CSC 413 Project Documentation Fall 2018

Vincent Santos 917723539

Section 2

1 Introduction

1.1 Project Overview

In this project we were given the task to create an Interpreter to read low-level instructions known as bytecodes and perform their respective executions.

1.2 Technical Overview

For our project the entry point was the Interpreter file. Through here we loaded the given file via the ByteCodeLoader class and returned the program into a Program object. We then instantiate a VirtualMachine object with the program object from earlier and run the VirtualMachine method .execute(). This method is what ultimately executes each bytecode.

1.3 Summary of Work Completed

Interpreter – Class was already fully implemented for us.

ByteCodeLoader – Parsed Tokens from file and used reflection to load objects into Program object.

Program – Implemented resolvedAddress() to give proper addresses to jumpCodes.

RunTimeStack – created interface methods for Virtual Machine class to use to modify and retrieve information about the stack.

VirtualMachine – created interface methods for the bytecodes to access the RunTimeStack. Also, created getters and setters for the VM data fields for the bytecodes to access.

ByteCodes – implemented each class to they can execute the respective tasks.

2 Development Environment

Java SE 9

Intellij IDE

3 How to Build/Import your Project

I built my project in Intellij IDE using the “Build Project” option.

4 How to Run your Project

Run the main in the Interpreter class in order to access all the components of the project.

5 Assumptions Made

1. We assumed the .cod files we were given are free from errors (such as 1 bytecode every line and correctly spelled bytecodes)
2. The necessary methods to implement the VM were implied through abstraction of the rest of needs of the bytecodes.
3. We were expected to not break encapsulation and abide by the OOP principles.
4. It wasn’t mandatory but there was an assumption that there was a better hierarchy of the bytecode classes so that is easier to implement the resolveAddress() in Program.

6 Project Reflection

This project had a lot of moving parts. At least more so than what I have been accustomed to so far in the curriculum. Having classes at times at about 4 layers deep, it is real easy to break encapsulation but it was necessary to keep for good practice (and a good grade). Although it didn’t seem so at first, the dump aspect of the project (especially the formatting of the printed stack) was one of the most difficult. I believe the heftiest part of this assignment was the monstrous pdf we were given and trying to process all that information to understand what hell was going on.

7 Project Conclusion/Results

In conclusion, when given a very instruction heavy assignment, with tons of pages, it is best to just digest that information first and draw it out if needed before starting the code implementation. I used a toString() override to display all the necessary bytecodes when dumping. My results from my code ended well with my output matching what was supposed to be the output from the bytecode executions.