There were several design patterns that our group has used to complete Assignment 2B. We have tried to use as many patterns we learned in lecture as we could, however we have not used all of them. Here are the design patterns we used followed by where in our project we used it as well as why we chose to use it.

Singleton

JShellFileSystem <40, 46-50>

We chose to use the singleton design pattern to accommodate for the possibility of multiple JShells running. In this case all JShells should share the same file system (ie If any change in the file system like making directories is done in one JShell, the same should occur in other JShells).

Inheritance

Command, (Each sub class of Command), Output and OutputToJShell

We decided to use inheritance since all commands like ls or mkdir is a command. This allowed us to reduce duplicate code and overall made the code better.

Interface

FileSystem and JShellFileSystem

We used interfaces in our project to create an outline of what a file system can do. This allowed us to work around this outline, for we knew exactly what a file system behaviour is. If any design changes were made to file system the interface would reflect it allowing all group members to be on the same page when implementing features.

Dependency Injection

JShellDriver <40-46>, JShell <83, 95>

We used dependency injection to allow JShell to take in any type of file system so long as it behaved as a file system. This would allow the possibility of updating our file system or even switching to a completely different system.

NOTE: Bold is design pattern and Italics is location used (Class and <Line Nums>)

Abstract Class

Command

We chose to make the command super class an abstract class for a few reasons. First, there was no reason to create a generic command class. We chose not to make this class an interface although an abstract class can do everything an interface could. The reason we chose an abstract class was because although all commands would have common behaviours, there were some implementation that were same across all commands. Redirection is one example of this. All command had the behaviour of redirecting but there was no need for each command to redirect in a different way.

Factory Methods

Directory <54-64, 73-76, 86-89>

We used factory methods in our Directory class since the constructors of the directory class would create an instance of Directory in two different ways but it would take in same parameters. One way to create a directory was to create it without linking it to a parent directory while the other way is to link it to parent.

Polymorphism

JShell <49, 83, 92, 234>

We used polymorphism so that we could execute the run method of any command since all Command objects have the run method. We also used polymorphism so that we could take in any file system.

Generic and Hash Table

JShell <46, 88, 125-146, 207-240

After the lab where we covered hash tables, we decided we should use this design pattern to clean up our code and make the same code be used to execute a command. This way if a new command is introduced we wouldn't have to change anything in JShell since we know it would still work.

NOTE: Bold is design pattern and Italics is location used (Class and <Line Nums>)