CSC 413 Project Documentation

(The Interpreter)

Fall 2019

Angelo Reyes

917566993

CSC413.02

Repo Link: ***https://github.com/csc413-02-fall2019/csc413-p2-angiereyes99***

Table of Contents

1 Introduction

1.1 Project Overview

The program in the repository is an interpreter for the simplified java mock language “X” that processes bytecodes.

1.2 Technical Overview

Using subclasses that acts as commands for the ByteCode, the main Interpreter class runs a mock Language “X” to compute two sample programs, that have extensions “x.cod”, that represent the Fibonacci Sequence and factorial of a number. The interpreter is a assembly-like interpreter that compiles and processes bytecodes as it runs.

1.3 Summary of Work Completed

The key highlights of work done in the program was creating all 16 ByteCode subclasses inside the bytecode folder. I ended up adding a “ConnectCode” class to connect the FalseBranchCode, GoTo code, and CallCode so that in the returnAddrs() function, I can just use an if statement to check the instance of Connect Code. I was able to complete most, if not all, the Program, ByteCodeLoader, and VirtualMachine classes. I was able to correctly implement most of the “init” and “execute” methods for each of the bytecode subclasses. I unfortunately wasn’t able to completely finish the dump method. I was able to correctly implement getters and setters for the run time stacks and return addresses. I was also able to compile the code but not fully as it stops after the Read Code.

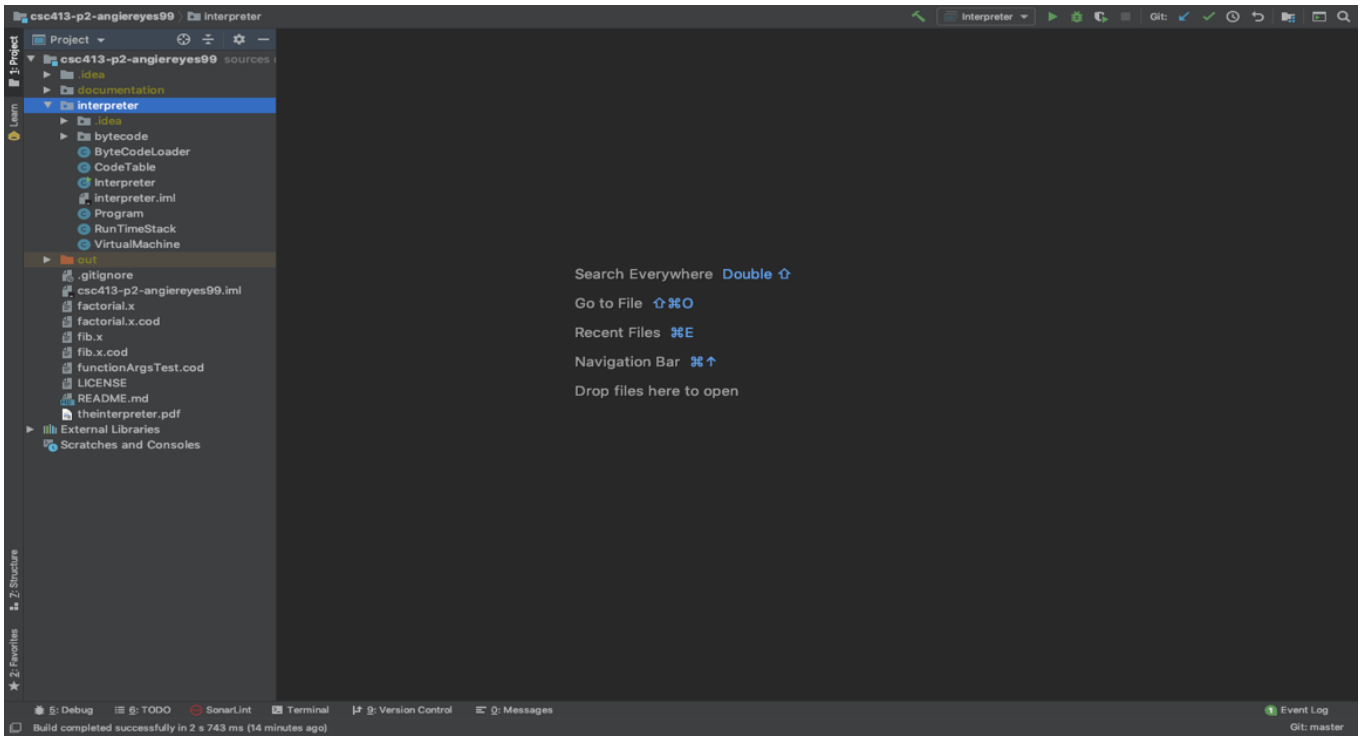
2 Development Environment

The IDE I preferred to use in this project was IntelliJ with a Java version of 12.0.2.

3 How to Build/Import your Project

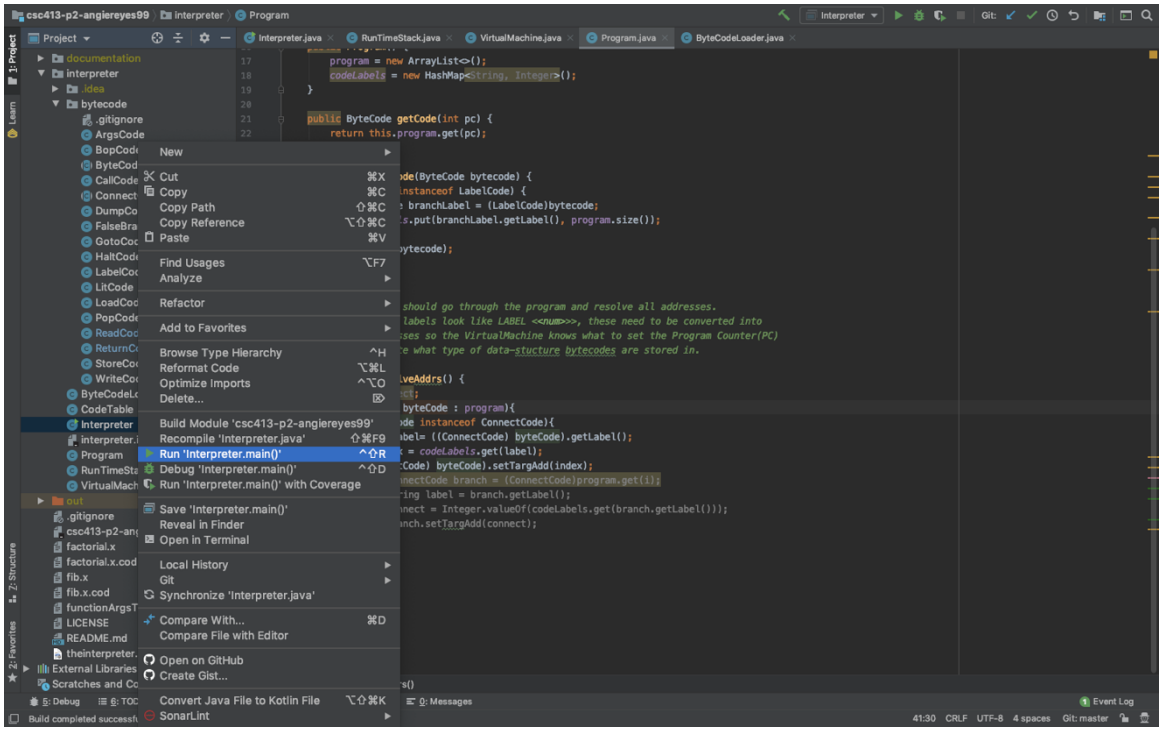
To import the project, you will need to:

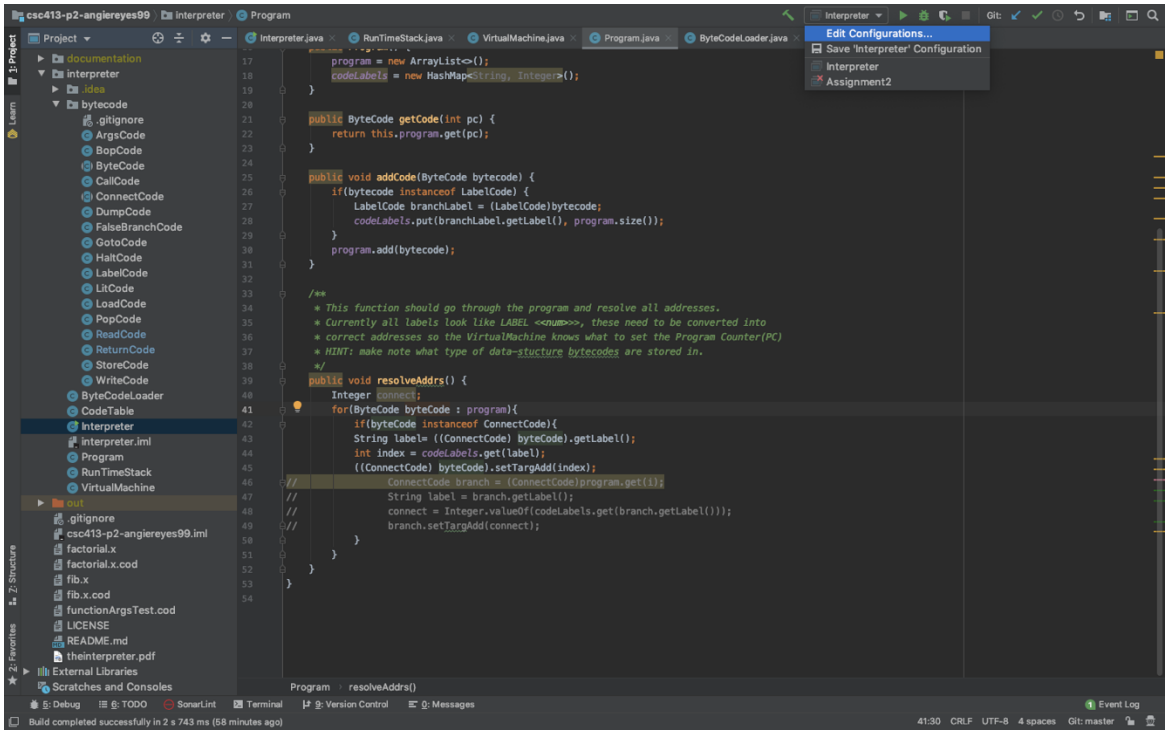
1) Go on the repo and clone the repo link.
2) Go on terminal and run: git clone “repo link you”
3) A folder should appear in your desktop; open up the IDE of your choice.
4) Choose “import project” (this is an option on IntelliJ)
5) Make sure to select the whole folder to import and leave the default settings.
6) Click finish and your IDE should look like:
7) Click finish and your IDE should look like this.

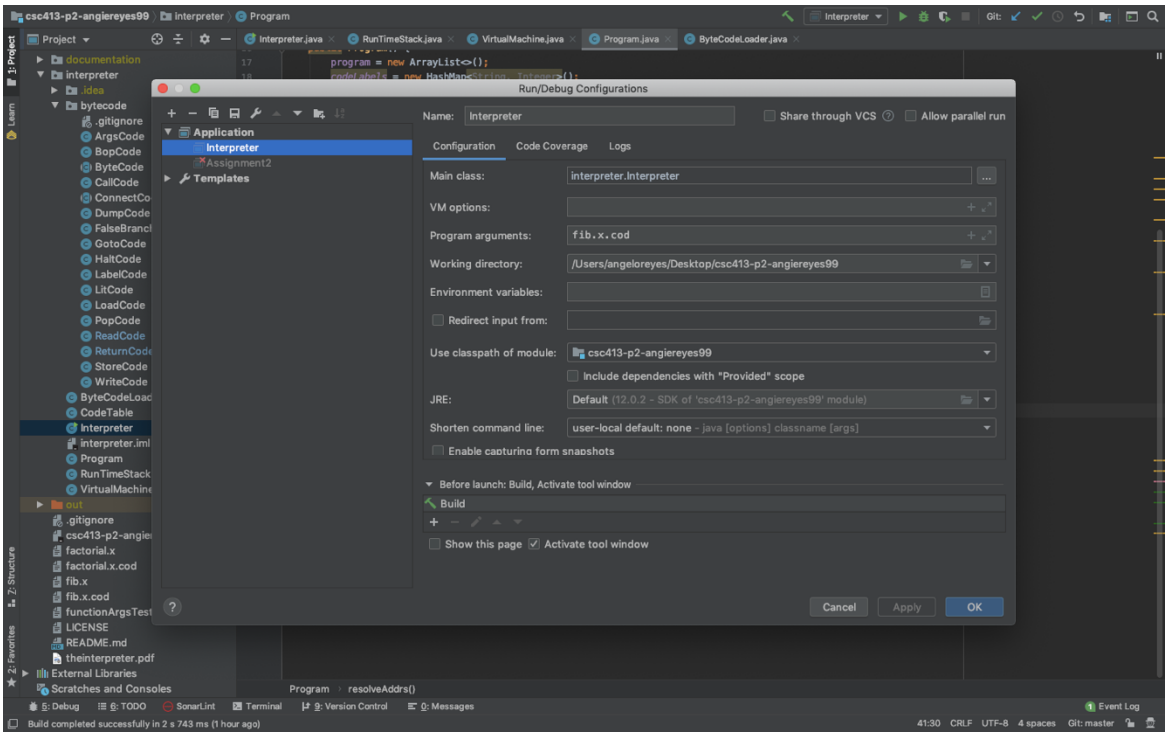


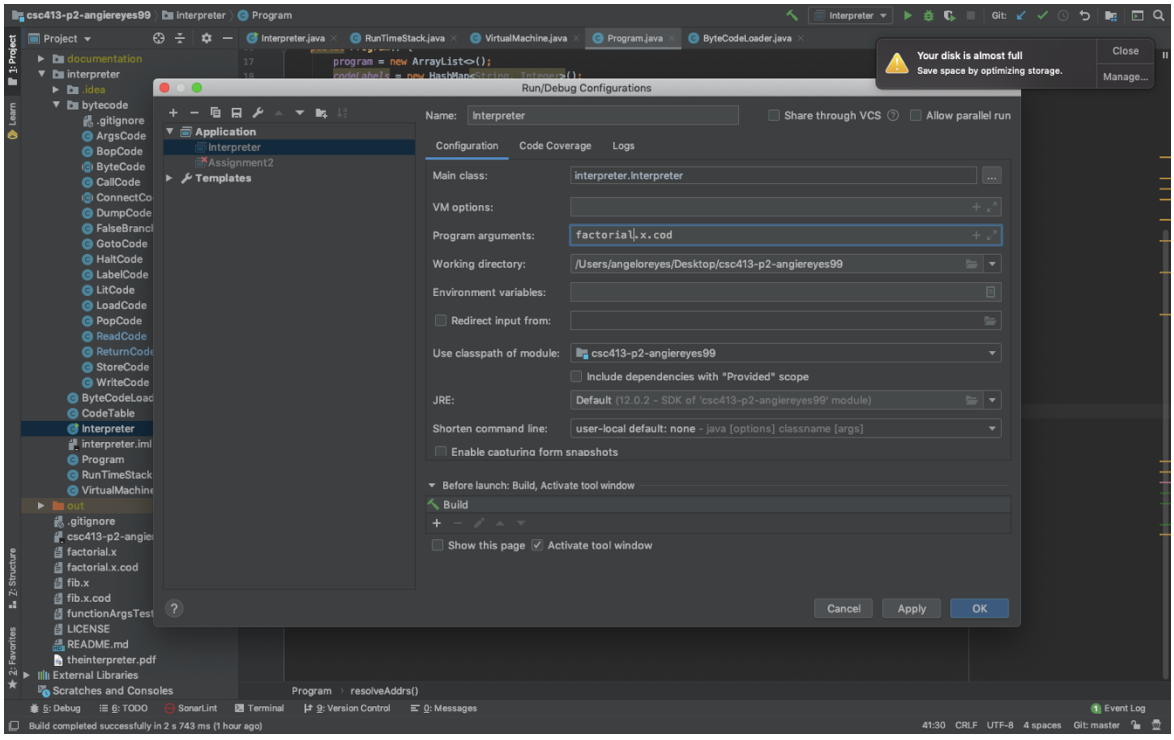
4 How to Run your Project

To run the project, make sure you run the Interpreter main method. In the edit configuration, under, “Main Class”, you should put in either the factorial.x.cod or the fib.x.cod. Hit “Ok” and run the program.







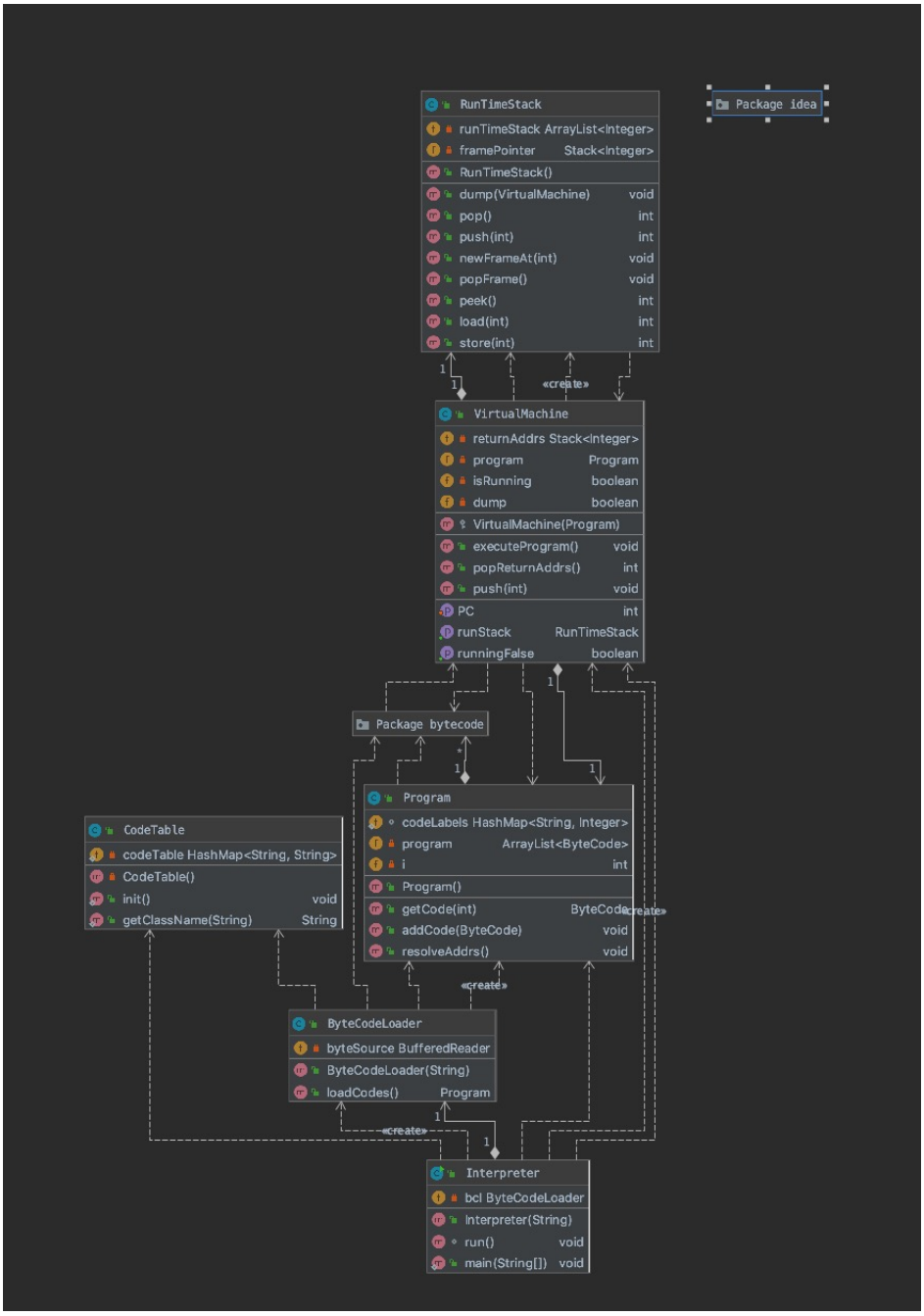


5 Assumption Made

Some assumptions I had for the class was the dump method, as I thought dump drops labels in the ByteCode. Another assumption I made was that the Interpreter should be outputting binary numbers after GOTO. I assumed that the subclasses were all being called in the Interpreter main class.

6 Implementation Discussion

6.1 Class Diagram



7 Project Reflection

Overall, the project presented to be much more difficult than I presumed. I was able to get all the ByteCode classes and also implement the correct functions for the Program, ByteCodeLoader, and VirtualMachine classes. Through the difficulty of the project, I had to use my debugging skills many times during the project and had to use problem solving techniques to

dive and conquer parts of the interpreter. The project was also hard to understand in the beginning when it was assigned which made the process much harder. I got help from tutors and also worked with classmates to code the project. Although the project was difficult, I got to learn and improve more of my skills in a project I'm not used to doing. I also got to collaborate with other individuals to help tackle this project, which made my communication better.

8 Project Conclusion/Results

My Interpreter successfully compiles and prints out the first few Bytecodes. However, after you hit Read code, the program stops and Return code does not execute. This could be due to my Return code not properly implemented. So, although my code compiles, it still does not correctly compute the fib.x.cod and factorial.x.cod classes correctly.