CSC 413 Project Documentation

Fall 2019

Sawara Bhattarai

918575273

Class. Section: 413.3

GitHub Repository Link:

https://github.com/csc413-03-fall2019/csc413-p2-sawarabhattarai.git

Table of Contents

1 Introduction

1.1 Project Overview

This project is about writing code for interpreter. Interpreter takes and reads input as a source program. It also takes data from the code and changes the instruction of source program into machine language. The source code is written in bytecodes and are executed by virtual machine.

1.2 Technical Overview

The computer program that is executes instructions that are written in script or program language without previously being compiled in the machine language program is an interpreter. In interpreter class there is virtual machine which doesn't have registers to sort arbitrary value that's why everything is pushed in the stack before the calculation is done. I got background knowledge about interpreted language and how it works.

1.3 Summary of Work Completed

I added classes to the packages to run the project. In the first package, interpreter there are six classes, Interpreter.java, ByteCodeLoader.java, CodeTable.java, Program.java, Runtimestack.java and Virtualmachine.java. I started working from creating byte codes inside the package interpreter.bytecode. In the package, I created seventeen class in which fifteen there given in the Code Table class and I also added two other classes named branch code and bytecode. Interpreter.java class contains main, and function of the class is to initialize, load bytecode from the file and make it run in the virtual machine by sending it. I did not make any change to this class as said in the instruction. In the other class ByteCodeLoader.java I added the code which makes it loads the bytecode to the program class. It also builds an instance class that corresponds to bytecodes. It reads a bytecode and then builds a new instance for the code and then adds it to the "Program" object. Program is one of the other class in the package. In this class after all the bytecodes I created gets loaded, it makes all the symbolic address gets resolved. In the class Codetable.java I implemented the code and it is used by the ByteCodeLOader.java class. This class contains a HashMap that has the bytecode strings like Read, Dump, Call and the name of the corresponding class like Read Code, Dump Code, Call Code.

Next class, I implemented Program.java class, it has methods to add bytecodes to the program class. It is also used to resolve address of all bytecode instance of the program class. Then, I implemented RunTimeStack.java class which records and process the stack for active frames. It uses stack frame Pointer that helps maintain the stack's that are in active frames. It also uses Array list to hold the actual data. So, I added the code for frame Pointer list but when I run the program frame doesn't show. I tried by debugging but couldn't figure out the problem. The last class in this package is Virtiualmachine.java and this class helps to execute the Program.

2 Development Environment

Version of Java Used: Java 11

IDE used: InteliJ IDEA

3 How to Build/Import your Project

The first step I did was open the GitHub and then I copied the link to clone the project. After that I created the folder name git in my desktop and then used the terminal to get the code. In terminal, first changed directory to desktop then again changed it to git the I wrote clone and copy pasted the link that I got from GitHub. After that, I went to my IntelliJ and did open project and open the downloaded file and went through each step.

4 How to Run your Project

Open the main of the program which is Interpreter class for this project. Then, the first way is to locate and click the small green button at the top of the window. Another way to run your program is to click on the "Run" tab at the top of the window and then select run configurations. But in this program, we need to add a file name factorial.x.cod. So, to add it we build the main program run it then go to edit configuration where we can file a line that says Program arguments then, in the box we have to write factorial.x.cod and press enter and run the code again using the green button.

5 Assumption Made

Project skeleton was given already and in the pdf of requirement it said what we were supposed to do but it was very confusing and had to read it multiple time. After reading it again and again I thought we need to create all the bytecodes inside the bytecode class and bytecode is needed to execute the code. After reading the pdf once, I also assumed that I need to run virtual machine which was not correct which I figured out after reading the pdf couple of times.

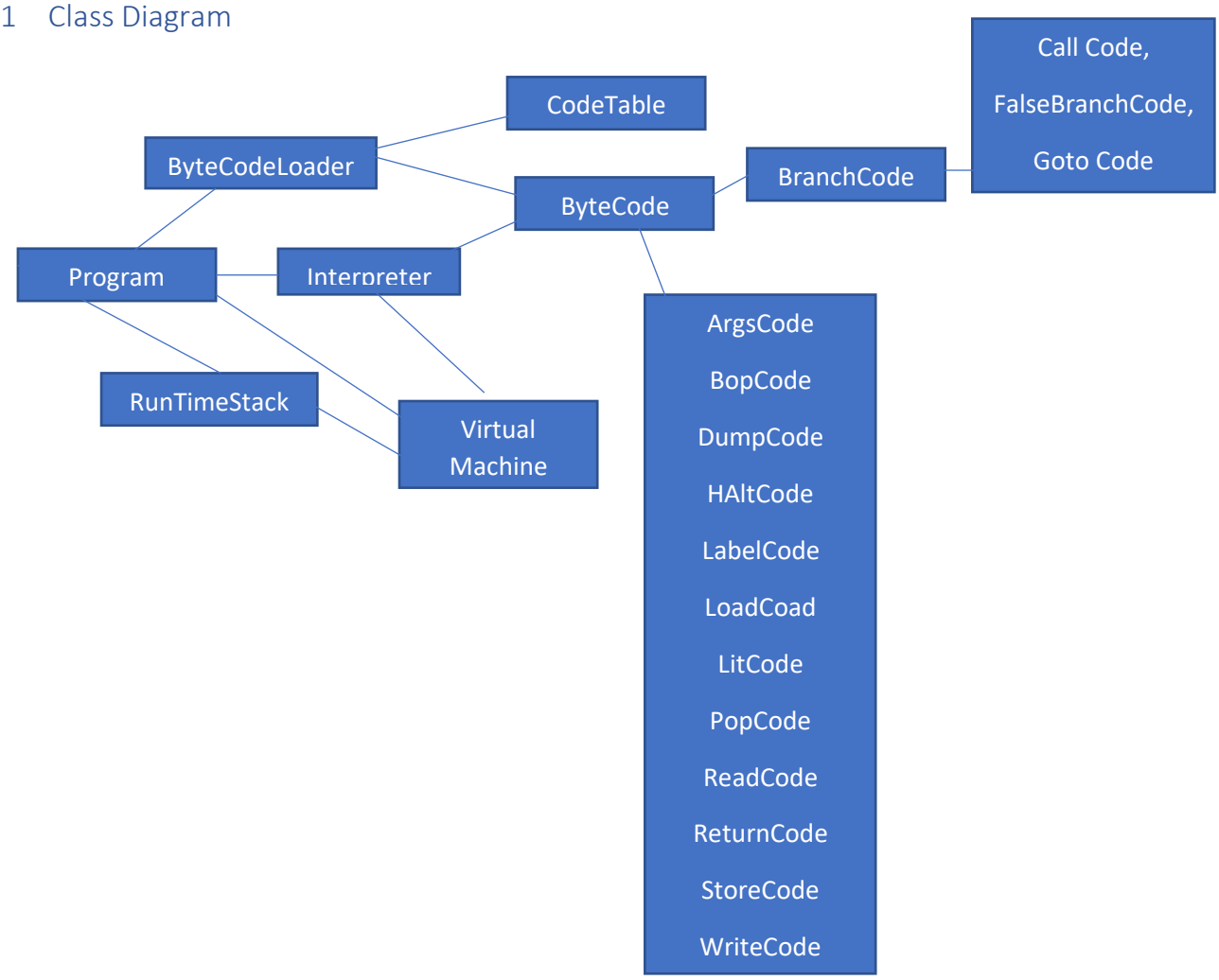
6 Implementation Discussion

In this project, six classes were given inside the package interpreter and which had its own functions. The classes are Interpreter.java, ByteCodeLoader.java, CodeTable.java, Program.java, RuntimeStack.java and VirtualMAchine.java. Interpreter is the main of the program. Bytecode loads the bytecodes to the program class. CodeTable contains the HashMap and it is used by BYteCodeLoader. Program class has method to add bytecodes. Runtime Stack records and processes the stack and virtual machine executes the program. I created other sixteen class inside the package interpreter. bytecode.

ArgsCode.java is used to notify the interpreter to set up a new frame which has a number given from arguments. Bop.java helps leveling the stack and performing the given operation. BranchCode.java is an abstract class to branch the bytecode. ByteCode.java is an abstract class for the bytecodes. CallCode.java helps transfer the control of the given function. DumpCode.java helps determining that the instructions given are dumped or not. FalseBranchCode.java makes the top of the stack to pop and if the stack top pop is false it makes corresponding label to branch else it just executes the given next bytecode.GotoCode.java helps setting the code that makes the given label jump. HaltCode.java make the program to terminate.LabelCode.java helps setting the targets for the branches. LitCode.java used with one argument, such as, LIT n it loads the literal value n. If it has two arguments, such as, LIT 0 I, it loads 0 on the stack in order to initialize the variable i to 0 and reserve space on runtime stack for i. LoadCode.java pushes the value in the slot which is offset by a

given number from the start of the frame onto the top of the stack. PopCode.java pops a given number of levels of the top of runtime stack. ReadCode.java prompts the user for an integer value and then puts the value just read on top of the stack.
ReturnCode.java sets the code to return from the current function. StoreCode.java pops the top of the stack and stores the value offset a given number from the start of the frame. WriteCode.java writes the value on top of the stack to output and leaves the value on top of the stack.

6.1 Class Diagram



7 Project Reflection

I was struggling and couldn't figure out what I needed to do while starting the project. I read it many times and discussed with my friends to figure out how we need to do. I started my project by making all the bytecode classes. I finished implementing the code but after completing everything when I ran the code it did not run and gave an error. The code was pointing to null. I started debugging the whole code, but I couldn't figure out the problem. Then I started reading the pdf again and I figured out that I need to edit configuration and add a file to the project. Finally, the code ran and gave the output.

8 Project Conclusion/Results

I was able to interpret all the bytecode classes and file factorial.x.cod. I was having problem to figure out the model of instruction to structure in the code and when to approach the project. I was also having trouble compiling the interpreter. Finally, I figured out the problem and ran the code.