CSC 413 Project Documentation

Summer 2020

Naweeda

920462358

CSC0413-01

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

Table of Contents

1 Introduction 3

1.1 Project Overview 3

1.2 Technical Overview 3

1.3 Summary of Work Completed 4

2 Development Environment 4

3 How to Build/Import your Project 5

4 How to Run your Project 5

5 Assumption Made 5

6 Implementation Discussion 5

6.1 Class Diagram 6

7 Project Reflection 8

8 Project Conclusion/Results 8

1 Introduction

This goal of this is project is to design an interpreter for a simulate language which is called X. The mock language X is a simplified version of java that has useful tool for testing, and it gives you the ability to test what you write without having reliance issues.

1.1 Project Overview

This project consists of an interpreter that simply consists of two input codes, fib.x.cod which computes a value at a specific index in a Fibonacci sequence and factorial.x.cod computes the factorial of a number. Moreover, the interpreter has six designed classes, two of them are already implemented for us, and the other four has methods that we need to implement. And the interpreter has a package which is called bytecode, and we suppose to extend it with a bytecode subclass and design with different types of supported bytecodes. The classes that we suppose implement are: Bytecode loader which is accountable for loading bytecodes from the source code file into a data-structure. The Program class is responsible for storing all the bytecode read from the source file. Run Stack Time class is responsible for the processing the stack of active frames. And the last class is Virtual Machine which is responsible for executing every bytecode as a controller.

1.2 Technical Overview

For technical overview, I want to mention that this project was super challenging for me. Since we had three weeks for this project, therefore the first step for designing this project was reading the PDF for five or six times. For me structuring this project seemed a little tedious. After reading the PDF carefully, I went over the implemented classes which were interpreter and code table, then I decided to create all the supported byte codes, and after analyzing the code table I assumed that there need to implement an abstract subclass with three methods for every bytecode, which are initializing method, an execute method, and toString method. Next, I set up the initializing method for every bytecode. After, I start working on Bytecode Loader class, that consists of two methods. The first one is responsible for opening the file and puts in a scanner to get ready for reading in the program, and if the file is not found it inserts error. The second method is responsible for loading the bytecode into program object and resolving all

the symbolic addresses. The program class which is linked to three main byte codes for resolving symbolic addresses, and those classes are: goto, call, and falsebranch. These three-byte codes are responsible for setting the address and getting code table. It is designed with a hasp map and I created couple methods for getting the bytecode and its size. The hasp Map maps each name associated with a label bytecode to the address of that label byte code. And I designed a loop that run through the bytecodes to resolve the string target of “GOTO”, “CALL”, and “FALSEBRANCH” bytecodes to the integer address of the corresponding label bytecode. The next

class, which I spent the most time on it and edited it a lot of times, was RunStackTime. I wanted to make sure to avoid encapsulation issues, and all the stack methods are implemented logically. This class consists of ten methods which I do not want to name all of them. But for dumping method, I had different ideas of how to design the framework to go through each stack without popping the value and returning it properly. So, I ended up using an array list for cloning the runtimestack and frame pointer. The last class is virtual machine, which is the most important. It required the set up the dumping, it would only dump if the current code is not dump and dumping is set. In this class I created some functions to abstract the operations needed by the bytecodes. Each time the program returns from the functions, the address is pushed onto the return Address stack. Another important element was that every bytecode object should have an execute method. I tried my best to create a clean and brief functions. Also, not to create duplicate code.

1.3 Summary of Work Completed

As I mentioned most of my work at the technical overview section. I want to mention briefly, that I basically followed the coding hints from PDF, and I think those hints are super helpful. At first, I was very confused, but after reading the PDF couple time I tried to make common sense of the project.

2 Development Environment

Version of Java Used: Java 13

IDE Used: IntelliJ IDE

3 How to Build/Import your Project

- Go to my repository link.
- Then click the green button which is called “Clone or Download”.
- After, click Download Zip.
- Then, extract the zipped folder.
- After loading the Ide, click import project.
- Then select “Create project from existing sources” and enter next.
- Then keep entering next until you get to the end, and press finish.

4 How to Run your Project

The interpreter class is the entry point for the project, right click on it then run interpreter.main

Then expand the interpreter at the upper right corner of the window to create the run configurations.

After, set the command line arguments. Which is going to be “factorial.x.cod” and click on apply/ok.

The final step is, click on run interpreter. Then you will be able to see the program output.

5 Assumption Made

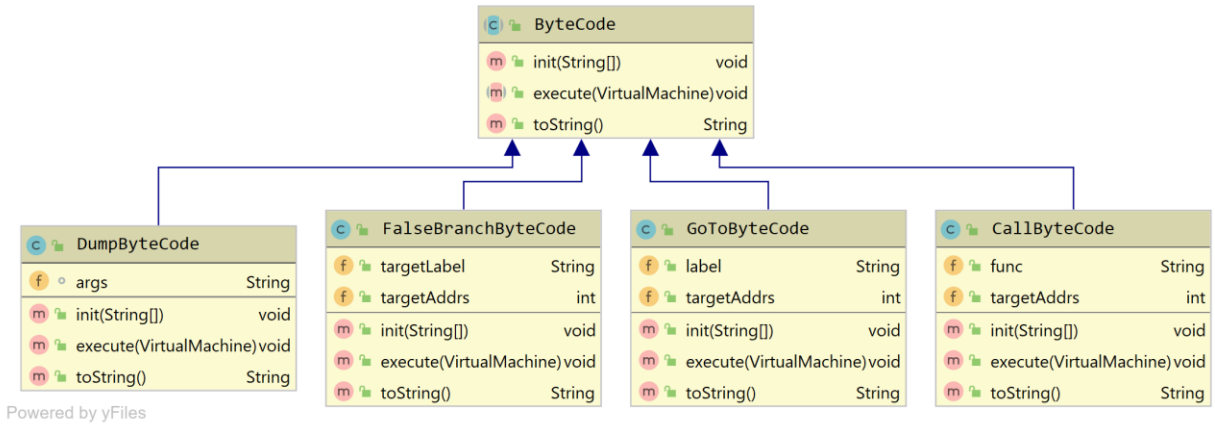
Assumptions I made for implementing this project, was structuring the program. Overall, I assumed it is preforming an activity or execution. Some take arguments or some arguments will need to be modified. And with some bytecodes I suppose to get arguments. While expanding my thoughts, I realized there exist lot of logic and connectivity in the whole program.

6 Implementation Discussion

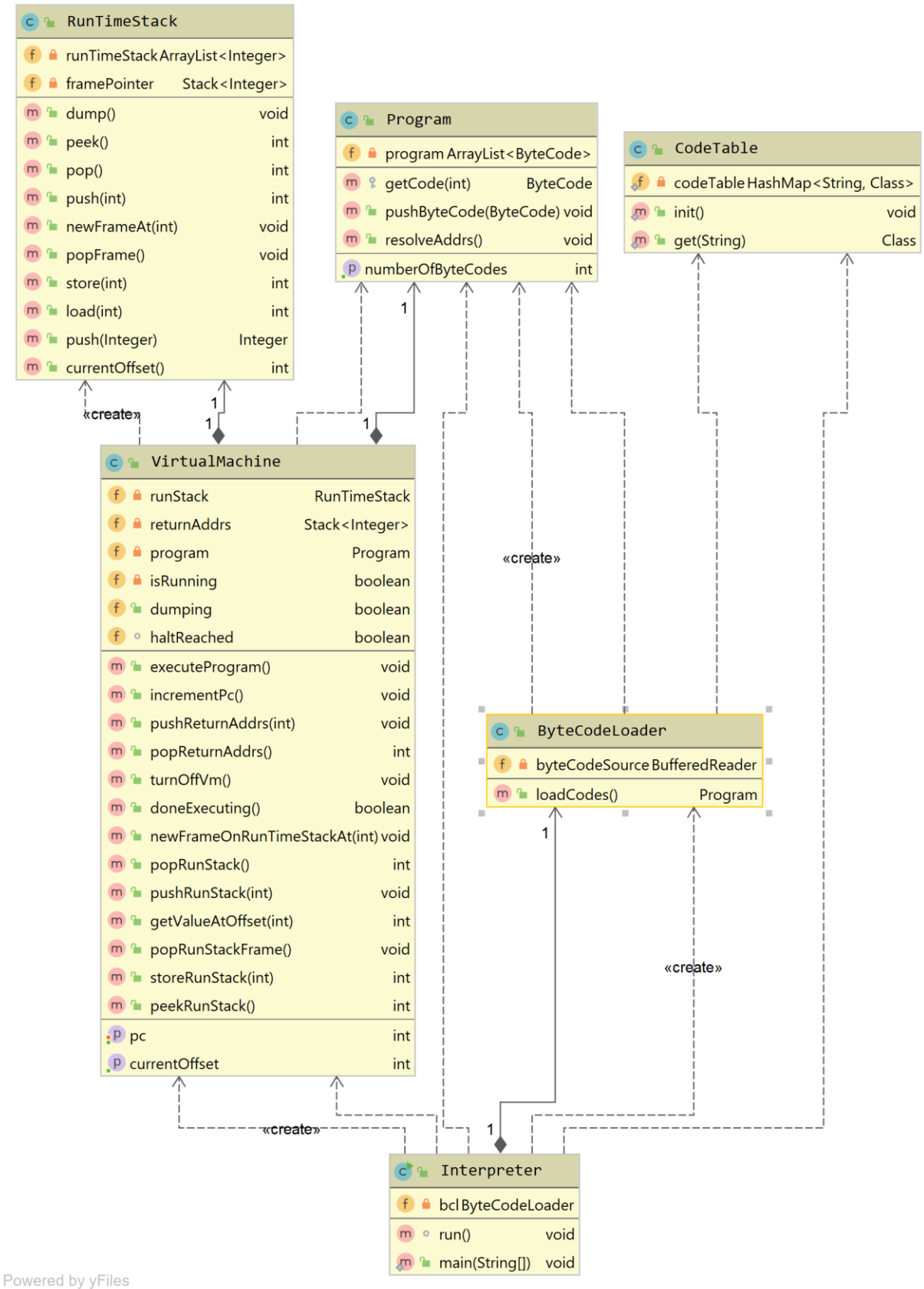
As for implementation we are required to design a UML diagram. But for most of my projects, first I am trying to draw everything and connect the dots. Since my learning method is based performing visually, by drawing I will have idea of how to start the project.

6.1 Class Diagram

The Diagram of ByteCode sub class:



The Diagram of Interpreter class:



7 Project Reflection

This project was good practice of object orientation program and not breaking the encapsulation. The first time I read PDF, I was very confused. But after reading couple time, it gave me the ability to understand the structure of the program. And made me to learn complexity. Honestly, the coding hints in the PDF, are helpful. I was super challenged doing this project and I struggled with most part of the project, but it improved my programming skills.

8 Project Conclusion/Results

The amount of time given for this project was fair enough, since there was a lot of logic behind implementing this project, but there was enough time to work on it. The issue, I was struggling was interpreting the bytecode file “fib.x.cod”, and “factorial.x.cod”. when I tried to leave the dump on inside the cod files the bytecodes are not printing. After debug, I figured the issue is with the dump method in the Runtimestack class. I tried different ways like creating an iterator to print the bytecodes but did not work. There might a better solution, but I was running out of time. Overall, I enjoyed doing this project, but at the same time it requires a lot of time and critical thinking.