Introduction

In this assignment, it was instructed to design and build a file system through the concepts taught in CSCB07, simulating a software engineering basis. This assignment has two main components “2A” and “2B”. In 2A, it was asked to build an initial file system, with a set of initial requirements. As the assignment progressed, changes were supposed to be implemented, most notably in 2B. In this report, we address the design patterns used to adapt to these changes, and the major changes from 2A to 2B.

Design patterns used and why it was used

One of the design patterns that was used is the command design pattern. The command design pattern is defined as “A request is wrapped under an object as command, and passed to invoker object. Invoker object looks for the appropriate object which can handle this command and passes the command to the corresponding object which executes the command.” (“Design Patterns - Command Pattern”, n.d.) Prior to research, when we were designing our file system, we inadvertently used this design pattern to our convenience. The abstract “Command” class was made so that every command has a contract to be instantiated and be executed. The convenience lies within our Console class; we would not know what commands are being fed to the shell, but we know that each command can be instantiated and can be executed. Thus, within Console (lines 170 - 272 in Console), we would make a command, instantiated it based off of the user’s input (using a hashtable), then execute it accordingly. This way, we do not have to worry about making a large amount of conditional statements to check which command we are bound to use.

Another pattern that was used is the Singleton Design Pattern. The Singleton Design Pattern is defined as a way to “create an object while making sure that only single object gets created” (“Design Patterns - Singleton Pattern”, n.d.). The purpose of using this design pattern is so that we only have one file system; if a user were to open a seperate JShell, it would make sense for the user to have all of its contents from the session prior to the new JShell. We used this design pattern to have this feature with the JShell (lines 50-67 in FileSystem).

The composite design pattern is used when “we need to treat a group of objects in a similar way as a single object” (“Design Patterns - Composite Pattern”, n.d.). In one of our first implementations of the directory class, we initially had difficulty with containing Files and Directories in a single directory. To counter this, we said that both Files and Directories ARE-A FileSystemObject; this way, a directory can hold both types of objects conveniently (lines 36-37 in Directory). Having this simple pattern did pay off in the long run as we needed ways to traverse through the file system. This simple design pattern later paid off further when we had to adapt to new requirements like “mv” and “cp” (lines 143-159 in Cp, 84 and onwards in Mv)**.**

Changes in design from 2A to 2B

The most significant change in our design from 2A to 2B was changing the way we passed the output from each class. Instead of using System.out.println statements, we created an output queue and output class. Each command has an output queue which stores the command's output. The output class has the responsibility of handling the output and displaying it on the prompt. The purpose of this was to meet the redirection requirement. Our group needed a way to differentiate between std out and std error. Creating an output queue allowed us to store and identify if an output was std out or std error. The output class would then handle the output from output queue accordingly. This design change allows for flexibility in handling output from potential new customer requirements, while still handling output from previous requirements.

Bibliography

Design Patterns - Command Pattern. (n.d.) Retrieved July 22, 2018, from
https://www.tutorialspoint.com/design_pattern/command_pattern.htm

Design Patterns - Composite Pattern. (n.d.) Retrieved July 22, 2018, from
https://www.tutorialspoint.com/design_pattern/composite_pattern.htm

Design Patterns - Singleton Pattern. (n.d.) Retrieved July 22, 2018, from
https://www.tutorialspoint.com/design_pattern/singleton_pattern.htm

Zheng Hong, L. (2015). JUnit with System.out.println. Retrieved on June 30, 2018, from
https://limzhenghong.wordpress.com/2015/03/18/junit-with-system-out-println/