CSC 413 Project Documentation

Summer 2020

Javier Gonzalez

916582412

CSC413.01

https://github.com/csc413-01-SU2020/csc413-p2-gonzalezjavier.git

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 4

6 Implementation Discussion 4

6.1 Class Diagram 5

7 Project Reflection 5

8 Project Conclusion/Results 5

1 Introduction

1.1 Project Overview

This project is an interpreter that takes in compiled code from the mock language X and processes the code to be run in Java. Once the compiled codes are processed, a virtual machine runs the program. The output is the same output that was to be expected from the original mock language program.

1.2 Technical Overview

The interpreter works by taking in byte code from the from a compiled code file. Each line contains its own byte code. When the code is being read and stored in a program object, we create new instance of the specific bytecode and read the associated arguments. The bytecode types are created using a HashMap that have the code set as the key and their values being the name of the class associated to the specific concrete class. The program object that contains all the code is then sent to the Virtual Machine class to be ran.

1.3 Summary of Work Completed

The project works as expected and everything runs as if it were being run by the original X code file. Correct answers are given for the two sample files provided for testing and all the required work was completed. The Byte code classes were created and completed based on their own characteristics and requirements along with the ByteCodeLoader, Program, VirtualMachine, and RunTimeStack classes.

2 Development Environment

During the development of this project I used the IntelliJ IDEA with the version 13 of Java.

3 How to Build/Import your Project

Using the IntelliJ IDEA, to import the project you can open the IDE and click ‘Open or Import’ button. From there you can go the project folder and click on the ‘csc413-p2-gonzalezjavier’ folder which is the main repo folder. Select this folder to import. If the IDE is already open, click from the file menu and use the ‘New Project from Existing Resources’ option. From here you can import the project by clicking the same ‘csc413-p2-gonzalezjavier’ folder. It then will show the required libraries needed and include them in the project. A JDK will also need to be installed and you will be given the choice to install one if needed. At this point you should be able to complete the import of the project.

4 How to Run your Project

In order to run the interpreter, you should be in the ‘Interpreter’ class and press the ‘Run’ button. This will compile the code for the first time, and you must then click the ‘Run’ dropdown menu and click on the ‘Edit Configurations’ option. You can then select the compiled code file that ends in the ‘.x.cod’ configuration type the full name in the ‘Program arguments’ text field. Apply the changes and rerun the program.

5 Assumption Made

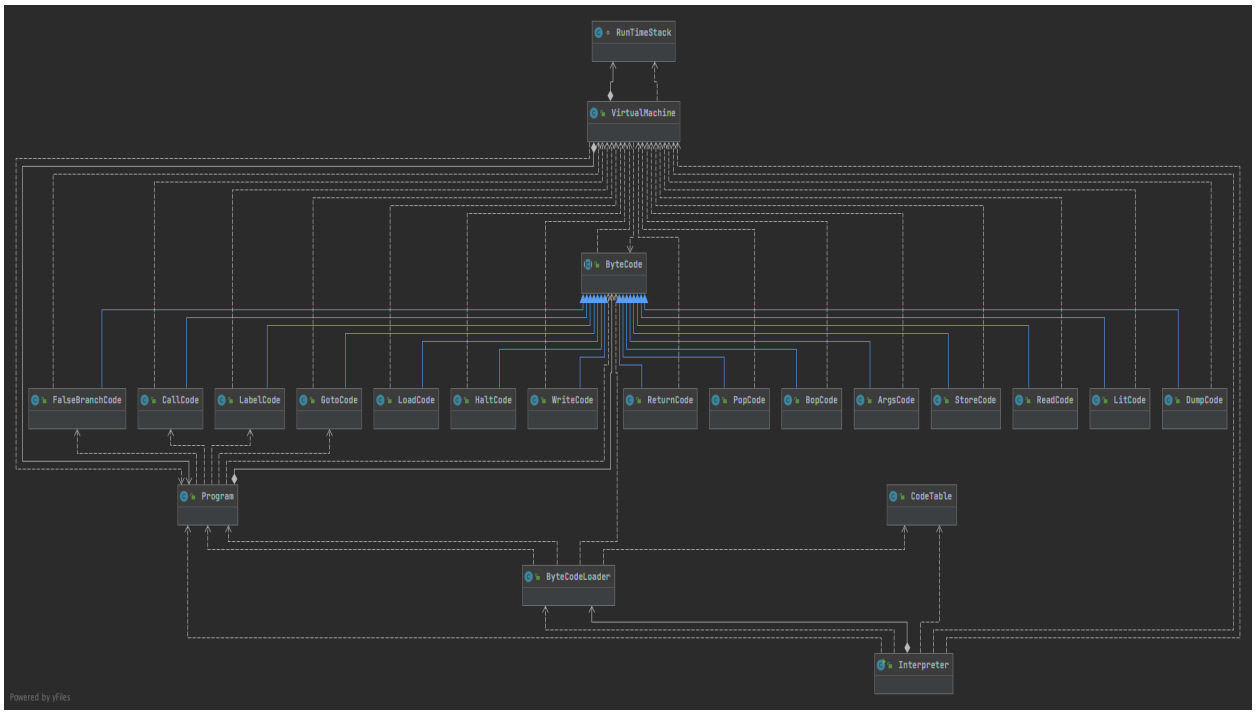
The assumptions made include that all literal values were of type int for printing purposes. Also, we assume that the compiled code being loaded in as the program are written exactly in the format of the provided test code. This means capitalizations and arguments are passed in the same order.

6 Implementation Discussion

For this project, we had to load in the test code in order for it to run as a program in the virtual machine. There was a ByteCode abstract class and from this many concrete classes were created for each type of code. These byte codes were loaded in and saved as their respective byte code object. After the byte code was loaded, they were added to an instance of the program class and stored in its array list. The array list was used since it allows us to jump to specific indexes when needed. Some resolving of addresses had to be done for the falsebranch, call, and goto byte codes within the program class before the program would be sent to the virtual machine so that it knew which address/index to jump to.

Within the virtual machine, the execution of our program was called. The VM served as the “middle person” for the RunTimeStack class and the individual byte code classes. Based on what each bytecode’s characteristics called for, the execution was done by the byte code and whenever resources or methods were needed from the RunTimeStack class, they were accessed by invoking a method in the VM class first, which then requested from the RunTimeStack class. This was done as a way of not breaking encapsulation. It may not have been required but was done in to be a way of practicing encapsulation. The VM class then carries out the program until it is finished. Dumping of the run time stack was completed along with the dumping of each individual byte code. This will only work by adding the ‘DUMP ON’ bytecode to the compiled code file. The following figure shows the UML Diagram for this project.

6.1 Class Diagram



7 Project Reflection

This project showcased the ability to think deeply about what the fundamentals of a project are and to create the methods and classes need in order for them to all work together. I was able to figure out many things about the project and how they would work by writing a lot of it out on paper and thinking about the core elements needed to complete the task.

8 Project Conclusion/Results

I was able to produce a successful project based on what was required and needed to complete the project. The interpreter successfully interprets the compiled code from the mock language X and executes it as expected.