Assignment 1: Design

February 2nd

Winter Quarter 2017

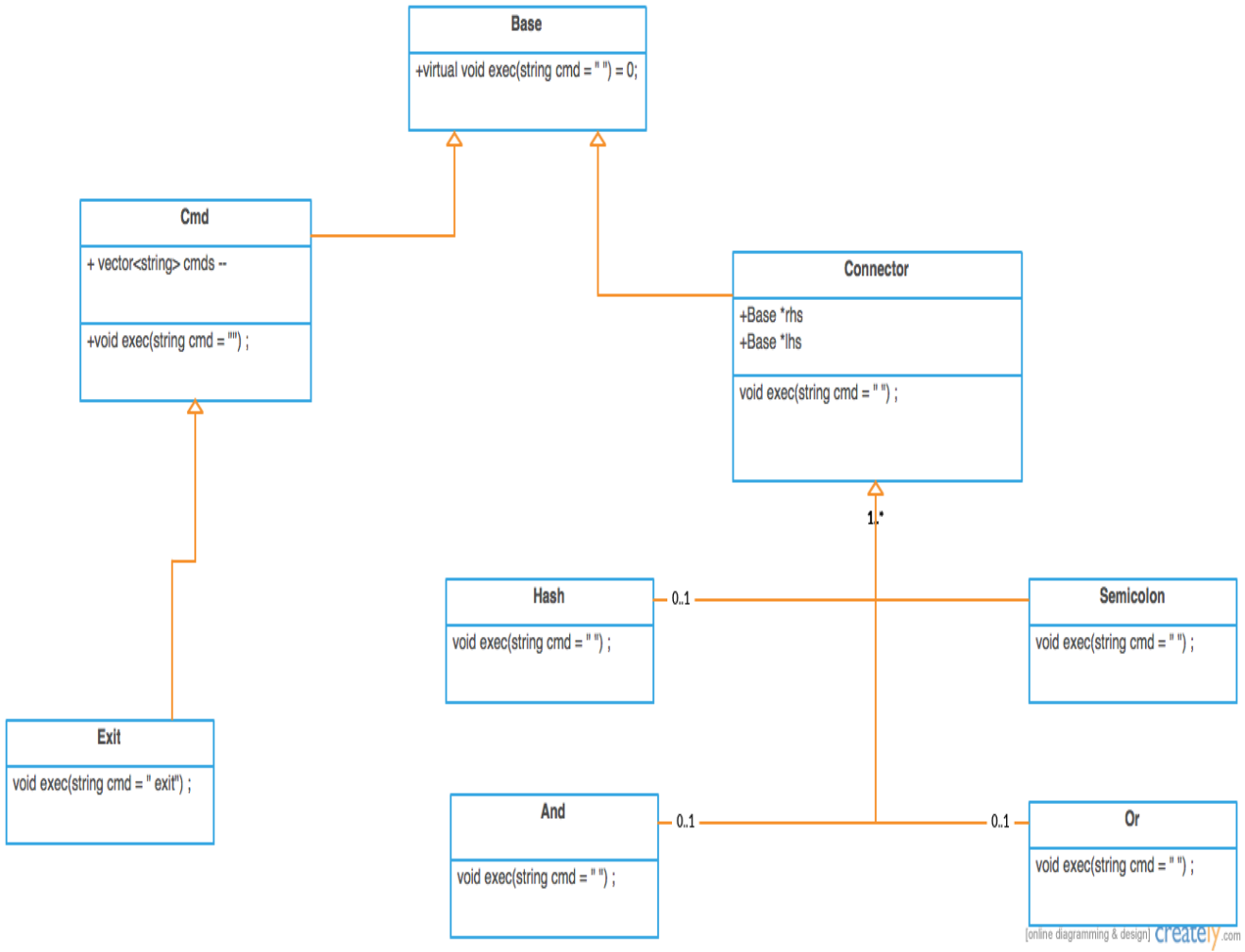
Taimaru Provensal: tprov001

Arav Batra: abatr002

Introduction:

In this assignment we will be making our own shell named “R’shell”. It will be able to take in commands from the user like a normal shell/terminal would such as “ls”, “cd”, “mkdir”, “echo”, etc.... Furthermore to simplify things the user will be able to enter multiple commands on the same line such as “cd Users/user_name/desktop/cs100; mkdir assignment1”. What that command will do will save the user time from having to type in those two commands separately and he’ll be able to go to the right directory as well as make a new one within that directory. The commands that the user gives us will either be separated by a semicolon (;), two lines (||), or two ands (&&). We will be explaining each one’s specifics in the design class design below. To accomplish this, we will be using a variety of systems as well as composite and strategy design patterns. Finally, our shell will have a special command that will exit the shell.

UML Diagram:



Classes/Class groups

Base:

This is our abstract base class that contains a pure virtual exec() function which serves to execute the commands and the connectors code. The exec() function contains a string parameter that passes a command which specifies which command to execute. If nothing is passed in it is assumed a connector is present.

Cmd:

This is the command class and contains a vector of strings that contains the command that needs to be executed.

Exit: This inherits from the Cmd class. This specific command will exit the shell once the execute function is called.

Connector:

This class holds two Base pointers: right and left. These pointers serve to access commands on either side of the connector to determine whether or not it needs to execute.

Hash: Considers everything left of it a comment and does not execute.
And: Executes only if the right command successfully executes.
Or: Executes only if the right command fails.
Semicolon: Executes the next command regardless.

Coding strategy:

We will first be tackling two commands first. We can first test each command on it's own to make sure that they are all correctly working. Then once we have those two commands fully tested and running, then we can start making use of the connectors and testing their functionality. Hypothetically once all the commands are up and all the connectors are up we can test various combinations and certain edge cases to make sure that they work in unison fully. Lastly we will be making the exit command to exit from the shell.

Coding Roadblocks:

Enabling the user to chain commands with the use of the connectors could prove to be quite difficult when parsing the user's input. Furthermore, my partner and I are not

so familiar with making make files with this many files therefore we will have to spend some time looking at the proper way to make those.