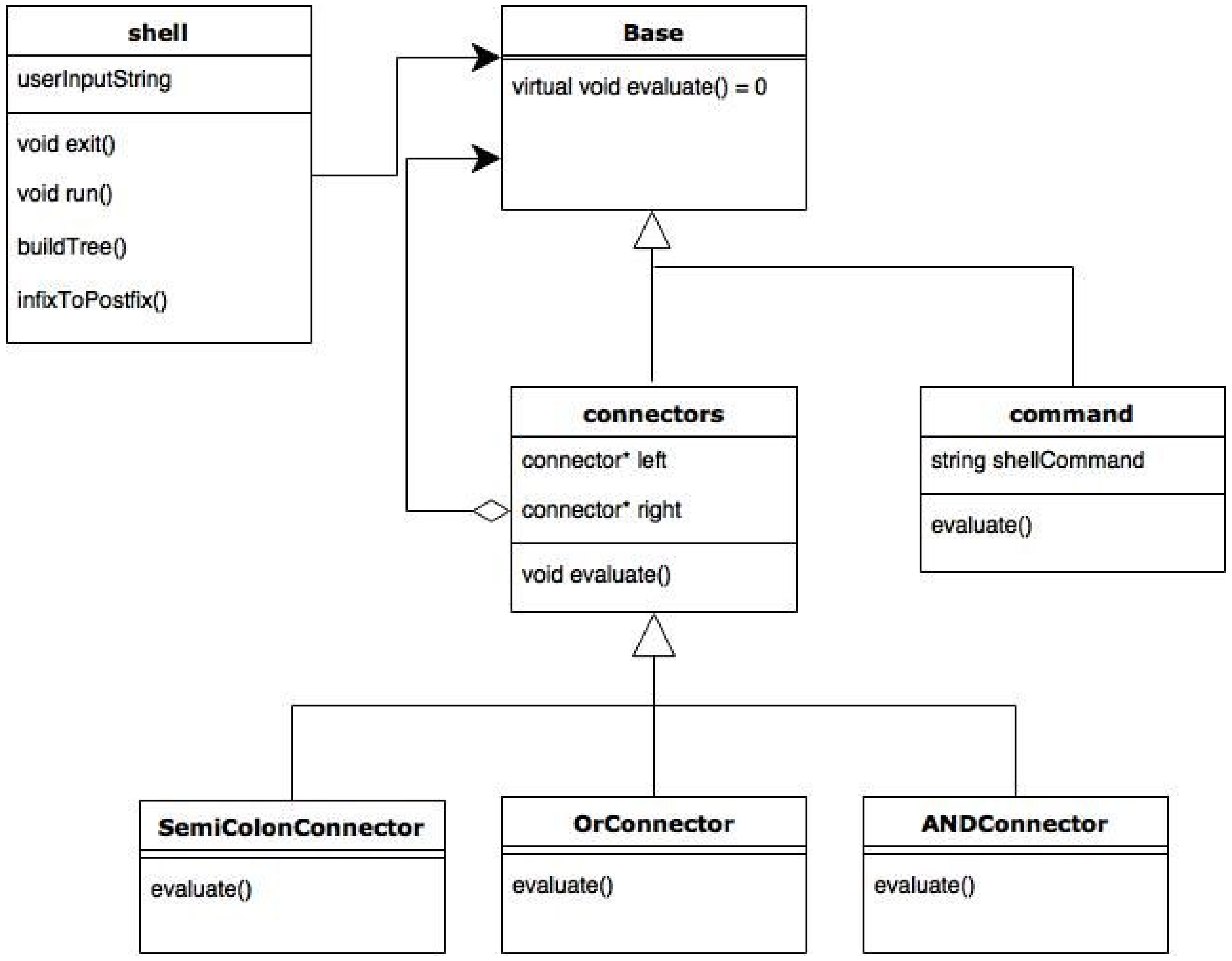
Assignment 1: Design

Cameron Morin - cmori007@ucr.edu
Andrew Smith - asmit034@ucr.edu

January 23, 2018
Winter Quarter 2018


shell
userInputString
void exit()
void run()
buildTree()
infixToPostfix()
Base
virtual void evaluate() = 0
connectors
connector* left
connector* right
void evaluate()
command
string shellCommand
evaluate()
SemiColonConnector
evaluate()
OrConnector
evaluate()
ANDConnector
evaluate()

Introduction:

In our design we use a composite design pattern to implement a bash shell command line. Our composite component is the connectors class which will allow us to build an arithmetic tree and to keep the order in which we need to operate the commands read from the user's input. The Base class enables the use of single evaluate function for all of the connector classes and the command class. This is the main focus of the design so we can have a streamlined interface that is easy to understand and quick to reimplement.

Classes/Cass Groups:

We have 7 different classes: Shell, Base, Connectors, Arguments, SemiColonConnector, ANDConnector, and ORConnector.

- Shell Class is our main class that has the main loop function that reads in the user input and makes the function calls to the other classes to build and implement the expression tree.
- Base Class is the base for all other classes to come. It has the basis for both the command and connectors classes.
- Connectors Class is a class derived from the Base class, and it involves the three different connectors: AND, OR, and SemiColon. These classes all evaluate the different connectors, and help to create our arithmetic tree.
- Command Class is our class that evaluates our commands and places them in the leaves of the tree. This class also make the bash function calls to the corresponding functions to be executed, as requested by the user.

One class group that we have would have to be our Connector Class and its derived classes because all of its derived classes are just more precise in what connector it actually is.

Coding Strategy:

We will be dividing our coding up into two sections:

- Cameron will be implementing the tree-building portion of the code;implementing the buildTree() function and evaluating the input string and breaking it down into an arithmetic expression tree.
- Andrew will be implementing the input evaluation portion of the code that Cameron will be using to build the tree. Andrew is going to build the functions in the Shell class that reads in the string and determines how to connect it to the rest of the program, and also the functions that call the command functions inside of the bash terminal.
- Together, we will integrate the tree and and the input coding portions in order to successfully implement our own shell program.

Roadblocks:

The major roadblock to implement this design is coding the correct algorithms for converting the user input from infix to postfix and to be able to create the arithmetic tree.

The implementation for the evaluating logic for each separate operator and using the new fork and execvp commands to execute the commands in separate processes.