CSC 413 Project Documentation

Spring 2021

Kyle Harvey

915139815

CSC 413-01

https://github.com/csc413-su21/csc413-p1-k-harvey

Table of Contents

1 Introduction 3
1.1 Project Overview 3
1.2 Technical Overview 3
1.3 Summary of Work Completed 3
2 Development Environment 3
3 How to Build/Import your Project 3
4 How to Run your Project 3
5 Assumption Made 3
6 Implementation Discussion 3
6.1 Class Diagram 4
7 Project Reflection 4
8 Project Conclusion/Results 4

1 Introduction

1.1 Project Overview

For this assignment we will be implementing an interpreter for a mock language X which could be thought as a simplified version of Java. We will have the interpreter to be responsible for processing byte codes that are created from the source code files.

1.2 Technical Overview

1.3 Summary of Work Completed

So far we were provided some code with the Interpreter class along with the CodeTable class which we weren't allowed to touch. The main classes provided that we needed to code mostly where within the vitualmachine such as Program RunTimeStack VirtualMachine and also ByteCodeLoader class. The other separate java classes we needed to add as well were all the bytecode classes we needed to add into as well for our interpreter to read each line from the source code putting them into a stack. I was able to add the ArgsCode, BopCode, ByteCode, CallCode, DumpCode, FalseBranchCode, GoToCode, HaltCode, LabelCode, PopCode, ReradCode, ReturnCode, StoreCode, and WriteCode. Although there seems to be no issues visible in the codes, trying to run one of the source code shows errors within the bytecodeloader which I wasn't able to figure out, which seems to come from trying to throw exceptions.

2 Development Environment

Version of Java Used: Java 16

IDE Used: InteliJ IDEA 2021.1.2 Ultimate Edition

3 How to Build/Import your Project

4 How to Run your Project

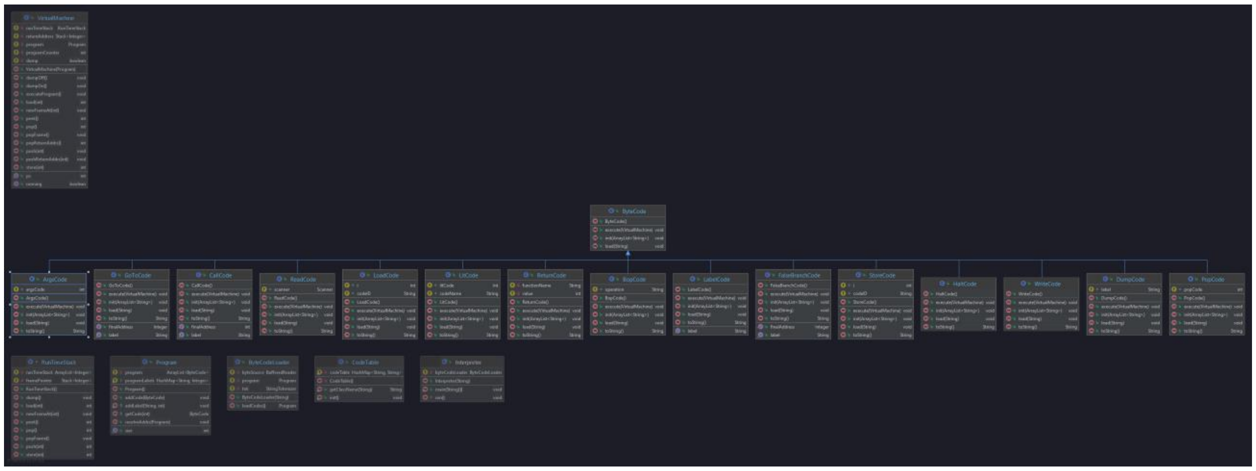
5 Assumption Made

The assumption was made that we were working with only integer type variables.

6 Implementation Discussion

For the program's main method that was being contained within the Interpreter Class, it's where the Interpreter starts. It calls the ByteCodeLoader class which is where it's supposed to load codes from the provided source files such as factorial.x.cod compiling the code, and tokenize it to make an instance of each token for their appropriate Bytecode type. From there the ByteCodeLoader will create these objects using the CodeTable class, and reflection. After all the different ByteCode objects we have made and initialized, the BytecCodeLoader will call the Program class to sort out the addresses for the different objects. Once all the ByteCode's are then passed to the VirtualMachine Class it will start to execute each bytecode. We would be utilizing polymorphism to dynamically execute those ByteCode. Those single ByteCode will be using a function in a Virtual Machine to carry out their operations through the runtime Stack. The virtual machine will basically act as a middle man to make sure encapsulation is maintained the whole time.

6.1 Class Diagram



7 Project Reflection

My thoughts on this project, although it wasn't too difficult in the sense of what we needed to do the hard part was trying to figure out where to start and go from there. Most the time consuming part came from piecing everything to together and making sure they work with each other.

8 Project Conclusion/Results

Overall for this project it was a good experience with learning more Java programming with by trying to get the Interpreter program to work. Since the idea is to work with a program that's supposed to interpreted a mock language that could be thought as a simplified version of Java, we needed to implement ways for our virtual machine to break down a source code. So we added in different bytecodes for it to read. This helps us grasp the functionality of our program and what we could have it do. Although unfortunately I wasn't able to get my program to compile properly to run any of the test in time, it was still a good experience to try and put it together. Hopefully in the future I would be able understand and get the program to work for future projects.