instance, in directory `/`, it is only legal to execute

      `ln A link-to-A`

      or

      `ln B link-to-B`

      but not

      `ln B/B-1 link-to-B-1`

      All other cases are illegal. But the test data only contains illegal cases where the directory does not exist, where you should output

      `Illegal command.`

      and you can handle other illegal cases arbitrarily.
    * as a parameter of `cd` to jump to the directory pointed by the link (if the directory exists), outputting

      `Illegal command.`

      if the directory does not exist (because it has been removed some time).
    * as a parameter of `rm` to remove the link (not the directory)

  * as a parameter of `cat` to show the path stored in the link

    In the test data, we require the path from the root to be shown for this part. For instance, `cat link-to-A-1` should output `/A/A-1`.

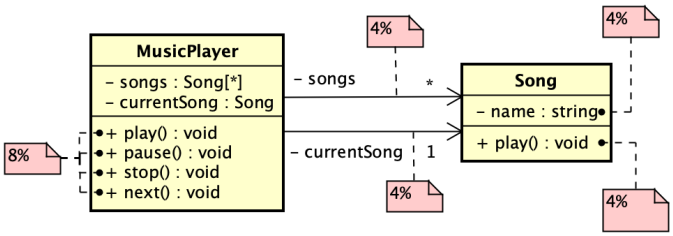
- Adding the link type of node will break the tree structure of the file system. In this bonus, you must follow the Open-Closed principle on accommodating the link nodes into your file system's structure, without any modifications to your node classes (e.g., Node, Directory and File).

Design Report

The purpose of the design report is to help the TAs grade the software design (and bonus) part by human. We hope that you can illustrate every class of your design with

- its name
- a one-sentence description of its duty

The TAs will try to grade by checking whether your design satisfies some "ideal" design choices using an internal graph of class relations like this.



We are not strictly asking you to produce this kind of graph (as we have not taught you about how to do so). But please feel free to illustrate your classes with a similar graph if you want to.

Other than those required above, please feel free to add anything that helps the TAs understand your design. But **please keep your design report as concise as possible**.

While the TAs grade with the "ideal" design choices, there is always a possibility that your design choices are better than the "ideal" ones. In this case, you can certainly check/argue with the TAs to get the points that you deserve afterwards. So please do not be too worried about whether your current design choices are "ideal" or not.

Submission Files

- your source code in the `src/` directory of your repository.
- a design report `Design-Report.md` in the root directory of your repository.

Tips from TAs

1. Though this HW3 is a simplified version, HW3 tests your awareness of the "structural variation." We all have learned some structures like a **tree** or a **graph** in computer science. Well, the "structural variation" indicates that we should apply a **polymorphism in every node** of the structure, so that we can extend our file system with various structures (e.g., mount another file system at a directory, a network attached file system or a link node) in the future. That is partially what we are challenging you to think about in the bonus part.

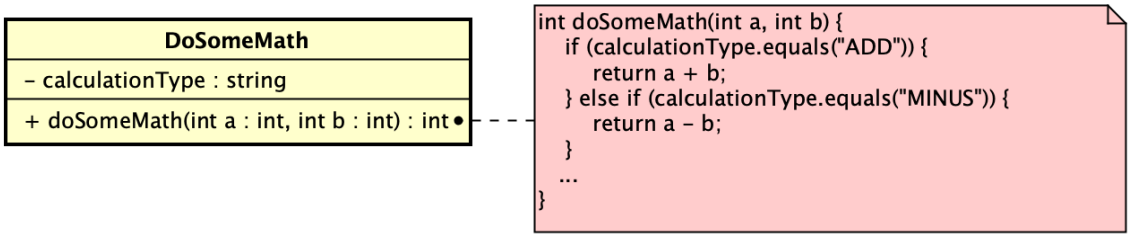
2. Refactoring steps: To accommodate the variations following OCP, you may refactor your design incrementally with the following three steps:

[a] **Encapsulate what varies**: Create a class for each behavior variant with a method encapsulating its behavior.

[b] **Abstract common behaviors**: Abstract those variants by extracting an abstract class or an interface with well-defined abstract methods that represent the common behaviors.

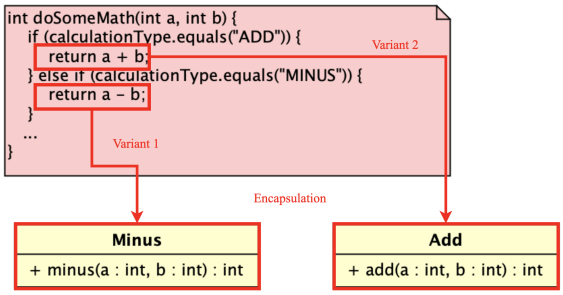
[c] **Delegation/Composition**: Delegate the behavior to the abstract class or the interface that is extracted in the previous step.

For example, you have a class `DoSomeMath` which has a variation in its math calculation method:

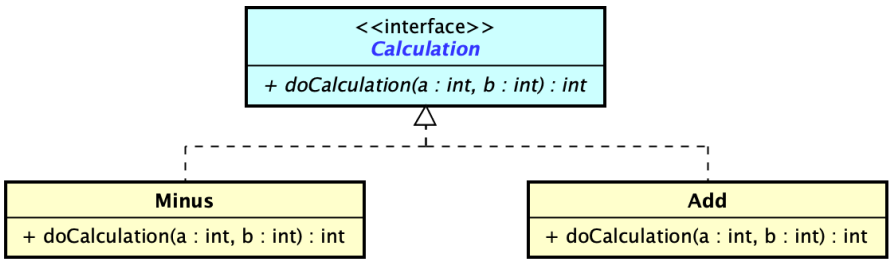


Then, you can conduct the three refactoring steps to model such variation.

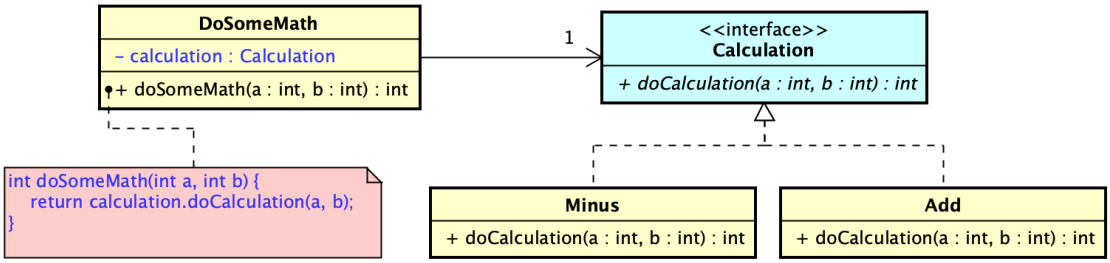
[a] Encapsulate what varies



[b] Abstract common behaviors



[c] Delegation / Composition



Other References

1. Homework Submission Guide: `https://hackmd.io/zAsrNRprQWqfI77msYQ5Bg`
2. Course Policy: `https://www.csie.ntu.edu.tw/~htlin/course/foop21spring/doc/policy.pdf` The TAs will compute the optimal usage of your gold medals after all the homework scores are announced. That is, you do not need to do anything on your side (except for remembering how many you have on hand).
3. Clarification of the Open-Close principle: `https://hackmd.io/rFlV4kBGRROFkyI1yuo6JA`