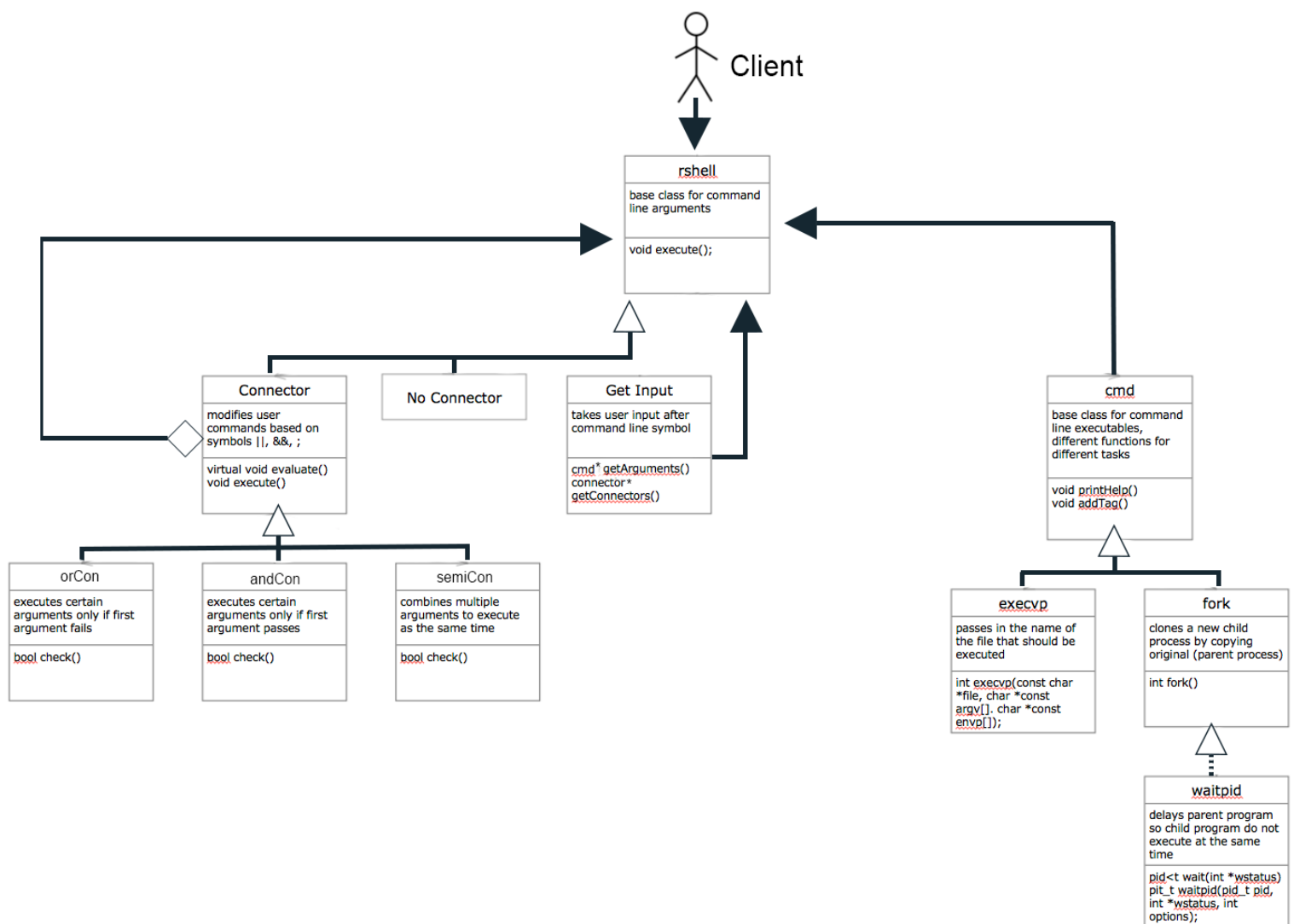# Assignment 2: Writing a Basic Command Shell

11/15/18

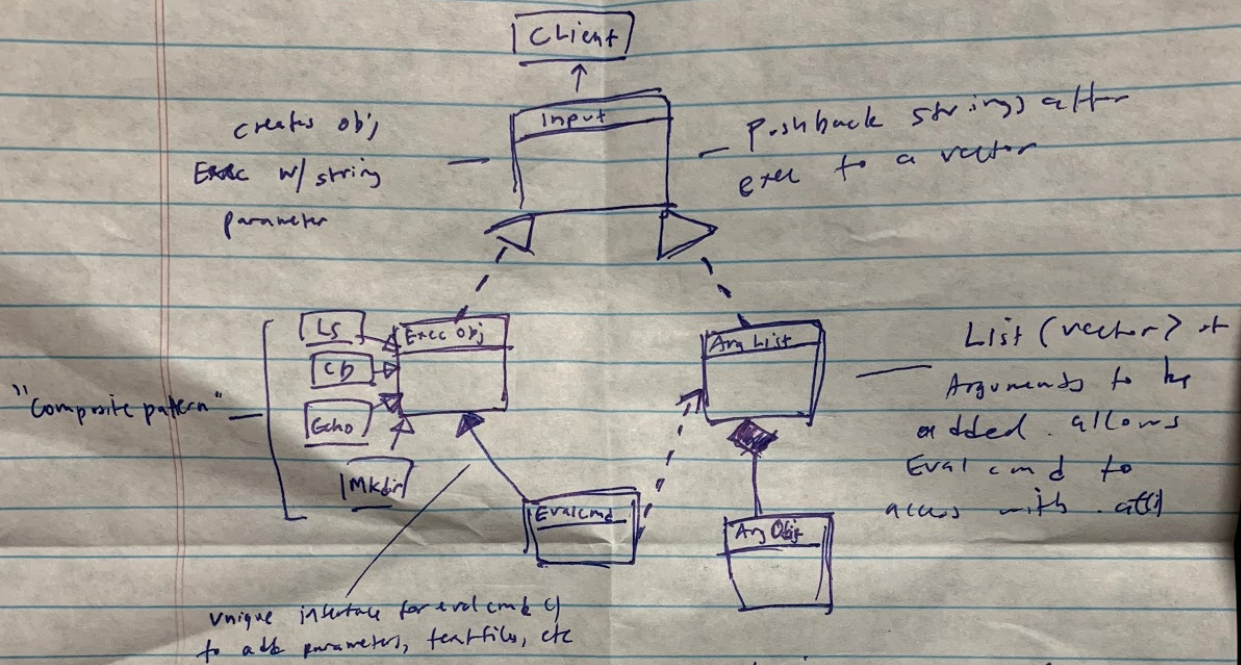Fall 2018

Raymond Kim, Lucas Song

Introduction:

For this project, we will be designing a basic command shell. This command shell will be able to print a command prompt, read in a line of user input, segment this input into an executable, argument list, connector, and more, and use said input to execute the proper commands using fork, execvp , and waitpid. The shell should be able to run multiple commands at once, utilizing symbols to determine how multiple commands should be run in unison. These include "| |, &&, or ;."

Diagram:

ASSN 2: R'shell

Client

creates obj
Exec w/ string
parameter

Input

— Pushback string) after
exec to a vector

"Composite pattern" —

LS
CD
Echo
Mkdir

Exec Obj

Arg List

List (vector) of
— Arguments to be
added. allows
Eval cmd to
access with .at()

Evalcmd

Arg Obj

unique inheritance for eval cmd ()
to add parameters, textfiles, etc

mkfile (vim) → & string (getline (cin, string)

mkdir → vector <string>

cmd: Execobj → evaluate (Arg List<string> a)

Classes/Class Groups:

Input:

- **Input();** Constructor
- **~Input();** Destructor
- **Input(string userString);** Stores input from user into userInput
- **void clearInput();** Clears input and vectors for next input
- **vector<baseExec* > returnParsedExec();** Returns parsedExec vector (helper function)
- **vector<vector <string> > returnParsedNoSpace();** Returns parsedNoSpace vector (helper function)
- **vector<string> returnStrings();** Returns parsedStrings vector (helper function)
- **vector<Connector* > returnConnectors();** Returns connectors vector (helper function)
- **void callExit();** Sets bool exit = true
- **bool checkExit();** If bool exit == true, quits command prompt
- **void runInput();** Prints command prompt and takes input from user
- **void parseInput();** Takes in user input and tokenizes string into substrings based on connectors, and then tokenizes yet agian to remove spaces in the substrings. Returns a vector of strings to be then used to generate argument objects.
- **vector<string > parseSpaces(string withSpaces);** This is a helper function that is virtually the same as the first step of parseInput(), using spaces as delimiters instead of the connectors to remove spaces from string withSpaces
- **void parseConnectors();** Takes connectors from userInput and pushes into vector of connector objects
- **baseExec* makeExec(vector<string> exec);** Helper function that returns a baseExec object based on the zero index of the vector passed in and takes the rest of the vector as the argument list
- **void makeExecutableTree();** Constructs executable tree based on 2D vector and connector vector, sets baseExec* head to the head node of tree.
    - head->eval should execute the tree with respect to the connectors
- **void callExecute();** Calls execute on the head node
    - Essentially recursively calls execute for elements of tree with respect to connector conditions
- **string returnUser();** Returns name of user (Extra Credit)
- **string returnHost();** Returns name of host (Extra Credit)

Arg:
- **arg();**
- **Virtual void execute() = 0;**

baseNode
- Base class for connectors and executables
- **baseNode();**
- **~baseNode();**
- **Virtual void execute();**
- **Virtual void setLeft(baseNode* leftChild);**
- **Virtual void setRight(baseNode* rightChild);**
- **Virtual baseNode* getRight();**

Connector : baseNode
- **Connector();**
- **Void setLeft(baseNode* leftChild);** Sets left child
- **Void setRight(baseNode* rightChild);** Sets right child
- **baseNode* getRight();** Returns rightChild

And : Connector
- **And();** Constructor, points leftChild and rightChild to NULL
- **Void execute();** Calls execute for left and right children ***

Or : Connector
- **Or();** Constructor, points leftChild and rightChild to NULL
- **Void execute();** Calls execute for left and right children ***

SemiColon : Connector
- **SemiColon();** Constructor, points leftChild and rightChild to NULL
- **Void execute();** Calls execute for left and right children

baseExec : baseNode
- **baseExec();**
- **Void addArg(vector<string> arg);** Adds arguments to vector

echo : baseExec
- **echo();**
- **Void addUserInput(string userInput);**
- **Void execute();** Prints arguments on newline

ls : baseExec
- **ls();**
- **Void execute();** Prints files in directory

cd : baseExec
- **cd();**
- **Void execute();** Change directory based on argument passed in

mkdir : baseExec
- **mkdir();**
- **Void execute();** Makes new directory

error : baseExec
- **error();**
- **Void execute();** Outputs error statement

Evalcmd:
- **Evalcmd();**
- **Bool EvaluateCommand();**

Coding Strategy:
We intend to split the work in such a way that we are both able to familiarize ourselves with the program. We will both implement the classes and their respective functions, however, we will test each others' code so that we are exposed to all aspects of the program and its design.

Roadblocks:
An issue that we anticipate is in parsing the user input. Since there are many ways for spaces and stray symbols to introduce problems in the program, we must ensure that the data is properly compartmentalized.

In addition, coding out the logic of connectors seems a little confusing. We are not completely sure as to how to approach the case of multiple connectors with multiple arguments that must execute in different cases.

Not only this, we infer that there will be many changes to the structure of our design. As such, keeping an accurate account of changes that have been made and will be made should allow for smoother transitions between adding new features.