CSC 413 Project 2 Documentation

Summer 2023

Ruxue Jin

923092817

Class Section:01

GitHub repository Link:

csc413-SFSU-Souza/csc413-p2-RuxueJ: csc413-p2-RuxueJ created by GitHub Classroom

Table of Contents

1 Introduction

1.1 Project Overview

We are given a mock programming language X, and two programs computing Fibonacci and Factorial written in language X. This project serves as an interpreter/Virtual Machine for the mock language X. It processes byte code from source code (mock language X), and executes the code according to the logic, finally gives the right output.

1.2 Technical Overview

This project stands between modern programming languages and machine language. It examines the mechanisms how programming works in functions and logics: 1) what data structures are needed to store the data in functions. 2) how to pass parameters, call a callee function, and return a value to the caller function. 3) how to resolve the address of each code before execution of the program, so that programming counter can set to the target address efficiently.

1.3 Summary of Work Completed

Step 1: I iterate the source file by lines. Each line is a ByteCode command. I split the first token as ByteCode name, other tokens as arguments of the ByteCode. I create an instance of ByteCode according to the ByteCode name and its arguments.

Step 2: In the package bytecodes, I store all ByteCode classes. I design an interface ByteCode to abstract the getNewInstance and execute function, and an interface JumpCode to abstract functions for JumpCode, GotoCode, and FalseBranchCode, which all jumps to another resolved address. For each ByteCode, I create a class to design the function. Thus, I have the total 15 classes, 2 interfaces in this package.

Step 3: I have a List<ByteCode> in program class, which stores the ByteCode list from Step 1.

Step 4: After I get the List<ByteCode>, I resolve the address of each label ByteCode and put the jump code in HashMap with bytecode as key, target address as values.

Step 5: I created a RunTimeStack class to stores data: List<Integer> runtimeStack store variables, stack<Integer> framepointer stores the function scope, all functions needed to manipulate the data: push(), peek(), pop(), store(), load(), newFrameAt(), popFrame(), and getNewFrame().

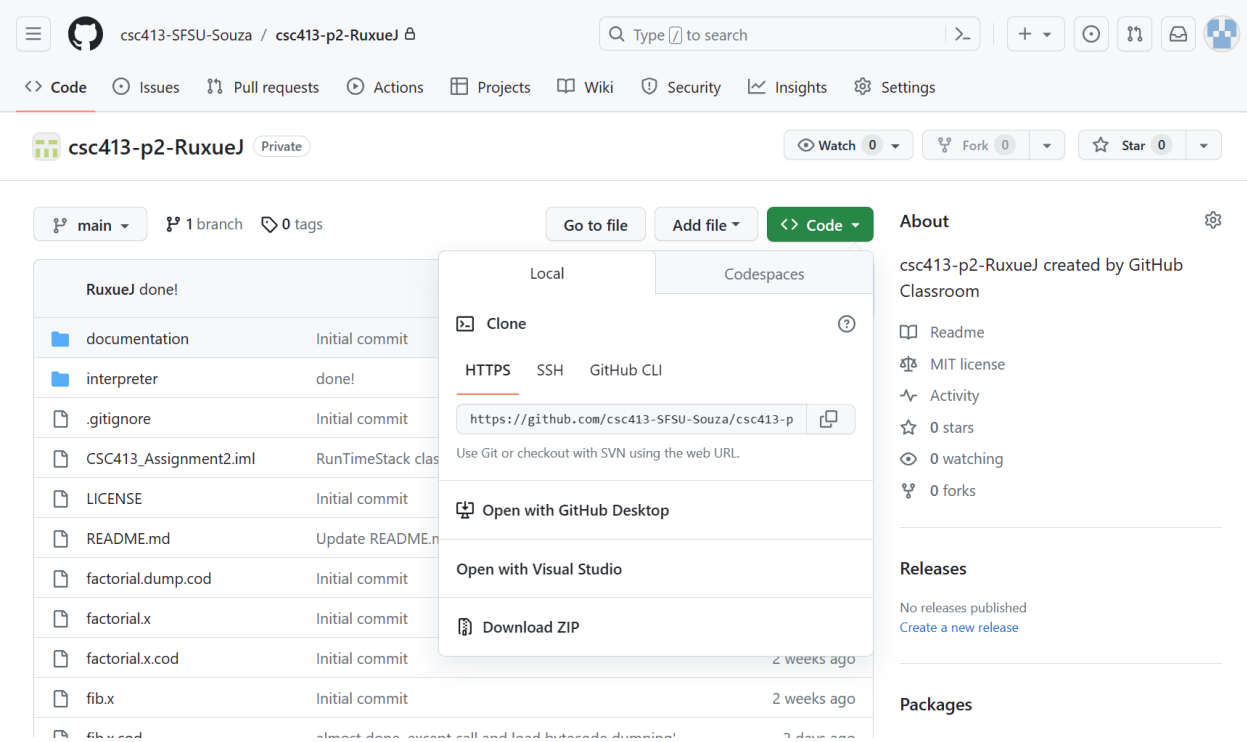
Step 6: After all the preparation, I execute the program in virtual machine. Get each bytecode from program and execute. Each Bytecode execute invoke VM to execute, and the VM calls certain methods in data structure RunTimeStack.

2 Development Environment

Java version: 17.0.6

IDE Used: IntelliJ IDEA 2022.3.2(Ultimate Edition)

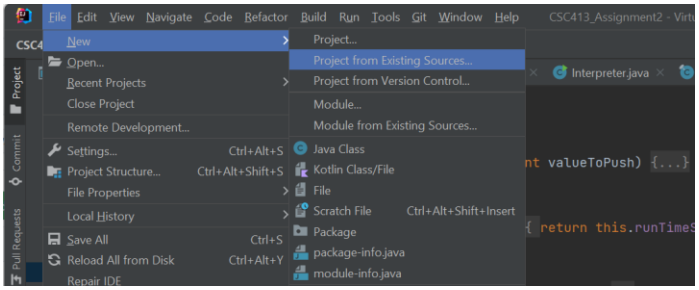
3 How to Build/Import your Project



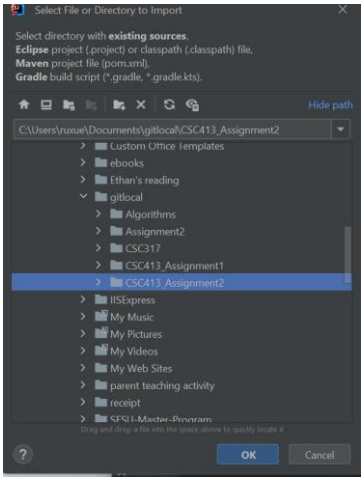
Click the green “Code” button on my repo’s home page. Then copy HTTPS.

In the terminal, cd to the folder you want to store the project.

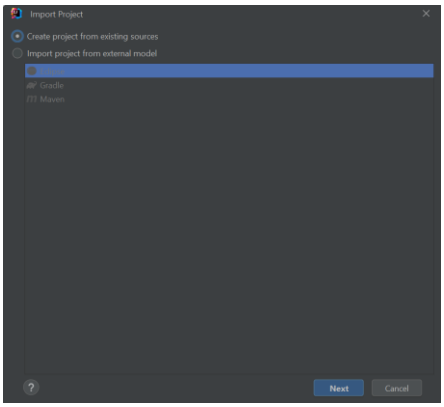
Then type: git clone repo_url_you_copied.



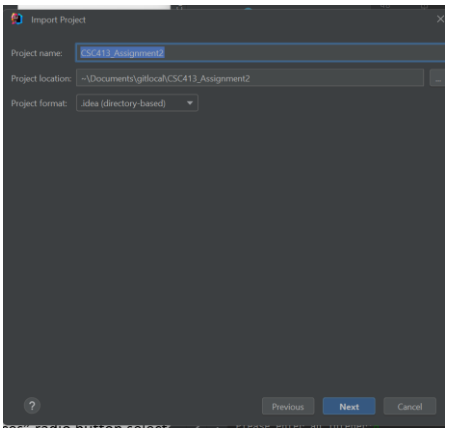
Open IntelliJ, click File -> New-> Project from Existing Sources…



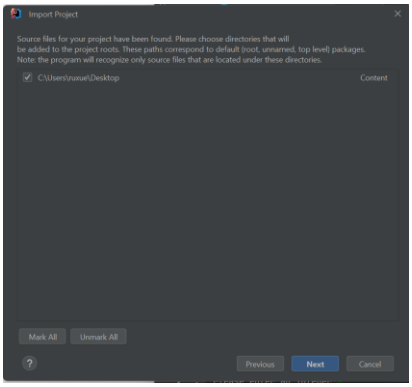
Select the root folder your store the project, click “CSC413_Assignment2” package, and click OK.



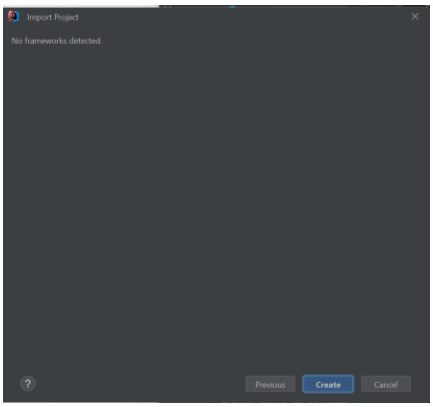
Keep the “Create project from existing resources” radio button selected.

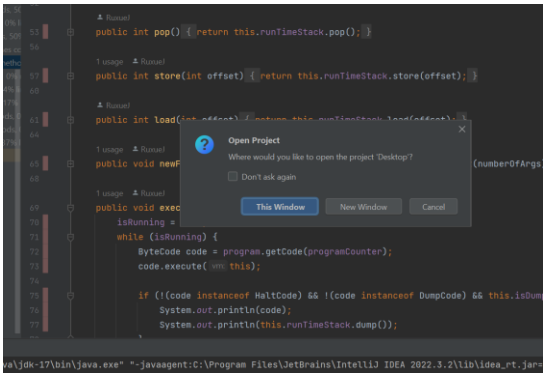


All default fields can be left alone here.

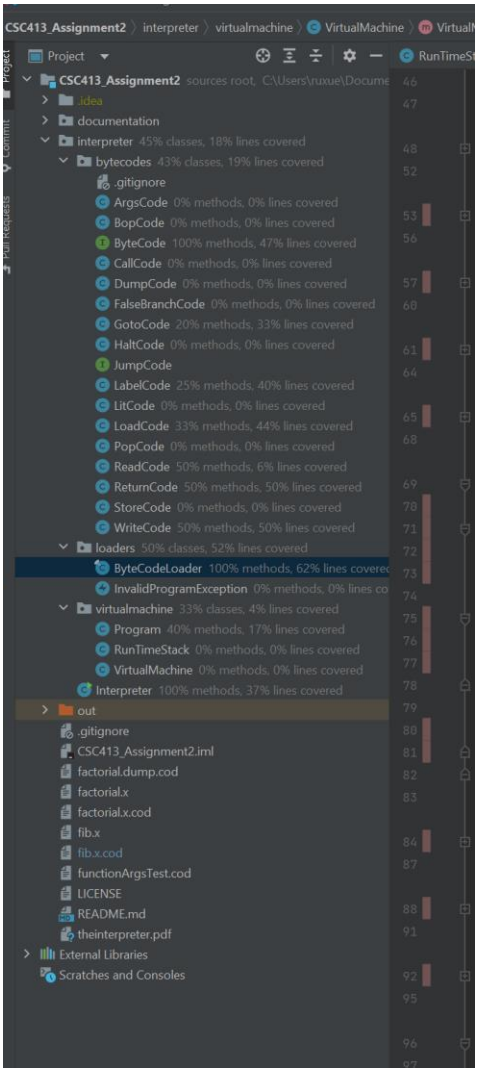


Select a location to store the project.



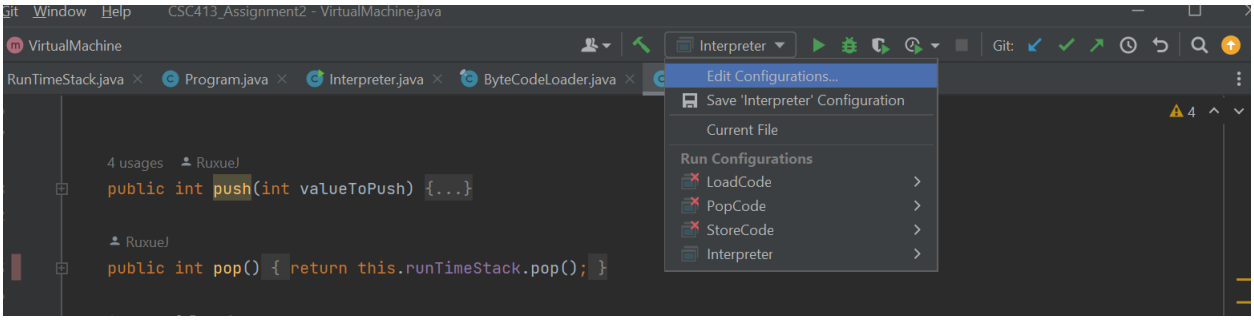


Click New Window.

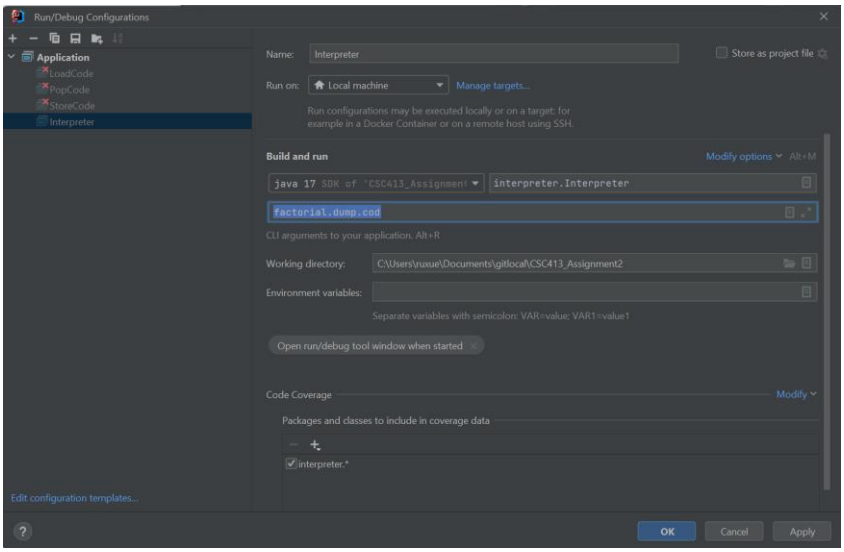


4 How to Run your Project

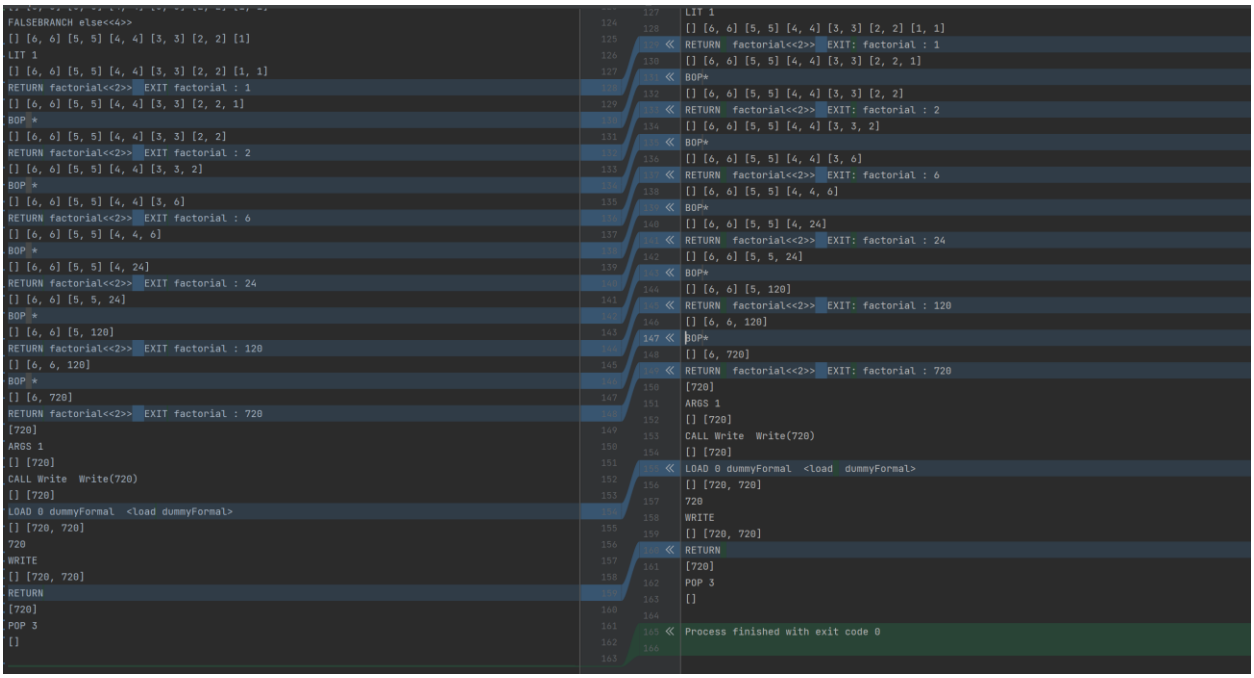
Test with factorial.dump.cod:



In the Interpreter class, click “Edit Configurations…”

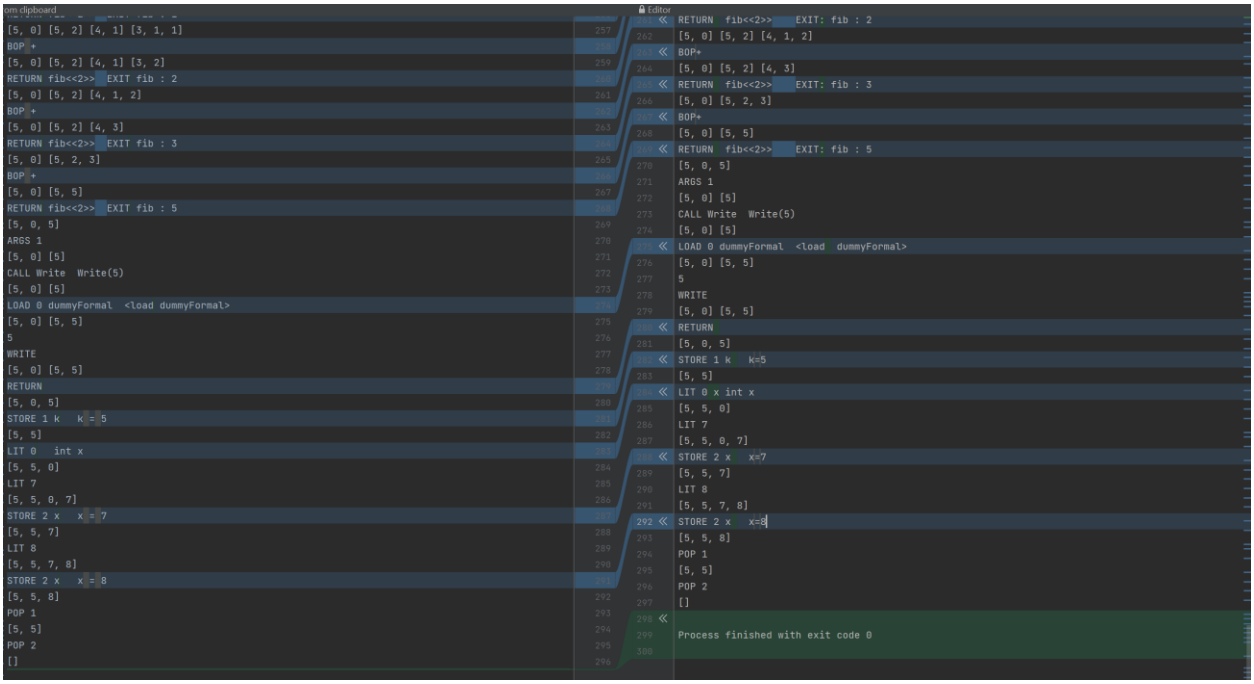


Type in factorial.dump.cod. Click Apply, and OK. Then run the program.input integer:6.

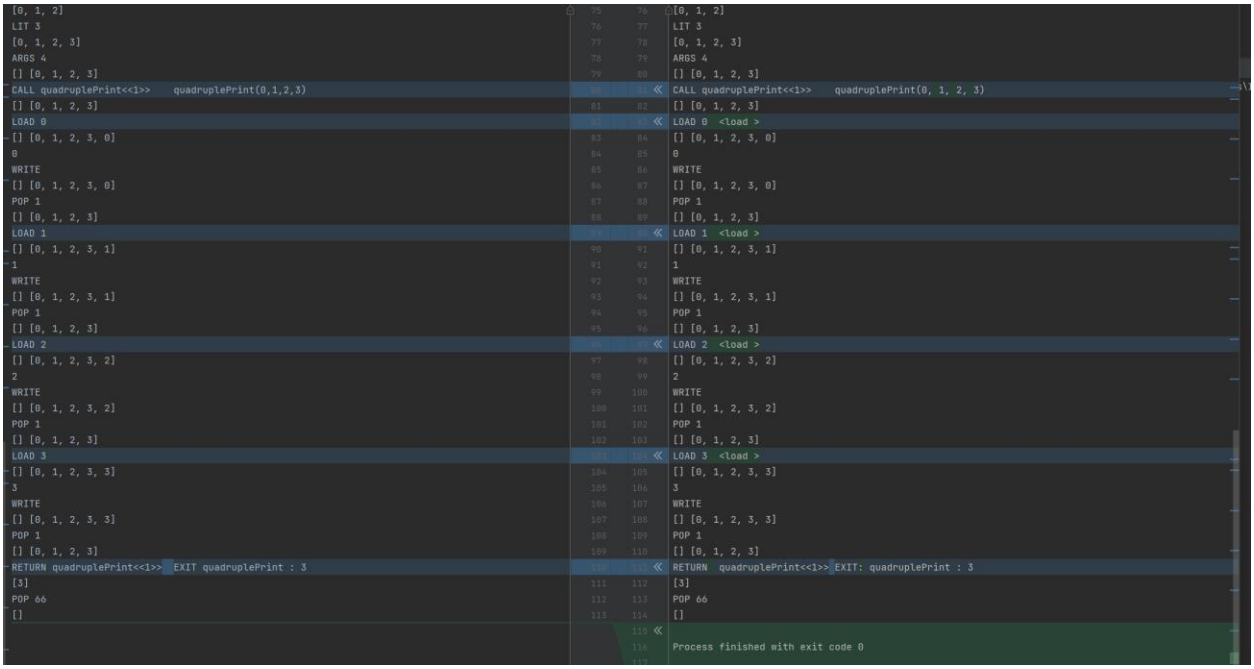


Test with fib.x.cod:

Same with the factorial.dump.cod, input integer:5.



Test with FunctionArgsTest.cod



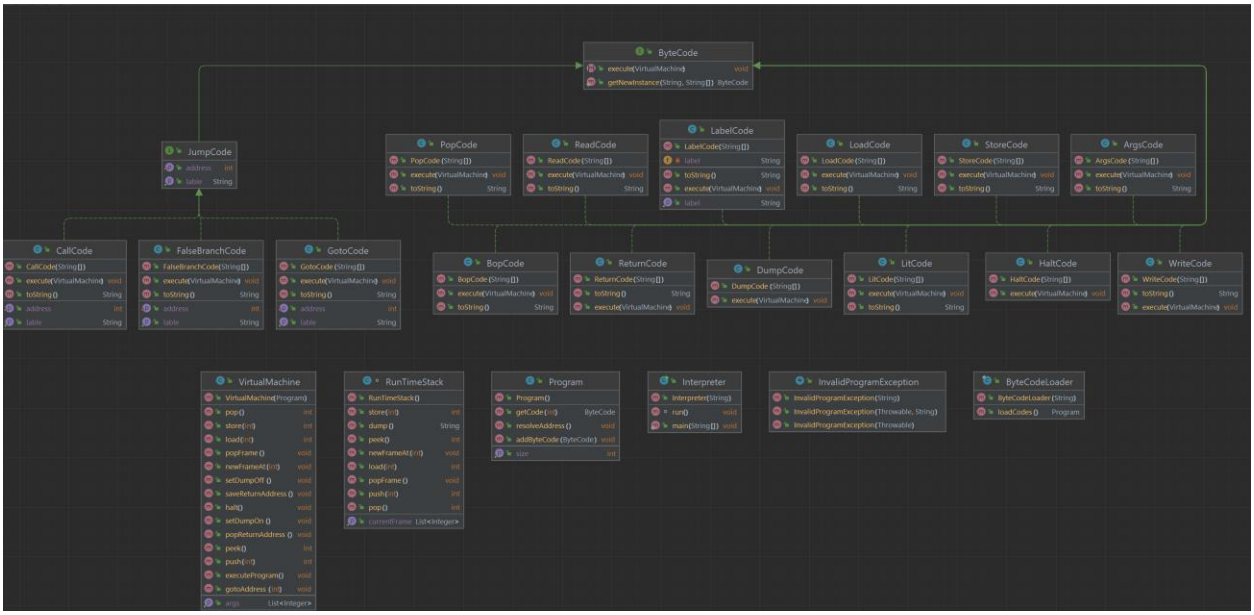
5 Assumption Made

The ByteCode .cod file is correct.

The arguments are integers.

6 Implementation Discussion

6.1 Class Diagram



7 Project Reflection

For this assignment, I feel it goes smoothly than the last assignment. I watched the first lecture video twice and fully understand how the program works. I read documentations for each bytecode before writing code. For each function and class, I test the code so make sure it works. So, it was not bad.

The lesson I learned from this assignment is that: understanding the process of the program, understanding the requirement, understanding each function and class, are essential to programming.

Think before writing the code, test after writing the code.

8 Project Conclusion/Results

The program works well!!