AP(IT), Assessed Exercise 1

Jan 2025

Description

- **Title**: Composite Pattern File System
- **Deadline**: 4:30pm on February 13th 2025
- **Contribution to final course mark**: 25%
- **Solo or Group**: Solo work

Specifcation

You are to build a simplified file management system (only the meta-data) using the **composite design pattern**. Within your system, you can store files and directories. Directories can include other directories and/or files. All files and directories are within one top directory. The file system needs to be able to do the following:

- Add a file/directory to a given directory, using method `void add(Component)`.
- Remove a file/directory from a given directory's direct children, using method `void remove(Component)`.
- Display a directory and all of its contents: name and size of each of its contents, whether file or directory.
- Get the size of a directory as the total size of the files within it.
- Search for a directory containing a file with a certain name. If more than one is found, only the first directory is to be returned. File name is to be matched exactly and is case-sensitive. You can only search for file names and not directory names.

You are **required** to use the following interface:

```
public interface Component {
    public String getName();
    public int getSize();
    public int getCount();
    public String display(String prefix);
    public Component search(String name);
}
```

Example:

```
root: (count=5, size=886)
    Settings (10)
    music: (count=1, size=120)
        ringtone (120)
    pictures: (count=3, size=756)
        personal: (count=2, size=335)
            family-holiday (201)
            portrait (134)
        misc: (count=1, size=421)
            wallpaper (421)
```

In the above example, there are 5 directories (`root`, `music`, `pictures`, `personal` and `misc`) and various files. *Please carefully note the format of the output including spacing and punctuation*: the filesize in brackets next to each file, the directory summary in count of files and total size, and that directory contents are indented with the `prefix` (e.g. "`\t`"). Note that directories do *not* count towards the count or size; only files do.

Using the previous example, the following search parameters and their results are given:

- `search("Settings")` -> return: reference to the `root` directory
- `search("portrait")` -> return: reference to the `personal` directory
- `search("new-photo")` -> return: null reference
- `search("settings")` -> return: null reference

Tasks

Note that your code will be automatically marked so you need to carefully follow the instructions below.

- Start by understanding the given interface as you will need to implement it.
- Sketch out the two concrete classes that you will need to build.
- Create the components. These should be named **exactly as follows**: `File` and `Directory` (case-sensitive). Remember to ensure that directories should be able to include other directories.
- Create a driver class: i.e. another class with a main method to test your system. This class will *not* be marked. We provide an example of this for your own testing purposes, but different drivers with unseen input will be used for marking. They will use similar method calls, so make sure the provided driver **without editing** runs with your code. If the provided driver class fails, this means that **you will be marked down** so make sure you know what's wrong.
- Make sure that the output from your `display` method matches the specification above and similar to the given example. Otherwise, the automated marker might not identify your output as correct. The marker will identify changes in whitespace, but will deduct marks accordingly. **This is the part where most students drop marks, so please make sure you match the expected format: punctuation and spaces (including line breaks).**
- Add all your classes to a package called `file1234567`, where 1234567 is your student ID, ignoring the letter at the end. This means that all 3 java files you submit should have a package line at the top. If you do not follow this, you will be deducted 15%.

Note that you are **only** allowed to import from `java.util`. Other imports will penalised by 30% of the total marks. To be clear, you **DO NOT** need to import from `java.io` or any other namespace.

Submission Steps

Please follow these steps **exactly** or **YOU WILL BE MARKED DOWN**.

- We expect from you **3** files as described next (**note that Java class names are case-sensitive**): `File.java`, `Directory.java`, and the given interface **unedited** except for adding your package name. Any other change will be penalised. Do **not** submit your test driver class.
- Place your Java source files (i.e., `.java` NOT compiled `.class`) in a folder called the name of your package. If you submit `.class` files, **you will receive no marks**.
- Any file structure different from the one described above (e.g. /bin /src ...) will NOT be accepted.
- For example, if your GUID is 1234567, then your folder will be of the same name, as illustrated below.

- Name the components as "File" and "Directory" exactly. Make sure that components are named starting with a capital letter.
- Compress the whole folder into a ZIP file of the same name (i.e. `file1234567.zip`). **No letters, numbers or any other symbols should follow the GUID in the filename (penalty 30%). RAR, 7-zip or any other format will be marked down by 30%.**
- If the file is called anything else (such as "CW1-file1234567.zip" or "file1234567(3).zip") you will be deducted marks
- Submit your ZIP file through Moodle.

Marking Scheme

Your submission will be marked on a 100 point scale, broken down as indicated below. Note that there is substantial emphasis on working submissions, as reflected by the large fraction of the points reserved for this aspect. It is to your advantage to ensure that whatever you submit compiles and runs correctly.

- 25: Compilation
  - 5: Submitted files do not compile but due to a simple error
  - 10: Submitted files compile successfully on their own
  - 10: Submitted files compile successfully with unseen test drivers (i.e. interface and public methods match the specification)
- 40: Functionality (using unseen input)
  - 10: Add files/directories
  - 10: Remove files/directories
  - 10: Correct display of file system
  - 10: Search for directory containing a filename
- 15: Definitions (use of given interface)
  - 10: Implementation of interface
  - 5: Encapsulation
- 20: Design pattern
  - 20: Designing the appropriate concrete components

Multiple penalties for not following the specification laid here will be accumulated.