FA21 CSE 8B Homework 9:

Virtual File System (Part 2)

Due Date: Wednesday, December 1, 11:59 PM

Learning goals:

- Complete the Virtual File System using:
    - Recursion
    - Polymorphism

NOTE: This assignment should be completed INDIVIDUALLY. Pair programming is NOT allowed for this assignment.

IMPORTANT: You should NOT have to import any additional packages to complete this assignment. Any unnecessary imports may result in failure to compile on Gradescope.

Coding Style

As always, we will be enforcing the CSE 8B Coding Style Guidelines. These guidelines can also be found on Canvas. Please ensure to have *COMPLETE* file headers, class headers, and method headers, to use descriptive variable names and proper indentation, and to avoid using magic numbers. For reference, please see this mock PA with complete style. **These points are all-or-nothing, so please do NOT forget any headers or magic numbers/strings.**

Part 0: Getting started

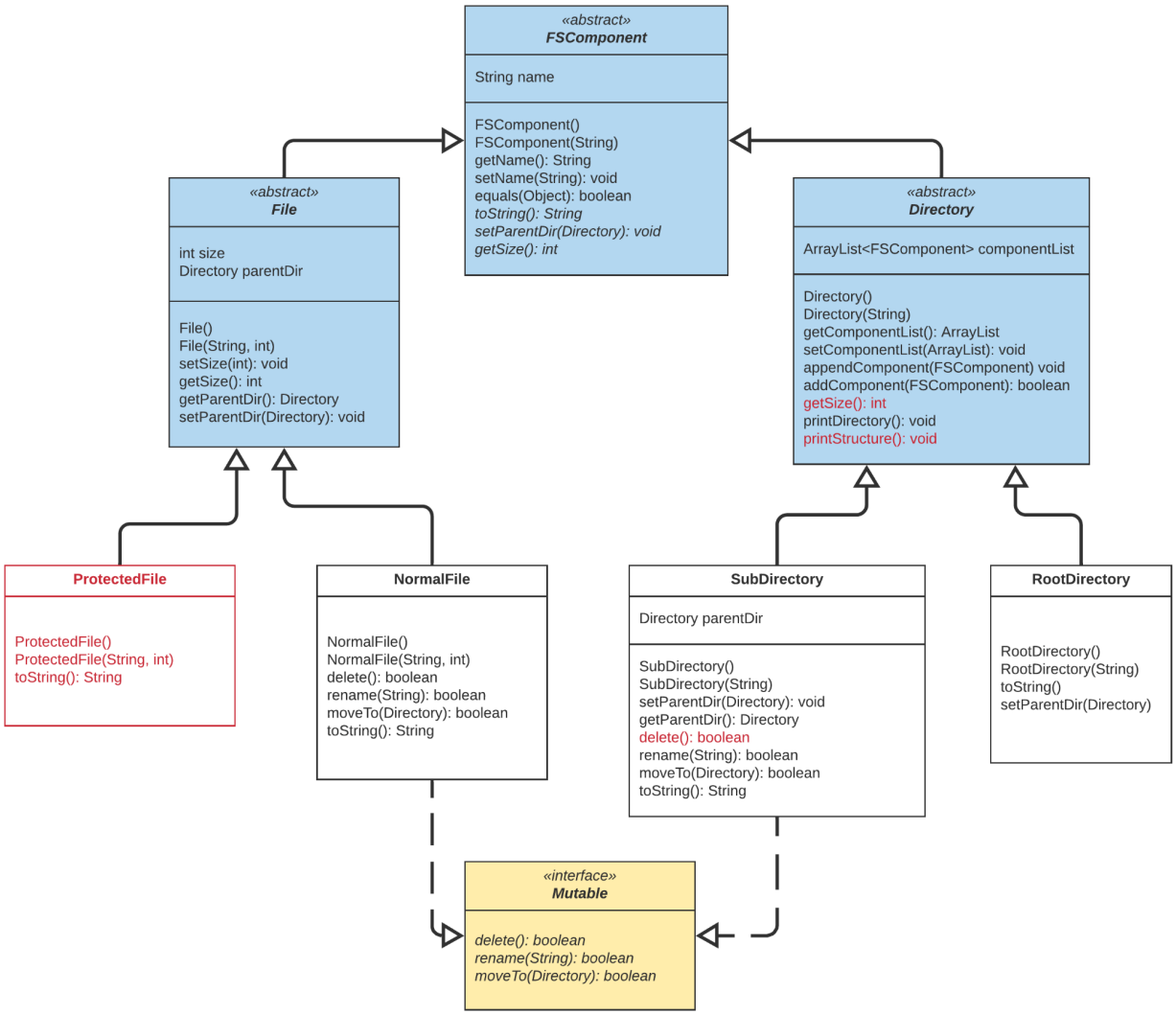
This assignment does not have a starter code. Instead, it starts with your completed code from Assignment 8. However, you must create an `Assignment9.java` file to replace the `Assignment8.java` file used in Assignment 8 (hint: copy the contents of `Assignment8.java` to `Assignment9.java` and replace all `Assignment8` with

`Assignment9`). Similar to Assignment 8, `Assignment9.java` will contain your testers. The instructions for unit tests are in Part 5. In addition, Assignment 9 requires you to create another file called `ProtectedFile.java`. The instructions for this file are in Part 2.

Part 1: Overview

Since Assignment 9 is an extension of Assignment 8, we will be reusing most of the UML (Unified Modeling Language) diagram for Assignment 8. The image below is the UML diagram for Assignment 9, showing the relationships between different classes. If the image looks blurry in the write-up, then open PA9_UML.pdf on Piazza.

There are some methods that need to be implemented or revised by you. Those methods are colored in red.



Part 2: `ProtectedFile.java`

You will have to create this file from scratch. Ensure the full file name (including the file extension) is `ProtectedFile.java`.

Like its sibling `NormalFile`, the `ProtectedFile` concrete class extends `File`. The functionality of this class will be covered in later parts. For now, add and implement the following methods:

1. `public ProtectedFile()`
   This is the default no-arg constructor. You do not need to initialize anything in this constructor.

2. `public ProtectedFile(String name, int size)`
   Implement this constructor by initializing all instance variables including the `name` and `size` instance variables in its parent class.

3. `public String toString()`
   This method should return the string representation of the `ProtectedFile` object. **To ensure full compatibility with the Gradescope Autograder, you should return the following EXACTLY: `return "Protected file: " + this.getName();`.** You do not need to worry about magic strings for this particular method.
   Do **NOT** forget the `@Override` keyword.

Part 3: `SubDirectory.java`

In Assignment 8 you implemented two constructors; setter and getter methods for the `parentDir` member variable; and `rename`, `delete`, `moveTo` and `toString` methods. All of those will not be modified for Assignment 9 except the `delete` method. **You need to revise what you did for Assignment 8 by following the instructions below:**

`public boolean delete()`
Unlike the `delete()` method you did for Assignment 8, this time the method should return false if there exists any `ProtectedFile` object under the current directory, or under subdirectories, sub-subdirectories, etc. That means for the figure below, you cannot delete any of the subdirectories. **You may need to write a recursive helper method to check the existence of `ProtectedFile` instances.**
If no `ProtectedFile` instance is found, then implement two-way binding by removing the `SubDirectory` from its parent and set the parent of the current `SubDirectory` to null.

You can always assume that the parent exists.

```
Root Directory: HOME
        Sub Directory: folder1
                Sub Directory: folder2
                        Sub Directory: folder3
                                Protected file: .config
```

Hint: All you need to do is add a check for protected file existence using your recursive helper before the code you wrote for Assignment 8.

Part 4: `Directory.java`

There are two methods in the Assignment 8 starter code that we stated would be implemented in Assignment 9. Implement them using the following instructions.

1. `public void printStructure()`

   This method will print out the all files and directories under the current directory hierarchically. This includes not only all elements in the `componentList`, but also all components in any subdirectories, sub-subdirectories, etc., if there are any. The printing format is shown below. Whenever digging into a deeper level subdirectory, add a tab to the front of the elements that need to be printed (**Hint**: the `toString` method of all concrete classes is already provided to you). The components of the directory must be printed in the linear order of the `ArrayList` (from the first element to the last element).

   ```
   Root Directory: HOME
           Normal file: cat.png
           Sub Directory: secret
                   Protected file: .cse8b_final_exam
                   Sub Directory: music
                           Normal file: rice.mp3
           Sub Directory: lecture
                   Normal file: Lecture10.mp4
   ```

   You must use recursion to implement this method. Creating a recursive helper method is HIGHLY recommended. Use `print` or `println` to output contents; use `"\t"` for encoding a tab character.

2. `public int getSize()`

Calculate and return the sum of all file sizes under the current folder (including all files in subdirectories and sub-subdirectories etc).
You must use recursion to get the file sizes in subdirectories, in sub-subdirectories, etc.

Part 5: `Assignment9.java`

You will have to create `Assignment9.java`. As always, this file contains all of your testers, a method `unitTests()` to run all of your testers, and the `main` method to run `unitTests()`. When writing testers, you can either use assertion as we did for Assignment 7 or use boolean as you did in the previous assignments. Just make sure your testers are consistent across this assignment; no mixing between the two. **To get full credit, create *at least 5* tester *methods***. It is OK to just use print to test your `printStructure()` method. **NOTE: The Gradescope Autograder will be reading the output from `printStructure()` to ensure correctness, so make sure that you do NOT leave any debug statements inside `printStructure()`. Otherwise, it is OK if your tests print to standard output.**

Submission

VERY IMPORTANT: Please follow the instructions below carefully and make the exact submission format.

1. Go to Gradescope via Canvas and click on `PA9`.
2. Click the DRAG & DROP section and directly select the 4 required files (`Assignment9.java`, `Directory.java`, `ProtectedFile.java`, and `SubDirectory.java`). Drag & drop is fine. **Please make sure you don't submit a zip. Make sure the filenames are correct.**
3. **You can resubmit an unlimited number of times before the due date.** Your score will depend on your final submission, even if your former submissions have a higher score.
4. The autograder is for grading your uploaded files automatically. Make sure your code can compile on Gradescope. You will receive 0 on the Autograder section if your code does not compile correctly on Gradescope.

> **NOTE: The Gradescope Autograder you see is a minimal autograder.** For this particular assignment, the Gradescope Autograder will only show the compilation results and the results of a few testers. After the assignment deadline, a thorough Autograder will be used to determine the final grade of the assignment. **Thus, to ensure that you would receive full points from the thorough Autograder, it is your job to extensively test your code for correctness via `unitTests`.**

Q&A

What if I did not get full points on the delete() method from SubDirectory.java in PA8?
You might want to fix the bug from your `delete` method of `SubDirectory.java`, because the same bug can happen to your `delete()` method in this programming assignment.

What if I did not get full points on the files that are not required to submit? Will I be penalized on PA9?
You will not get penalized for the mistake on those files that are not required to submit. We will use our solution code for `FSComponent.java`, `File.java`, `NormalFile.java`, `RootDirecoty.java`, and `Mutable.java`. This is also the reason why we do not ask you to submit those files on Gradescope. So you can assume for the methods from PA8, the solution code follows the writeup correctly. However, you need to understand those methods from PA8 in order to do PA9.

Can I get a copy of the solution code for PA8?
We cannot release the solution code for PA8. If you need help on your PA8 code, please visit any lab hours or set up appointments with any tutors/TAs.