CSC 413 Project Documentation

Fall 2018

Jinghan Cao

918818659

Class.Section 01

GitHub Repository Link

https://github.com/csc413-01-fa18/csc413-p2-KamyC

Table of Contents

1 Introduction 3
1.1 Project Overview 3
1.2 Technical Overview 3
1.3 Summary of Work Completed 5
2 Development Environment 5
3 How to Build/Import your Project 5
4 How to Run your Project 5
5 Assumption Made 6
6 Implementation Discussion 6
6.1 Class Diagram 6
7 Project Reflection 7
8 Project Conclusion/Results 7

1 Introduction

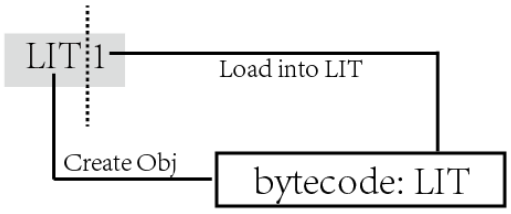
1.1 Project Overview

The whole project called “Interpreter” aims to decipher certain coding language X to get a computational result. X includes a group of Bytecodes such as LIT, STORE, RETURN, CALL etc. Each code contains a specific function to deal with the input. When we organize all these codes in some order and input a certain number, we will get a result after deciphering them. In this case, the project requires to translate the Bytecodes into Java and implement all the required functions of each code. For instance, a Fibonacci calculator is written in this X language and we are given its source codes which are Bytecodes. Then translating these Bytecodes and implemented in Java so that we could get a result.

1.2 Technical Overview

In this project, a bytecode table is created to store all the possible byte codes which will be encountered in the source file. Then each key will link to the name of each byte code class. While reading the lines of the x.cod file, the byte code objects are created and added to the program object. Meanwhile, parameters are taken out and loaded into each corresponding byte code object (see Fig1).

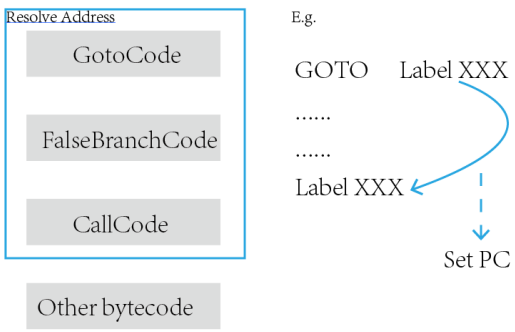
Fig 1



While loading the byte codes, the exceptions should be caught using try-catch just in case there are typo or reading errors in the source file.

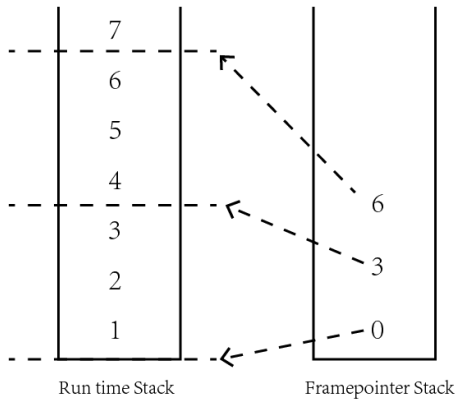
The program object mainly contains an array list to store all the Bytecode objects and also responsible to resolve the address. GotoCode, FalseBranchCode and CallCode all need to jump to certain label to execute the next line of code, therefore returnAddress Stack is needed to store the address, so that the program could return to its once address when this branch ends. While resolving the address, exceptions are captured to inform whether this jump action is proper or not. This process also requires the setting of PC so that the program know which line of code it should read for the next step.

Fig 2 Program ArrayList



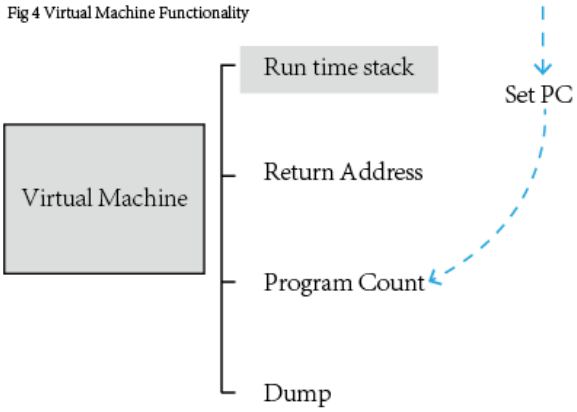
In the runtime stack class, two types of data structures are created. One is the array list to contain the input value and the temporary values in the process. The other one is a stack to store the frame pointer. Based on the assignment requirement for each byte code, the number in frame pointer refers to the frame index of frame (See Fig 3). But the runtime stack is an array list which provides useful API to accommodate the function of byte codes.

Fig 3 Run time stack and Framepointer



The virtual machine class mainly contains the methods to manipulate the runtime stack, program count, return address and dump (See Fig4).

Fig 4 Virtual Machine Functionality



Since virtual machine should be manipulated by byte codes, it should provide sufficient interface for each byte code to use. Firstly, there should be a switch-off flag to decide when the VM should turn on or off. Secondly, VM should execute the program and “activate” each byte code. In this process, the byte codes implement their initialization and start to do their job. Dump is a flag to determine whether the run time stack should be printed.

In the end, the interpreter class as the program entry will firstly initialize the code table and start to load the byte codes into program as well as resolving the address. Secondly, the virtual machine is activated and start to execute the program.

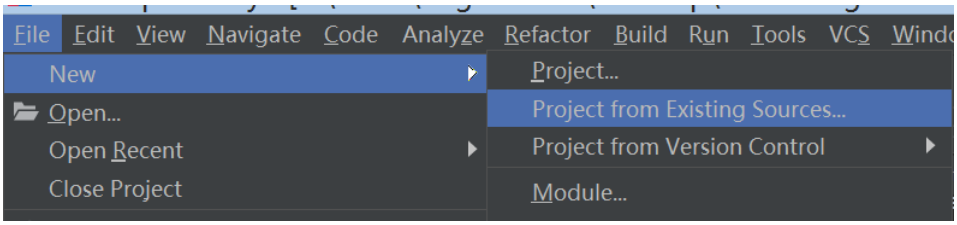
1.3 Summary of Work Completed

In this project, I have completed the required class and add enough byte code classes which are capable to deal with the factorial and Fibonacci .x.cod file. Among all the byte code classes, there is one abstract class and the rest 15 child classes inheriting from it.

2 Development Environment

I use IntelliJ to as the IDE and the Java version is 10.0.1. The operating system is Windows 8.

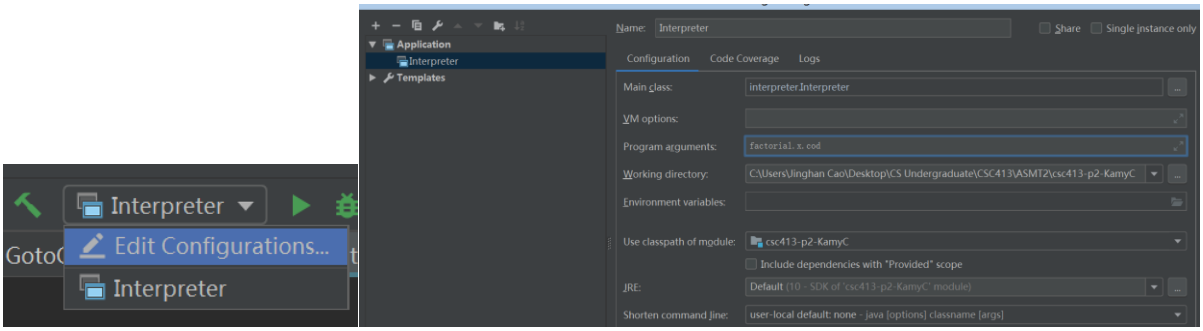
3 How to Build/Import your Project

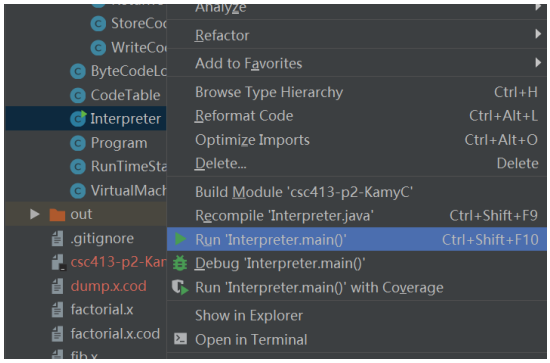


After cloning from Github, you can open the IntelliJ to find the directory of the repo and import it into the IDE.

4 How to Run your Project

Before running, you need to set the configuration and add the parameter source file. Once you have set the Java version to this project and the icon of each class is ready, you can start to run it.



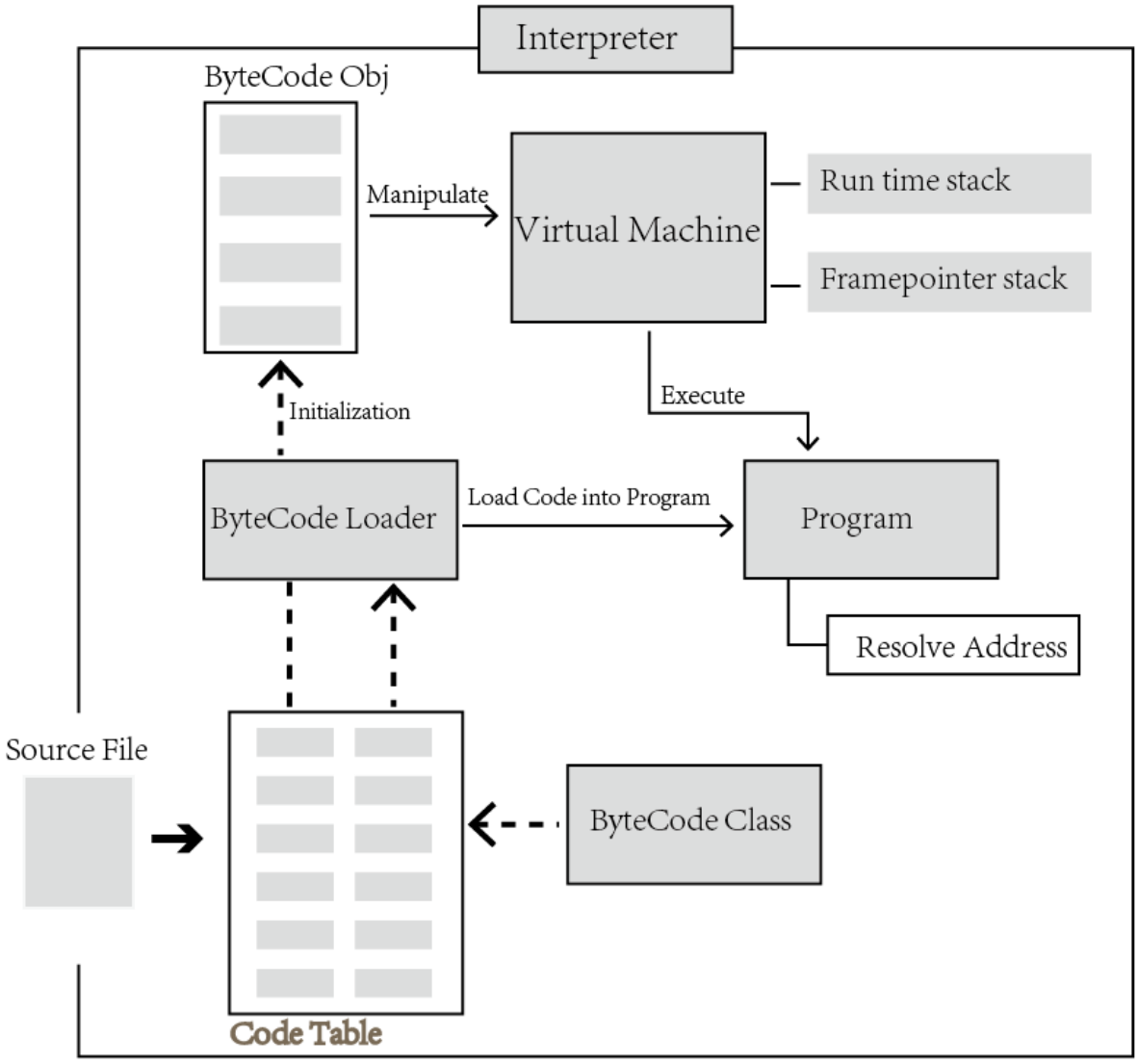


5 Assumption Made

I assume that the hard part in this project is the relations between byte code class and virtual machine class. The requirement indicates that the byte code should send requests to the virtual machine. And VM should then deal with the run time stack. Therefore based on this starting point, the VM plays an intermediate role between the byte code and run time stack. Another assumption is about the reading input. In this project, I should assume the source file code is correct. Then there should be a try-catch to avoid the possible input error. Once the source file format is wrong, warning will be generated to inform the user.

6 Implementation Discussion

6.1 Class Diagram



7 Project Reflection

I have learned the importance of encapsulation in a program development. In this case, each component is encapsulated and provide interface methods for other classes to invoke. The byte code sends the requests to the virtual machine which will execute the program and deal with the run time stack. I also have learned that it is necessary to split one project into several components or modules for its flexibility. If any changes should be made, it will cost less time than the approach of putting everything together. The design of the entry class should be simple and let the other classes to do the hard work.

8 Project Conclusion/Results

The program is capable to read and process each byte code in the source file and generates correct computational result. The dump method in run time stack class also prints out a visible result to display how the byte code manipulates the virtual machine which successively controls the run time stack and frame pointer stack.