CSC 413 Project Documentation

SUMMER 2020

Student name: Yangesh KC

Student ID 920771082

Class.Section 02

GitHub Repository Link:

https://github.com/csc413-02-SU2020/csc413-p2-yogeskc.git

Table of Contents

1 Introduction

1.1 Project Overview

The program implements an interpreter used to run a program loaded from a file containing "X" language compiled bytecodes.

1.2 Technical Overview

In this assignment, the program takes a file containing "X" language bytecodes as its argument. First, the ByteCodeLoader class must parse files from the corresponding bytecodes and store them in the Array List program. Secondly, the bytecode loader calls the program's resolveAddress() function to inspect each bytecode and resolve the address to which the byte code is referring if any exists. Then, the Interpreter creates a Virtual Machine to move through the program and execute each bytecode in order. The Virtual Machine is responsible to manage the entire data structures including an object of the runTimeStack class in order to maintain the state of the program and to avoid any collision or errors. Each bytecode requests the Virtual Machine to make changes to its structures and change the state of the program

1.3 Summary of Work Completed

For the following assignment I have implemented the ByteCodeLoader, Program, RunTimeStack, and VirtualMachine classes in the interpreter package to work alongside the provided Interpreter and CodeTable classes to run a program accordingly. Also, I implemented each ByteCode class to carry out the necessary operations on the program as instructed in the pdf. Additionally I added abstract classes ie AddressCode and IntArgumentCode to improve the bytecode inheritance hierarchy after numerous attempts to solve the issue and few references from online sources.

2 Development Environment

I completed the assignment with Java Eclipse IDE for Java Developers with Build Version in Windows 10. However, it should have no issue running in any java supported IDE ie visual studio, intellij etc.

3 How to Build/Import your Project

I imported my project via github into Eclipse from the file system by importing the master folder titled "csc413-p2-yogeskc". The initial setup was done automatically by the IDE however we need to pass the ".x.cod" file to compile the program and successfully run it. As demonstrated in the class, we need to edit program configuration to pass the .x.cod file to the project.

4 How to Run your Project

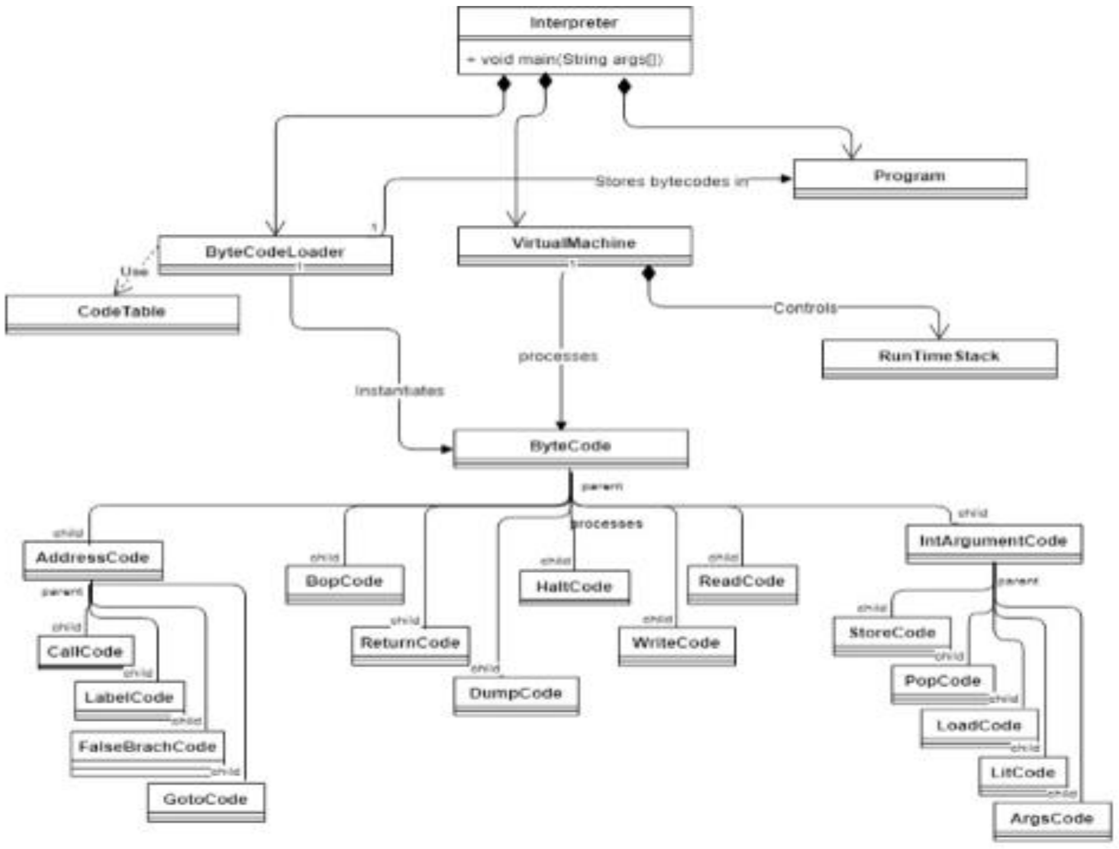
Project was run from within the Eclipse IDE by pressing Ctrl+F11 or from the "Run" dropdown menu. Program requires a ".X.cod" file as an argument, this was set in the Run Configurations in the IDE. Sample ".X.cod" files are located in the main folder. In intellij you can build the project first, then edit configuration, and pass the file via program argument in the pop up tab.

5 Assumption Made

The program assumes that the bytecodes represent the correct compilation of “X” language source code. However, the RunTimeStack class protects the program state from operations which would cause Stack related exceptions. In other words, the program is designed not to crash even if the source code contains errors. The program must be able to process out language X as given Fib.x.cod or Factorial.x.cod in java and giving the correct output.

6 Implementation Discussion

6.1 Class Diagram



7 Project Reflection

This assignment was implemented with the expected results. I encounter numerous failures along the way to finish this assingmnet. One of the most challenging and complex tasks was to implement the dump() function within the RunTimeStack. This function is meant to simply print the contents of the stack, but it requires altering several of the Run Time Stack data structures, and then putting them back together to print the program state from left to right, between every single execution of a bytecode. Some consideration was made to the best implementation for the bytecode inheritance hierarchy. There may be a better and cleaner way for each bytecode to inherit from ByteCode to inherit the methods and variables they each need, but after lot of research and stackoverflow discussion along with the slack conversations this implementation worked well. Separating the codes which would need an address resolved and having them inherit that variable from the abstract class AddressCode. The

IntArgumentCode allowed the separation of bytecodes which did not need to store and int argument, so that they did not have to wastefully inherit that variable.

8 Project Conclusion/Results

The program successfully implemented an “X” code interpreter. The included test files “factorial.x.cod” and “fib.x.cod” can be run to obtain expected results. The program is stable for very large inputs such as 14 factorial.