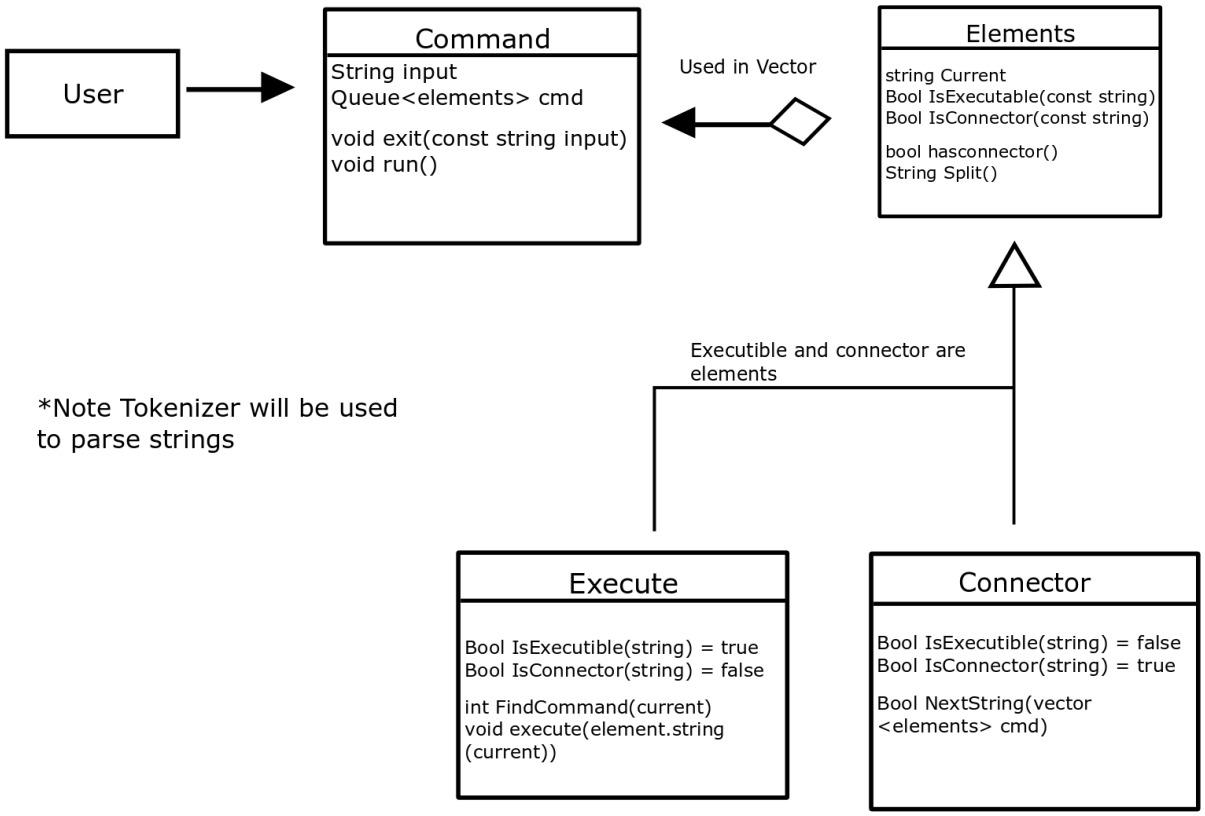
Assignment 1: Design
January 26th, 2018
Winter Quarter
Jason Mendoza - 861275834
Samuel Dominguez - 861233618

Introduction:

Though we had issues deciding whether to make the command class the base class or to implement a base class for the command to use, in the end we decided to implement the command class as a composite of the elements and subsequently the execute and connector classes. This adds modularity by ensuring that if new functions such as " cd " need to be implemented then we can either just add another class along with execute and connector or just simply create a function to elements that deals with that certain case.

Diagram:



Classes/Class Groups:

Command:

This is the overarching Class that contains the tokenizer, connector, elements, and execute classes in order complete our initial objective. It will first use the Tokenizer class to pass the string. Then use the elements class to create

a queue of elements to preserve order of operations. Afterwards it will validate whether or not it is a valid executable or connector . If it is valid function then it will call an execute command and then call the execute class version of that command to finally implement the function. Otherwise if its a connector (|| , &&, or ;) then it will call a connector function that will then chain the commands. This will continue until there is an invalid command or an endline is detected.

Elements: This class will be used to more easily create and analyze a vector. The class has elements and functions that determine if it is a connector or an executable.The Bool hasconnector() function determines if the (;) connector is embedded into the string while split() function splits the ; connector and returns it as a string to be made into an element class. Isconnector and IsExecutable are created to distinguish execute and connector classes

Connector:

We need the connector class because some of the connectors have different properties. && as well || are surrounded by spaces, but the semicolon connects is attached to the string of the previous command. The connector class evaluates each individual strings to find whether the string contains the " ; " connector or whether the other connector is after the string (&&, ||). It will have a function to check whether the next string (or boolNextString()) is followed by the connectors && or ||.

Tokenizer:

Suggested by the teacher to use for parsing strings. We will use this class to separate the individual strings which will then be put into a vector for evaluation. **THIS WILL BE USED FROM ANOTHER LIBRARY. I WILL NOT CREATE THIS CLASS, ONLY IMPLEMENT IT AND AS SUCH WILL NOT BE ON THE UML DIAGRAM.**

Execute:

If an executable is detected then it will go to the execute class which will have a function to find what executable was entered and then another function execute the bash command. If the executable is incorrect the int findcommand function will return -1.

Coding Strategy:

The command class is the essential class that we will both cooperate on to ensure that the rest of the program does not become bug riddled. Once the command class is well implemented we will attempt to both work on the connector and execute classes separately, but if problems arise that need cooperation we will change our

approach to approaching each class together. If we have to approach each class on an individual basis, we will work on individual functions of that class while attempting to reach an agreement as to how many private, public and protected variables or functions we need. To implement the coded segments, we plan on having a day where we are both free to discuss the individual implementations and how they meet the purpose of the class or function, and their efficiency. If we reach an agreement as to their purpose and efficiency, we will then work together to integrate each segment into the final version of the command class. Once the command class is “Finished” we will implement various test cases that try to devalidate our construction. Once we are confident in our test cases and the code is functional and efficient then we are truly finished.

Roadblocks:

The main issue I believe we will have is figuring out how to execute the Bash commands from our own implementation in C or C++. There are so many executables in /usr/bin that we can’t possibly implement each and every single individual case, so we have to find a single implementation that works for every case.