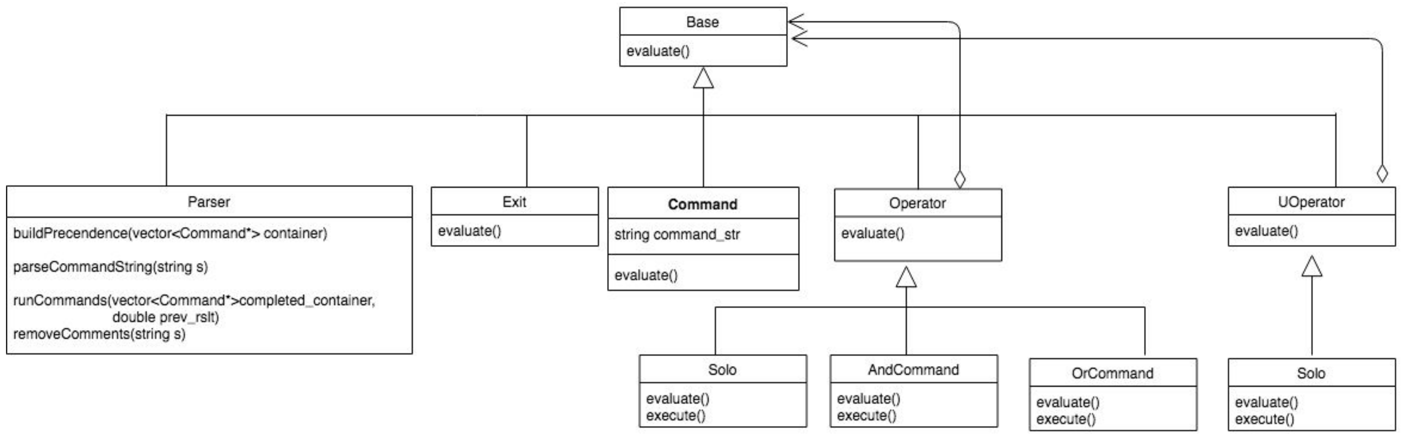
Assignment 4: Design

December 13, 2018

Fall 2018

By: Christopher Gentibano & Jason Jewell

Diagram:



Introduction

Our project design is composed of 9 different classes. We have the Base class, and 5 composites which would be the Operator, UOperator, AndCommand, OrCommand, and the Solo class. The Parse class is used for parsing the command line string and we have two leaves: the Exit class and the Command class. These classes are used to define and execute the command line string that is received via user input. Directly after user input is given, the Parse class is used to parse the string into a vector of command objects. It then iterates through that vector and executes the commands via the connectors.

Parse:

The parse class is used to parse the command line string into a vector of command objects. It is also responsible for handling the connectors and the different “chunks” (command objects with parenthetical precedence) by using recursion through parsing the chunks and executing them afterwards.

Base:

Our Base class is a simple implementation of the composite pattern with one virtual function “evaluate” that returns a string.

UOperator

Our UOperator class plays as a composite. This composite will be used to implement the Solo class after being passed in by the parser. This composite holds the virtual evaluate function for the Solo class to fulfill. The UOperator also has a constructor that takes in one Command object to assign a command.

__**Operator:**__

Our Operator class plays as a composite as well. This composite will be used to implement the AndCommand and OrCommand classes after being passed in by the parser. This composite holds the virtual evaluate function for the AndCommand and OrCommand class to fulfill. The Operator also has a constructor that takes in two Command objects to be assigned a command.

__**AndCommand:**__

The AndCommand is a class that is used to handle the connector “AND” and the logic behind whether or not the right most command should be executed. It has an additional “execute” function which calls execute on the command if the previous result (which is also a native variable) is a 1.

__**ORCommand:**__

The OrCommand is a class that is used to handle the connector “OR” and the logic behind whether or not the right most command should be executed. It has an additional “execute” function which calls execute on the command if the previous result (which is also a native variable) is a 0.

__**Solo**__:

The solo class handles the commands that are always required to be run. For example: every first command and every command following a semi colon are to be executed. These commands are sent to the Solo class because it does not require a previous result to determine whether the function should be executed or not.

__**Command**__:

The command class plays as a composite for the Exit class. In order for the user to exit the program, they would need to enter in an exit command. The command class also takes in a string in the constructor to assign a specific string command to the object. This class also contains an evaluate function which returns a string.

Exit

The exit class is created when the user wants to exit from the RShell. Once they type in Exit, we will call the Exit class to create the object, and runs “exit(0)” to end.

Coding Strategy:

We had to do a bit of reformatting to our parser. In order to handle the new symbols, they had to be handled differently in their own cases. Unfortunately we weren’t able to chain various commands for pipe, but we were able to get most of the commands to work correctly. We knew that the way this was parsed, and precedence took an important play for this assignment, so we went straight to learning how the new methods work so that we can implement pipe and dup with our current parsing strategy.

RoadBlocks:

The roadblocks that we ran into was that there were many different types of input necessary using pipe, >, and <. It was tough to think of all the different cases, and we were also having a hard time trying to chain all commands with piping. Implementing pipe to us was the most diffiuclt part. Learning dup, and pipe was also something we had to get used to.