The Interpreter
Software Documentation
Table of Content

1. Introduction ---------- page 2

1.1 Overview---------- page 2
1.2 Project Language ---------- page 2
1.3 Project Components ---------- page 2
1.4 Basic Structure (Hierarchy) ---------- page 2

2. Technical Guide
2.1. Scope of Work ---------- page 3
2.1.1 Algorithm of Interpreter ---------- page 3
2.1.2 Interpreter Development ---------- page 3

3.Testing and Execution ---------- page 4

4.Assumption ---------- page 5

5.Implementation discussion ---------- page 5

6.Results and conclusions ---------- page 6

7.Credit and Reference ---------- page 6

8. Support, Contact and FAQ ---------- page 7

9. License ---------- page 7

The Interpreter

Software Documentation

Contributor: Shan Kwan Cho
GitHub Repository: https://github.com/csc413-03-sp18/csc413-p1-ShanKwanCho.git

1. Introduction:

This project is called Interpreter project. The purpose of this project is to be able to implement an interpreter for the mock language X.

1.1 Overview:

The interpreter attempt to execute the bytecode file that contains sources of bytecodes (byte-code interpreter). That bytecodes execute to run the bytecode loader(initialization) and create the program in the virtual machine. That virtual machine leads the program to run the interpreter bytecodes from the corresponding bytecode loaders.

Note: This project is running on Mac OS High Serra and IDE is NetBeans
More info IDE of NetBeans: https://netbeans.org

1.2 Project Language

The Interpreter project is written in Java language.

1.3 Project Components:

- Technically, interpreter project includes major of six java files and a folder of bytecodes(17 byte code java files)

1. ByteCode Loader.java
2. CodeTable.java
3. Interpreter.java
4. RunTimeStack.java
5. Program.java
6. Virtual Machine.java

1.4 Basic Structure (Hierarchy):

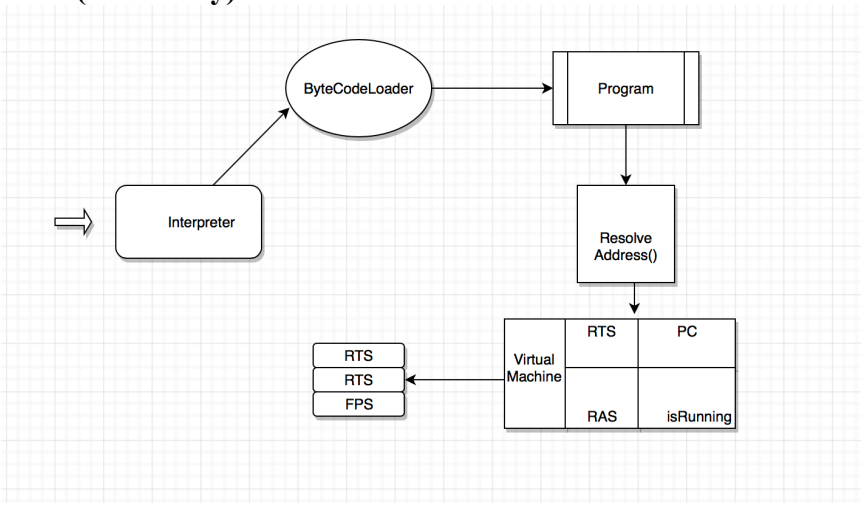

Fig 1.4.1

2. Technical Guide

2.1. Scope of Work

2.1.1 Algorithm of Interpreter

Implement the bytecode classes that involved in the projects components with correct abstractions for the bytecodes. The required classes that have to be implemented are: BytecodeLoader, Program, RuntimeStack and Virtual Machine. Importantly, every variables that needs to declared have their matched modifiers.

Last but not least and more importantly, breaking encapsulation cannot be accepted. Traditionally, virtual machine calls the individual bytecodes, run and pass by ifself as a parameter. In execution of bytecode, at least one methods has to be invoke in the runStack by virtualmachine.runStack.pop( ) which cause the break encapsulation. Instead, it can be done having a corresponding set of methods in the virtual machine. Virtual Machine can only execute passing the call to the runStack. Basically, initializing temporary variables within the virtual machine runStack.

Reference : Instruction guide for the Interpreter project

2.1.2 Interpreter Development

Interpreter.ByteCode: First of all, I implemented the bytecodes in the interpreter. Bytecode folder using HashMap method into the codeTable (halt code, pop code, …. etc).The codeTable contains collection of the particular byteCodes. Those codeTable perform creating new object ByteCodeLoader and the method loadcodes() run Virtual Machine to execute the program. Then resolve the address of the arrayList of Bytecode.

ByteCodeLoader.java: In ByteCodeLoader, a function called loadCodes is read by the BufferReader. The bytecode loader loads byte codes from the loadCodes function. Another function called tokenizer is created and initialize bytecode instance with arguments. And the program add the bytecode to the ArrayList, then clear the token list and program reads the next line. The resolveAddress function is involved for branching.

RunTimeStack.java: In RunTimeStack, the implementation of the multiple methods for the required and corresponding strategies functions such as methods of push( ), pop( ), peek( ), size( ), load( ), popFrame( ), dump( ), store( ), ArrayList( ).

Virtual Machine.java: To avoid break encapsulation, every method includes in virtual machine execute the methods which use in RunTimeStack.
For example:

```
public int peek(){
return runStack.peek();
 }
```

Program: The program file does the compairing operator and included HasMap Comparison Operator is used comparison the instance by type : (**instanceof**).

3.Testing and Execution

Testing can be executed by the user input arguments; depending on algorithm the that store in the bytecodes. In this case the argument cases are fib.x.cod and factorial.x.cod. Those files can be found in the home directory of the interpreter project.
Examples of output for both argument cases are shown below.

Case 1: Fib.x.cod (Fibonnaci)

```
run:
Enter an Integer.
3
Result is:
2
BUILD SUCCESSFUL (total time: 2 seconds)
```

Case 2: factorial.x.cod

```
run:
Enter an Integer.
4
Result is:
24
BUILD SUCCESSFUL (total time: 3 seconds)
```

```
2
WRITE
5,0[2]

RETURN
5,0,2

STORE1 k k = 2
5,2

LIT 0 x
5,2,0

LIT 7
5,2,0,7

STORE2 x x = 7
5,2,7

LIT 8
5,2,7,8

STORE2 x x = 8
5,2,8

POP1
5,2,8

POP2
5,2,8

HALT
5,2,8

BUILD SUCCESSFUL (total time: 4 seconds)
```

```
RETURN factorial<<2>>
[3,3][2,2,1]

BOP *
[3,3][2,2]

RETURN factorial<<2>>
[3,3,2]

BOP *
[3,6]

RETURN factorial<<2>>
6

ARGS1
[6]

CALLWrite
[6]

LOAD 0 dummyFormal
[6,6]

6
WRITE
[6]

RETURN
6

POP3
6

BUILD SUCCESSFUL (total time: 3 seconds)
```

Fig: 3.1 Argument Case

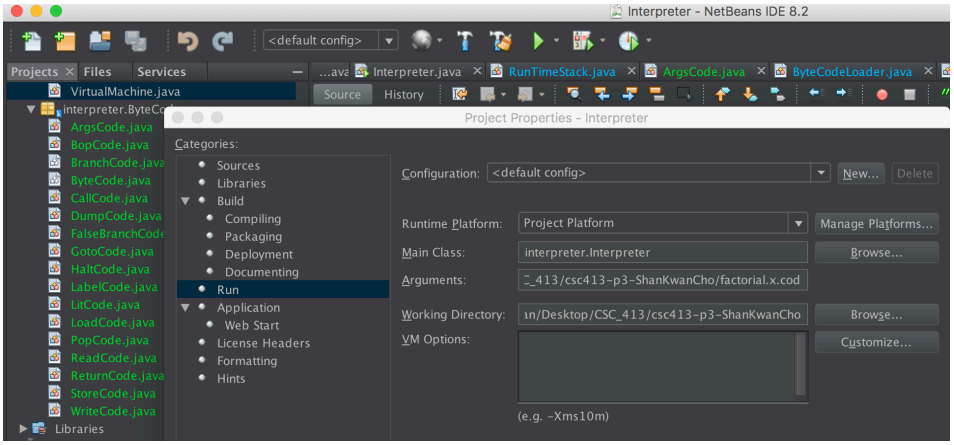

Fig: Configure the argument cases.

For errors and bugs, see section 8: Support, Contact and FAQ

4.Assumption

First of all, I have difficulties reading the instruction and took me time to understand what the interpreter program means and how to implement with other files such as bytecode, bytecodeloader, virtual machine, runtimeStack, program, codeTable and Interpreter. As well, parent-child terminology along with the bytecode structure would be challenge to be able to implement all together among those bytecodes.

5.Implementation discussion

For the implementation discussion, I had spent couple hours to understanding the concept of the Interpreter: child-parent class heritage hierarchy, how does arrayList structures between those files, especially trick concept which is avoiding break encapsulation, and the overall relationship with bytecodes, bytecodeLoaders and program call to run on virtual machine.

Slack private channel: csc413-spring2018
Collaboration Panel:

I collaborated with one of the Slack member @Thanh Le issuing with the topic of load(int offset) and store(int offset) methods.

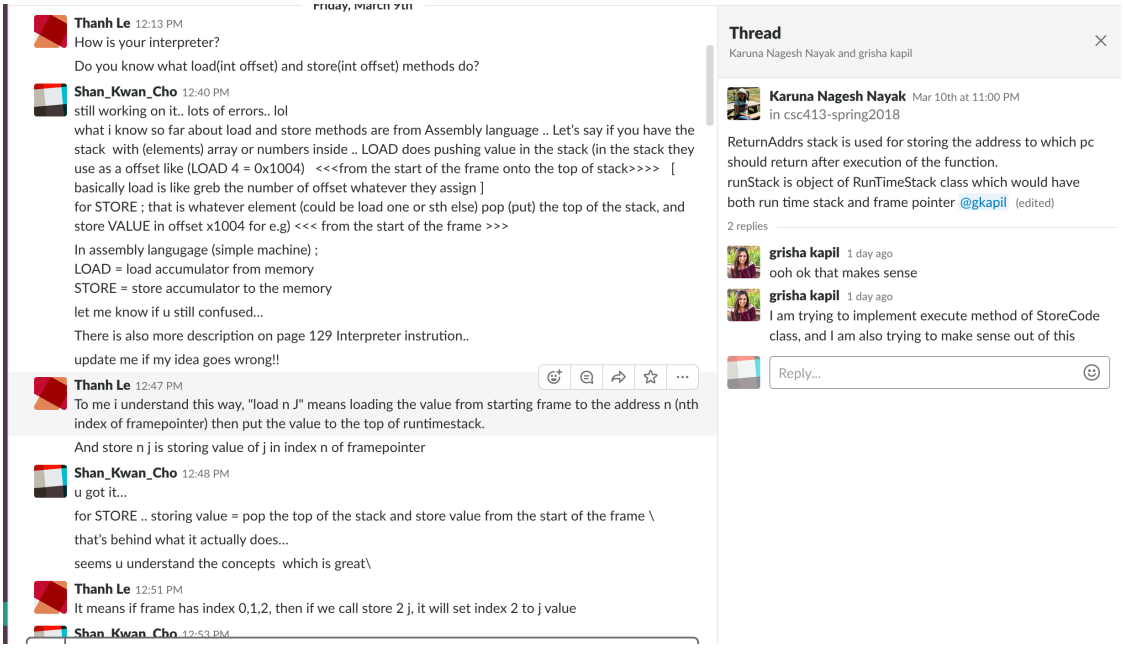

Fig 5.2 collaboration panel on Slack

6.Results and conclusions

Interpreter project has to implement with the bytecodes that need to be able to read by bytecodeLoader and execute to create the virtual machine which request to execute the program. Generally, the program interpreter from the bytecodeLoader which is embedded the bytecodes(machine language) and execute a virtual machine to run as a program. Especially, I learned the idea implementation of the interpreter; how bytecode embedded, how bytecodeLoader read those and system direct request new virtual machine system to execute the program. And I revised the usage of HashMap, stack, tokenizer, ArrayList, some override methods which needed in the project. And I am now able to deploy the pieces of code my code by reading the instructions, chopping into detail, searching google and figure it out little by little. I am actually learning that how system works behind the sense in more detail, how codes are embedded in the corresponding file and retrieve as a loader and pass those in order to execute. I do believe that testing piece by piece is safer and save time to debug the corresponding errors and prevent overloading of errors. One of the useful tips that I learned throughout this project is that collaborating with same project collaborator and contribute and collaborate, debug the issue, errors and develop the particular area together. This strategy is very useful and good routine to keep in the future in software developing environments.

7.Credit and Reference

Algorithm and Instruction: Instruction guide for the Interpreter project
Collaboration on Slack: tle25@mail.sfsu.edu
Slack: private channel: csc413-spring2018
IDE NetBeans: https://netbeans.org

8. Support, Contact and FAQ

If you have issue with the calculator project, please let me know by email: scho4@mail.sfsu.edu

9. License

This project is licensed under [Interpreter project Version 1.0]