

Assignment 1: Design

October 19, 2018
Fall 2018

Alex Chen
Kyle Tran

[Introduction](#)

[Diagram](#)

[Classes](#)

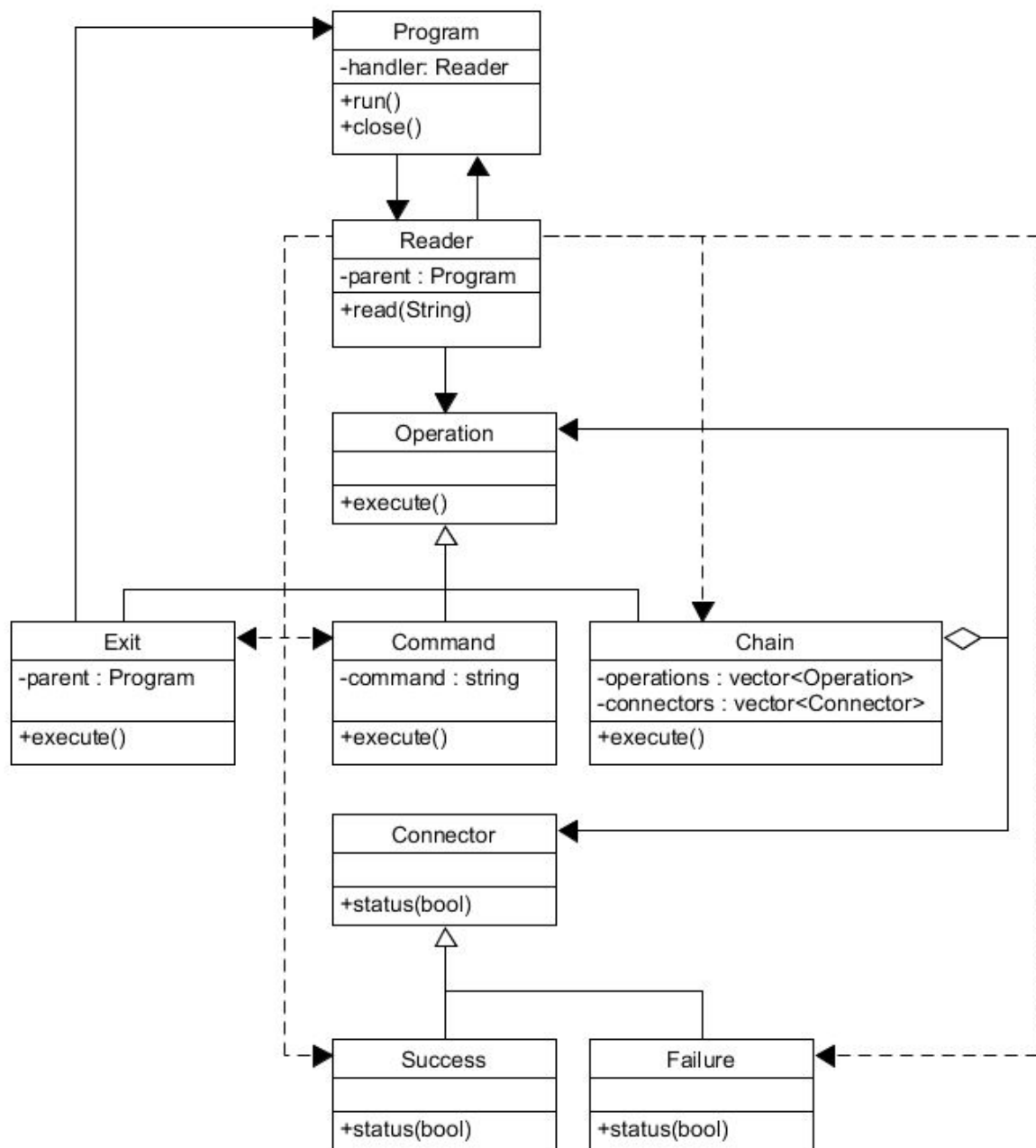
[Coding Strategy](#)

[Roadblocks](#)

Introduction

The design of our command shell uses the composite pattern to represent our commands and operators. The shell will initialize by creating a **Program** instance that will split each entered line by the semicolon into "statements." A **Reader** will parse the "statements" to identify individual **Commands** and **Chains** of **Commands** linked by **Connectors**. **Commands** and **Chains** are both subclasses of the **Operation** base class to fit the Composite pattern; the **Command** is a leaf of **Operations** and the **Chain** is a tree of **Operations**

Diagram



[Umllet File](#)

Classes

- **Program**: Prints the prompt and reads lines of commands using the **Reader**. Contains a **close()** function to be called by **Exit** for early termination. Splits lines by the semicolon into "statements" before passing them to **Reader**.
- **Reader**: Parses a statement for tokens using space as a delimiter and ignoring all input past the comment symbol. Contains a reference to the parent **Program** so that it can recognize and create the **Exit** command. Splits the line by the semicolon and then parsing each piece as a **Chain**
- **Operation**: Contains an **execute()** method that returns a **bool** indicating success
 - **Chain**: Contains a **operations vector** and **connectors vector**. When executed, it iterates through **operations** and **connectors**, alternating between executing the current **operation** and checking the status of the current **connector**, and returns a **bool** indicating the success of the last **Operation**. Serves as a composite of **Operations**.
 - **Command**: Contains a full **command String** to be executed. When executed, executes the **String** and returns a **bool** indicating success. Serves as a leaf of **Operations**.
 - **Exit**: Terminates the **parent Program** by calling **close()** and returning **true** when executed.
- **Connector**: Contains a **status(bool result)** function that determines whether or not to continue based on the **result** of the previous **Operation**.
 - **Success**: Returns **true** if the **previous Operation** succeeded (**result** is true). Returns **false** otherwise
 - **Failure**: Returns **true** if the **previous Operation** failed (**result** is false). Returns **false** otherwise

Coding Strategy

We will work on the initial program and reader together to ensure the foundation is sound. Same with the operation class so we know the functionality and agree how it should be designed. Kyle will write the Connector and Command classes. Alex will write the Program, Reader, and Chain classes. We will collaborate for the entire process however to ensure the correctness of our design. We will use separate git branches and merge when needed. With every completed feature we will commit and push. We will also split the unit test cases for our respective parts.

Roadblocks

- Parsing input. For this we will need to collaborate and think of all possible scenarios for error
- Merging our files. Make good commits so we know what has changed
- Chain class. This can get messy for iterating through commands and checking the status. For this heavy testing will be needed.
- Testing. We will need to think of every situation a user can use the program in and test accordingly.
- Using system calls. We don't have that much experience using system calls so we will need to do our research beforehand to learn the intricacies.
- Future assignments. We may have to revise the structure to fit new developments