Assignment 1: Design

3rd February 2017

Winter 2017

Kyle Semelka

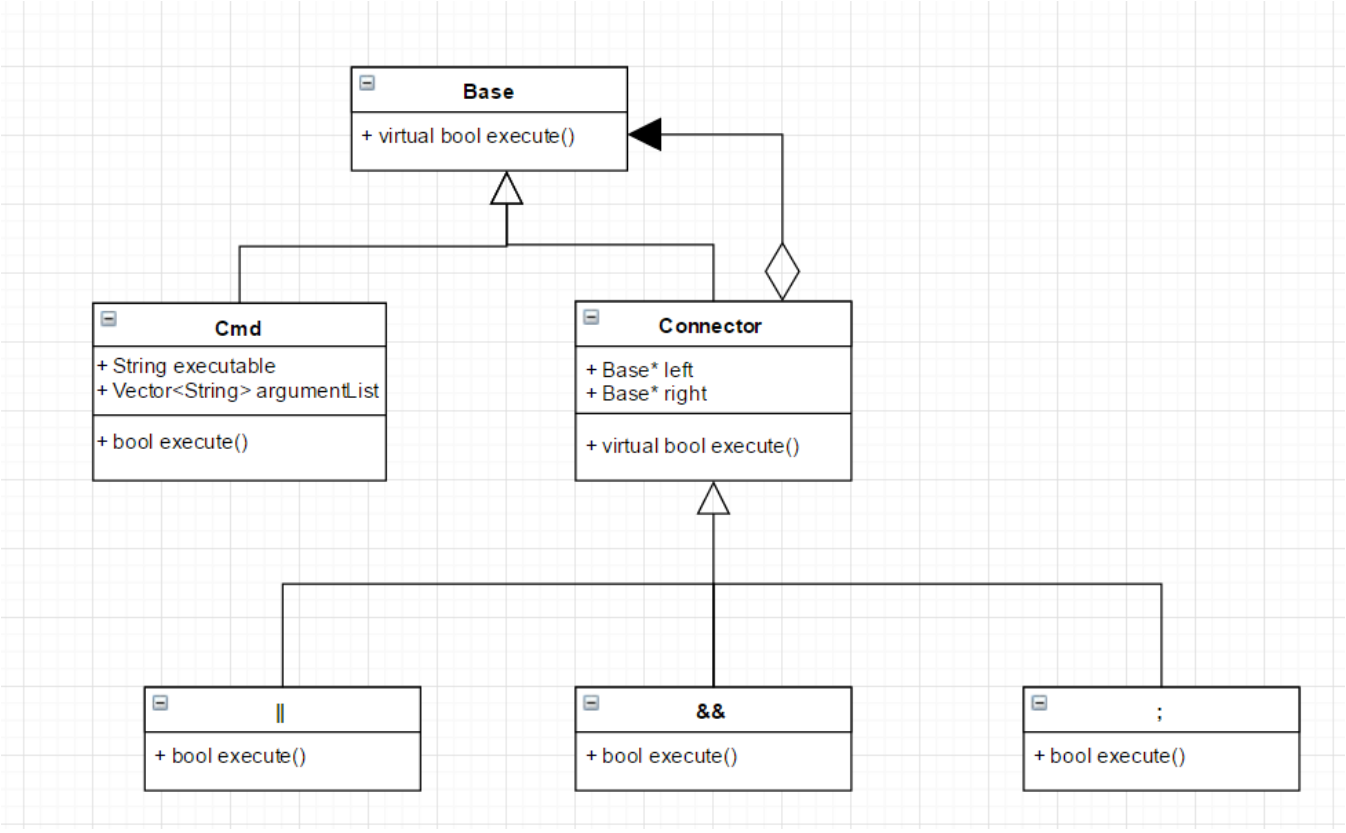
Kushagra Nikunj Singh Sachan

Introduction:

This is a design document in order to write a basic command shell. This shell will be coded in C++ and will be able to run a command prompt as well as read commands on one line (these would have the form of an executable or a connector). This program should be able to take any sized command defined from the user and be able to run it or return appropriately.

In order to create this command shell, we will be programming it around a 'Composite Pattern', which will be defined by composite class 'connector' and a leaf class of 'executable'.

Diagram:



Classes/Class Groups:

There are 6 classes in our implementation, with 1 class group.

'Base':

This is an abstract class that our following classes will inherit from.

Data:

(none)

Functions:

virtual bool execute() = 0 // Pure virtual function, all children will implement it

'Cmd':

This is the class in charge of running and storing the commands that the user inputs (excluding the connectors). This will serve as the leaf node for our implementation

Data: (using 'ls -a' as example command)

string executable // Stores the executable name (i.e 'ls')

vector<string> argumentList // Stores all the relevant arguments parsed in (i.e '-a')

Functions:

bool execute() // Runs the relevant executable with the correct arguments and returns 1 if the command ran successfully or 0 if it ran unsuccessfully

'Connector':

This is the composite class that will serve as a class group for the connectors (&& || ;), these will inherit the data members.

Data:

Base left* // Points to the relevant command that is run before the connector

Base right* // Points to the relevant command that is run after the connector

Functions:

virtual bool execute() // Virtual function, will be implemented by the children

‘&&’:

This is a child class from ‘connector’ that will serve to determine whether or not a command after a connector should be executed.

Data:

Base left* // Inherits from parent, points to the relevant command that is run before the connector

Base right* // Inherits from parent, points to the relevant command that is run after the connector

Functions:

bool execute() // recursively calls the execute of the left operand, if it returns a 1 it recursively calls the execute of the right operand. If that returns a 1, function returns 1, else returns 0.

‘||’:

This is a child class from ‘connector’ that will serve to determine whether or not a command after a connector should be executed.

Data:

Base left* // Inherits from parent, points to the relevant command that is run before the connector

Base right* // Inherits from parent, points to the relevant command that is run after the

connector

Functions:

bool execute() // recursively calls the execute of the left operand, if it returns a 0 it recursively calls the execute of the right operand. If that returns a 1, function returns 1, else returns 0.

';':

Base left* // Inherits from parent, points to the relevant command that is run before the connector

Base right* // Inherits from parent, points to the relevant command that is run after the connector

Functions:

bool execute() // recursively calls the execute of the left operand, then recursively calls the execute of the right operand. If that returns a 1, function returns 1, else returns 0.

Coding Strategy:

We will be using GitHub so that we can work concurrently without having to meet up. We will break up the work by each completing half of the functions. Making the segments work together should not be too difficult, as the design patterns we implement should take care of that.

Roadblocks:

We expect the biggest roadblocks to be in making our functions work together to produce the complete, final functionality of the project. We will overcome this by meeting up and trying to figure out our problems together and in-person.