Deepinder Bhuller

Student ID: 911770878

CSC 413.03 Spring 2019

https://github.com/csc415-03-spring2019/csc413-p2-rogue-7

Project 2:

Interpreter

Introduction

Project Overview

The basic idea of project two was to build and implement an interpreter for a mock programming language X. There are a number of parts of the Interpreter project that must be implemented to work successfully. The parts of the project are the bytecode folder, the VirtualMachine, RunTimeStack, ByteCodeLoader, Interpreter, CodeTable, and Program classes. The bytecode folder contains all the bytecodes that result from processing the language X files. When a file is passed into the Interpreter, the file is processed by the Interpreter and run in the Virtual Machine. The main requirements of this project were to **(1)** Implement all the ByteCode classes with the correct abstractions, **(2)** Complete the implementation of the following: ByteCodeLoader.java, Program.Java, RunTimeStack.java, and VirtualMachine.java, **(3)** Make sure all variables have the correct modifiers, **(4)** Make sure not to break encapsulation under any circumstances. The CodeTable and Interpreter classes have already been implemented.

Technical Overview

The Interpreter Project is an implementation of a mock programming language X. The Interpreter class is the entry point for this project and is where main() is located. When a file is passed into the interpreter, it is read line by line and bytecode instances are generated. This is accomplished through keeping track of the Frames and the Runtime Stack. Frames are the set of variables(arguments, local vars, and temp storage) for each function called during the execution of a program. A stack is the best way to implement this because the function calls and returns happen in LIFO order. For example consider the following execution:

```
Main() {
        A();
                B();
}
```

The corresponding runtime stack would would have Main() at the bottom, A() next, and B() at the top of the stack. First B() would be executed, then returned to A(), then A() is executed and returns back to main(), which then completes execution. Taking this concept, when applied to a sample Language X program, each function call represents another frame.

There are about 15 bytecodes, some examples are HALT, DUMP, READ, WRITE, ect. All of these codes can be abstracted to a general ByteCode class.