CSC 413 Project Documentation

Spring 2019

Student name: Sunny Srijan

Student ID: 917105649

Class.Section: CSC413.03

GitHub Repository Link:

https://github.com/csc415-03-spring2019/csc413-p2-sunnysrijan

Table of Contents

1 Introduction

This project is called Interpreter. The purpose of this project is to implement an interpreter for the mock language X. The interpreter file in the project is processing the bytecodes that are created from the source file with extension X. The interpreter and Virtual Machine file work together to execute a program for Language X.

1.1 Project Overview

All the codes we type in are high level language such as C, C++, and Java. But computer can only understand the binary form 0 and 1. But for programmers it's not possible to understand the language of 0 and 1, so we use Higher level language to make it more efficient. As the computer cannot understand the source code for higher level language, a source program is translated into machine codes of 0 and 1. The translation are done by using programming tool called interpreter.

1.2 Technical Overview

We are given with 7 files for this project to work:

1. **ByteCode**
2. **ByteCodeLoader.java**
3. **CodeTable.java**
4. **Interpreter.java**
5. **RuntimeStack.java**
6. **Program.java**
7. **VirtualMachine.java**

The interpreter file contains the main function and we were not supposed to change anything in it. These functions create a new interpreter object. These new objects initialize HashMap tables containing the ByteCodes. Then it creates ByteCodeLoader object which takes files to translate. The CodeTable file given is already implemented for us. This file contains the Hashmap that holds out ByteCodes and methods to obtain the keys. The input files are read by BufferReader objects. It also contains method that goes through this file for any chance of new instances for every instruction given. We use reflection in ByteCode to avoid bunch of if-else and switch statements. The Program file is used to store the ByteCode in a ArrayList that are read from this file. The method used in this is used to resolve the address of ByteCode files. Then RuntimeStack file works with the stack that contains the values from the instructions. It contains multiple function which help us to edit the stacks and then the Dump function prints the stacks out to the system. The VirtualMachine file contains the execute program() method which will go through every ByteCode to execute its codes. In the end, 17 ByteCode files which I created including two abstract classes take different number of parameters and perform different functions by requesting VirtualMachine to execute the instructions.

1.3 Summary of Work Completed

I started this project by implementing two abstract classes ByteCode and BranchCode and its subclasses. I added 15 ByteCode classes. After this I implemented ByteCodeLoader class. This class loads bytes from the source code file into data structure that stores in the program. Program object contains the ArrrayList which stores the ByteCode.

After this I implemented Program.java. This class deals deals with the string of all ByteCode that I created. ArrayList is responsible for all adding of ByteCode and its subclasses. Next I implemented RuntimeStack.java file. This class is responsible for processing the stack of active frames. This class contain two data structures which helps VirtualMachine to execute the program. Finally, I implemented a VirtualMachine.java. This class is responsible for executing the given program. VirtualMachine controls all the program and everything goes through this class. The interpreter class and CodeTable was already implemented for us so I didn't change anything in it. The interpreter.java contains the main function and we not supposed to change anything in it. These functions create a new interpreter object. These new objects initialize HashMap tables containing the ByteCodes. Then it creates ByteCodeLoader object which takes files to translate. The CodeTable file given is already implemented for us. This file contains the Hashmap that holds out ByteCodes and methods to obtain the keys. The input files are read by BufferReader objects. It also contains method that goes through this file for any chance of new instances for every instruction given

2 Development Environment

1. Java Version: 11
2. IDE: Intellij IDEA 2018.3.4 (Ultimate Version)

3 How to Build/Import your Project

Steps for Import

1. Open git bash clone the repository from GitHub using the link:

https://github.com/csc415-03-spring2019/csc413-p1-sunnysrijan

2. Open Intellij and click on **Import Project**
3. Go to the folder **csc415-03-spring2019/csc413-p2-sunnysrijan/interpreter**
4. Now import the project from **external model**. Then in the drop down select **Gradle** and click **Next.**
5. After this select: **Use gradle 'wrapper' task configuration**. And then select **Finish** to complete import process.

Steps to Build

1. First make sure the environment is clean by clicking
2. Then click **build project** under **build dropdown**

4 How to Run your Project

To Run this Project, go to Program.java and then click on Run dropdown and then click green Run button.

5 Assumption Made

This project in the be gaining was very confusing to me. Nothing made any sense. I read the PDF like 5 times but still didn't has clue where to start, but going to computer lab and talking to my peers helped me keep moving and ultimately, I finished it. My assumption is that every code needed this project to execute starts with ByteCode.

6 Implementation Discussion

Interpreter project purpose is to implement the bytecodes that are requires to be read by the ByteCodeLoader and execute to create the VirtualMachine which has ability to request to execute the program. ByteCode which is in machine language are embedded by the ByteCodeLoader and execute a virtual machine to run the program.

For this Project, we had to work on given 7 files in order to execute the program:

Interpreter.java: this is the file with the main method which executes the program

ByteCodeLoader.java: this file loads ByteCodes from the source file into data structure which stores the whole program.

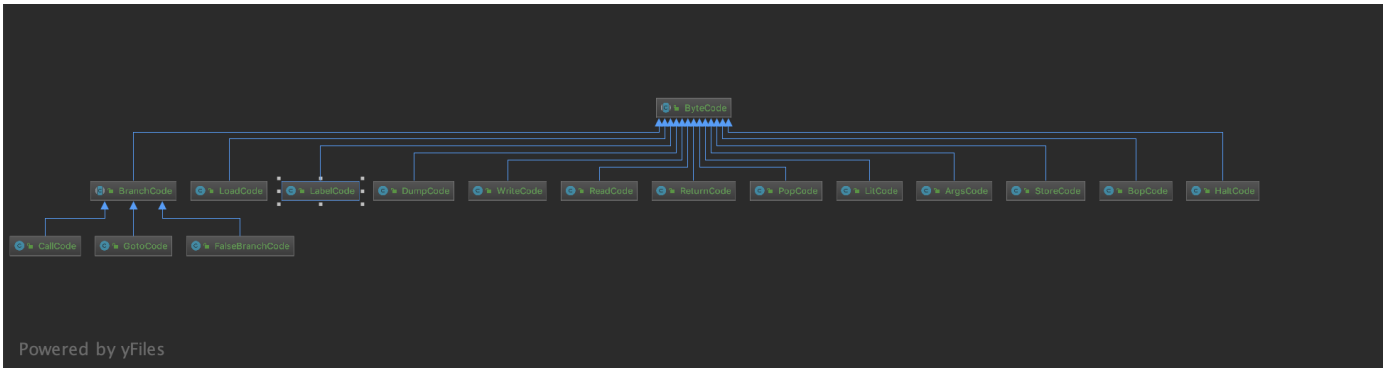
ByteCodeLoader.java: The CodeTable file given is already implemented for us. This file contains the Hashmap that holds out ByteCodes and methods to obtain the keys. ByteCodeLoader help hashmaping with bytecode.

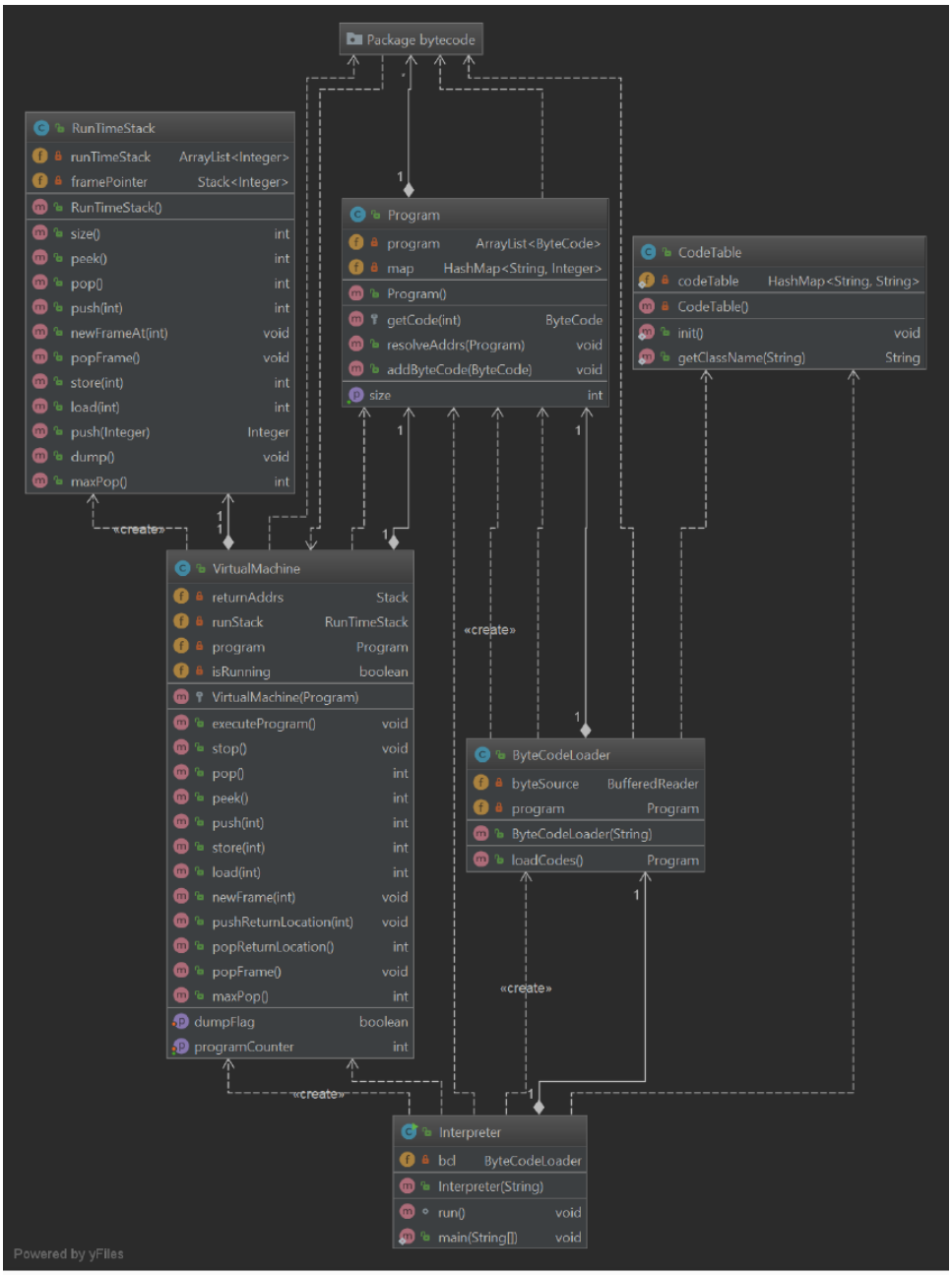
Program.java: The Program file is used to store the ByteCode in a ArrayList that are read from this file. The method used in this is used to resolve the address of ByteCode files.

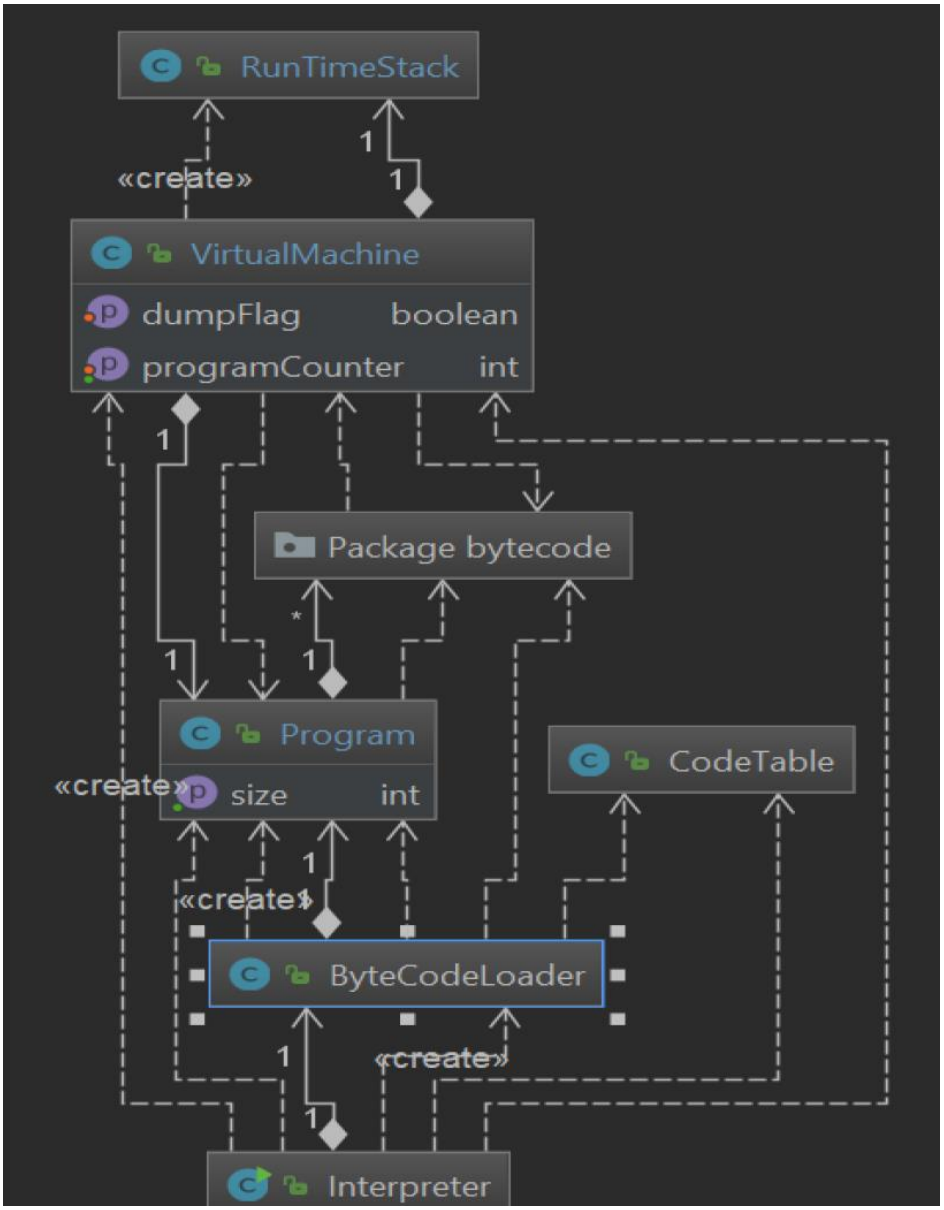
RunTimeStack.java: Then RuntimeStack file works with the stack that contains the values from the instructions. It contains multiple function which help us to edit the stacks and then the Dump function prints the stacks out to the system.

VirtualMachine.java: The VirtualMachine file contains the execute program() method which will go through every ByteCode to execute its codes. In the end their 17 ByteCode files which I created including two abstract classes. They take different number of parameters and perform different functions by requesting VirtualMachine to execute the instructions.

6.1 Class Diagram







7 Project Reflection

This project was way harder for me than the first project. I had to deal with lot of OOP structures which was kind of hard for me. I had to go through the topics like usage of Hashmap, stack, tokenizer, ArrayList, and override methods which were the key elements to execute the program.

8 Project Conclusion/Results

As I mentioned before this project for beginners is a challenge. There are lots of files in this project so in the beginning I had no clue where to start. Talking to my peers and computer lab sessions helped to understand the pdf which is the guide for us. Everything mentioned in PDF starts making sense ones you get the knowledge where to start things. All this classes and files has a single

responsibility of executing the codes properly in OOP design pattern. The out for my project is working fine as our professor gave us sample test for factorial 7 on Slack. My code give same out as required.