Assignment 1: Design

12/13/2018

Fall 2018

CS100: Software Construction

Authors:

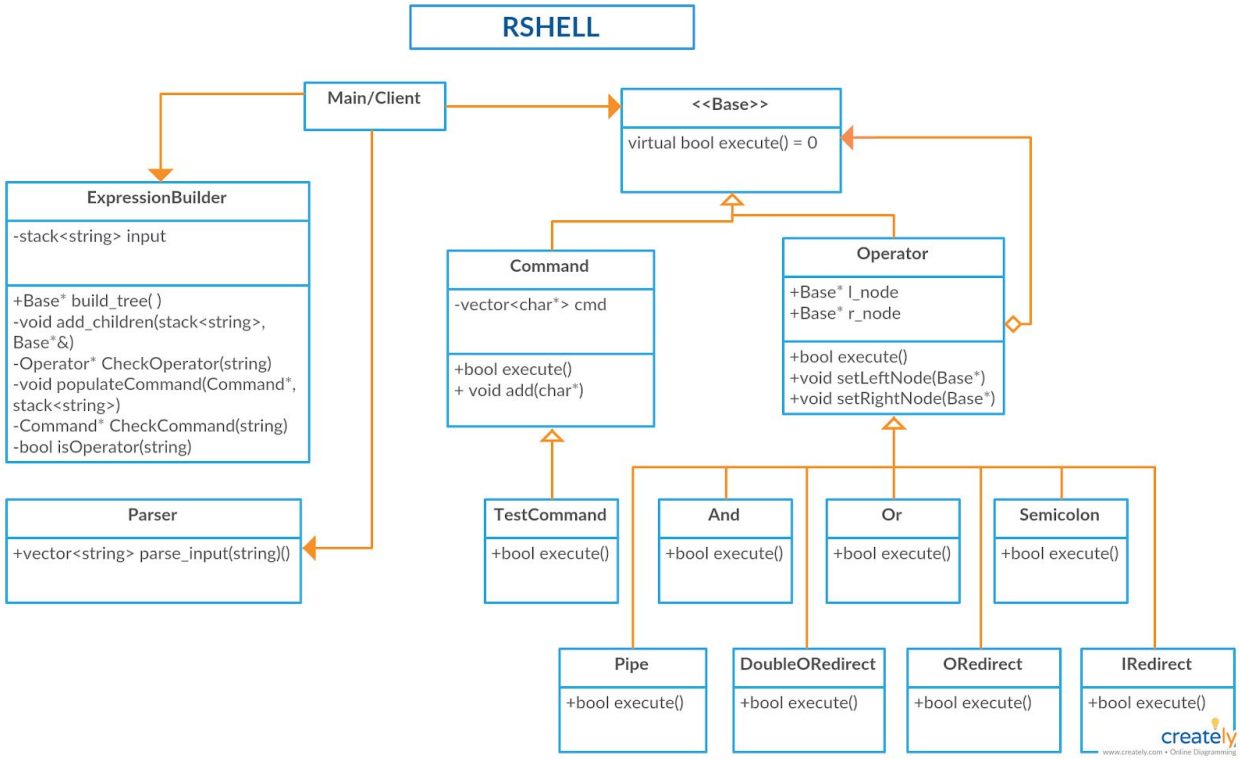
Travis Nasser

Dishon Jordan

Introduction: This is the design document for rshell. This program processes commands and returns an output similar to that of BASH in Linux. The design pattern used in this project is a *composite pattern*. We choose to implement this partitioning pattern because it allows us significant extensibility when adding new features. Commands have the following format:

```
$ executable [argumentList] [connector] [cmd] ..
```

UML Diagram:



Classes/Class Groups: We are able to achieve a highly modular program by partitioning our classes and maintaining a single interface in which the client can access.

Parser: The `Parser` class provides a means to translate the series of commands issued by the user, into manageable strings that can be handled by the `ExpressionBuilder` class. The `cstring` library is used to tokenize our input which is then stored in a `vector<string>`. The order of the commands and operators is

maintained by continually adding the strings into the vector as it is parsed.

ExpressionBuilder: The `ExpressionBuilder` class serves to manipulate the `vector<string>` created in `Parser`, and creates a tree of `Command`'s and `Operator`'s, implementing the composite structure outlined in the UML above. Converting the inputted `vector<string>` to a `stack<string>`, we add a string to an internal `stack<string>`, until an `Operator` or precedence operator is reached. If an `Operator` is reached we assign the strings in the temp stack to a `Command`, assign it to the right child of the `Operator` and recursively build the rest of the tree. If a precedence operator is reached, we add everything until the opening parenthesis into the stack and then we recursively built the internal tree of the parenthesis.

Base: The abstract `Base` class serves as the component class in the composite design pattern. We declare `pure virtual execute()` to be implemented in the subclasses: Command, And, Or, and Semicolon. `execute()` returns a `bool` value because we need to be able to compare whether or not the command gets executed, in order to follow the comparison rules dictated by each; `&& ||` operator.

Command: As a leaf component of `Base`, `Command` holds a `vector<char*>` of the command and arguments that will be executed. The `execute()` function uses syscalls to *execute* the user inputted command.

TestCommand: As a subclass of Command, the TestCommand class is instantiated when the user input is either `$ test 'argument'` or `$ ['argument']`. Once instantiated, TestCommand uses the stat() library to test whether the file exists. The flags `-d` and `-f` can also be used to check whether the argument is a directory or file, respectively.

Operator: As a composite component of `Base`, `Operator` serves to instantiate left and right nodes(`l_node, r_node`) that represent the command expressions on both sides of a connector. `Operator` also extends the `execute()` method to its subclasses.

And: As a composite component of `Operator`, `And`'s (`&&`) purpose is to execute the next command only if the first *succeeds*.

Or: As a composite component of `Operator`, `Or`'s (`||`) purpose is to execute the next command only if the first *fails*.

Semicolon: As a composite component of `Operator`, `Semicolon`'s (`;`) purpose is to always execute both commands.

ORedirect: As a composite component of `Operator`, `ORedirects`'s (`>`) purpose is to redirect the output stream of the rshell, writing the output from the left child into the right output file.

DoubleORedirect: As a composite component of `Operator`, `DoubleORedirects`'s (`>>`) purpose is to redirect the output stream of the rshell, appending the output from the left child into the right output file.

IRedirect:As a composite component of `Operator`, `IRedirects`'s (`<`) purpose is to redirect the input stream of the rshell, sending the output from the right child into the left output file.

Pipe:As a composite component of `Operator`, `Iipe`'s (`|`) purpose is to send the output of the right command as an input to the left command.

Coding Strategy:
Travis will be responsible for:

- Design Document
- Base
- Command
- Operator and its subclasses: And, Or, Semicolon

- Testing with Google Test

Dishon will be responsible for:

- UML Diagram
- Parser
- ExpressionBuilder
- Testing with Google Test

We believe this to be a fair strategy for approaching this project. With the above list in mind, we plan on supporting each other as much as possible. We plan on integrating our segments with git by merging branches.

Roadblocks: Some roadblocks we have considered can be listed as:

- Having a firm understanding of the syscalls.
- Implementing our exit command.
- Allocating meetup time in our busy days.
- Ensuring all cases are properly tested and working.
- Integration of both halves of the project into one.