Advanced programming – lab # 02

***“Interpreter for executing a list of declarative commands”***

Shumail mohy-ud-din

BSCS-2B | 01947

Documentation

This project consists of 4 files and 1 unit Test case file:

1. Variable.java
2. Interpret.java
3. ParseHelper.java
4. ExpressionEvaluator.java
5. Test\_Interpreter.java

GitHub Link: <https://github.com/shumail92/CS213-Advanced-Programming/tree/master/Lab-2-Interpretter>

## Variable.java

Variable class to store variables after interpretation, stores variable symbol and as well as variable value. It also offers getter and setters for class members. The class is **generic** and can hold **int, float** and **strings.**

## Interpret.java

This class holds the main (not used in case of unit tests) and provides the function of read\_file(). Read\_file() reads the specified file and passes that file to ParseHelper.java

## ParseHelper.java

This class is one of the core classes that first creates a Hash Table of type variable, and stores all the variables (symbol, value, type) in it. Then it reads the passed file stream and starts parsing it line by line. A line can lie in 3 categories:

**Case 1 : Declaration with keyword LET**

**Case 2: Printing a variable with keyword PRINT**

**Case 3: Perform some operation / evaluate some expression**

The parse() function reads the current line and scans it for the keywords to categorize the line into the mentioned cases and then performs required operation as per the case.

### Case 1

For case 1, it tokenizes from spaces, and then extracts the variable symbol and value from tokenized array. For evaluating the data type, it checks, if it contains a dot(.), then it’s float and if it contains “ or ‘ , then it’s string and integer otherwise.

Based on the data type, it creates variable of that data type and stores it into hash table.

### Case 2

For case 2, it reads the variable symbol after tokenizing and then checks the value in hashtable by passing that as the key. If found, it prints.

### Case 3

For case 2 - performing the operation, it tokenizes from =, and then for right hand side, it first replaces all the variables from their values by checking against hash table. Once replaced, the expression is checked and verified if it is valid by passing it into isExpressionValid() function. If valid, the expression is passed to ExpressionEvaluator Class, for evaluating. Once evaluated, it saves the value and updates the Hash table for the key from left hand side, and updates it’s value to the evaluated expression’s answer.

## Test\_interpreter.java

This is basically a simple unit tests that first calls read\_file() and then that itself calls parse and subsequently runs the whole program. To evaluate if test passes, I have a Flag called ASSERT\_FLAG. The value for this flag for passing test must be 0 and for failing it would be -1.

In case of any error in whole program while interpreting and running, once an error is encountered the FLAG is set to -1 and then compared at the end in assert for unit test result.

## helping information

All the events are logged on console.

The read statement is preceded by >>

Comments and info about operations on that statement is preceded by #

The output / result of expression is enclosed in << >>

The code is commented and program is logged properly on console.

2 text files are in directory:

**instructions.txt** contains all working commands and should pass unit test

**instructions\_error.txt** contains all working commands and should pass unit test

## Screenshot:

