CSC 413 Project Documentation

Summer 2020

Tin Thu Zar Aye

920615641

CSC413-02

https://github.com/csc413-02-SU2020/csc413-p2-TinThu.git

Table of Contents

1 Introduction

This project is called the “Interpreter”. We create the program as an interpreter for an mock language X.

1.1 Project Overview

- This project is called “Interpreter”. Basically we created the program as an interpreter to simplify version of java. We are creating the ByteCodes from the sources code file to the entire program.

1.2 Technical Overview

- In this project, we have to implement the RunTimeStack class, ByteCodeLoder class, Program class, VirtualMachine class and ByteCode class which are including the abstract ByteCode class from the CodeTable class. First of all, we have to implement the RunTimeStack class to record and process the stack of active frame. Secondly, we implemented the ByteCodeLoder class which is responsible for loading ByteCode from the source code file into a data-structure that stores the entire program. Program class which will be responsible for storing all the ByteCodes read from the source file. Then we have to implement Virtual Machine class which is used for executing the given program. Lastly, we created the Dump ByteCode to determine the VirtualMachine should dump the current state of the RuntimeStack and the information about the currently executed ByteCode.

1.3 Summary of Work Completed

- There are 4 main classes that we have to implement in this project which are RuntimeStack class, ByteCodeLoder class, Program class and VirtualMachine class. There are also including the test classes which are factorial.x and fib.x. We have to implement step by step, for example, first we have to implement the RuntimeStack clas; second, we have to implement the ByteCodeLoder class which is responsible for loding ByteCodes form the source code file into a data-structure that stores the entire program. Thirdly, we have to implement the program which to get the address of each ByteCode. This program class has to import to the VirtualMachime class because the VirtualMachine class is returning the whole program. Then, we import the RuntimeStack class to the VirtualMachine class to complete the each ByteCode sub-classes.

2 Development Environment

- I am using the IntelliJ IDE version – 2019.3.5. Build #IC-193.7288.26
- Java version 13

3 How to Build/Import your Project

- When I was importing the project, I copied the link from the GitHub repository and clone in my computer’s desktop them import to the IntelliJ. When I was importing the interpreter project, I used the root of my repository as the source instance of using the interpreter folder as the root.

4 How to Run your Project

- We can not run the project if there is no complete implementation. First of all, we have to change the configuration either factorial.x.cod or fid.x.cod.
- Then, before we run, we have to make sure that RunTimeStack class, ByteCoderLoder class, Program class and VirtualMachine class are working properly.

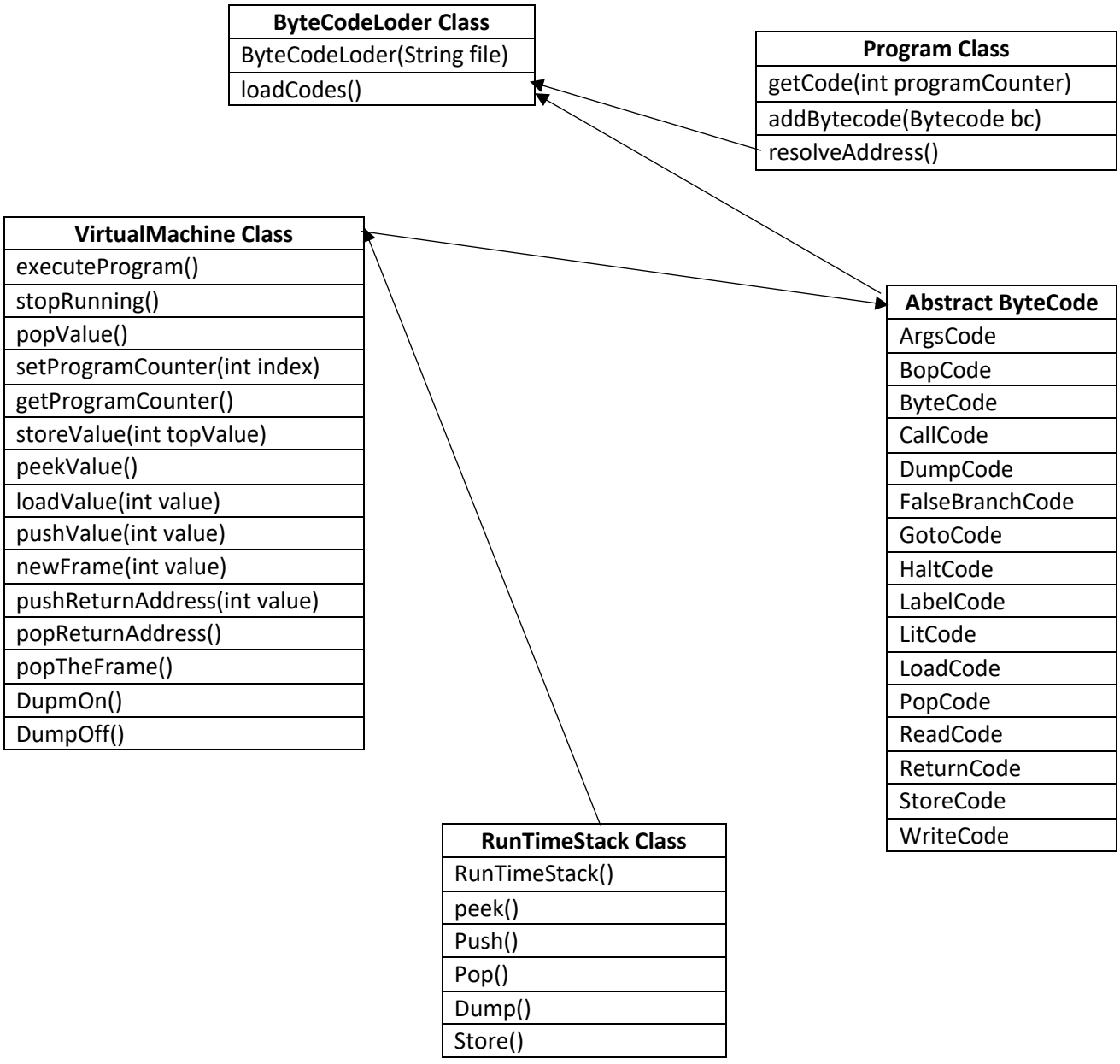
5 Assumption Made

- At first, I have to implement the RunTimeStack class as the video mentioned.
- Second, I created the abstract ByteCode classes and tried to implement the init functions.
- Third, I tried to implement the ByteCodeLoder class but the ByteCodeLoder class can't not be fully implement without completing the Program class because Program class is solving the address of each ByteCode. So, I tried to implement the resolveAddress function then call the resolveAdress function to the ByteCodeLoder class to get all the address of the ByteCodes.
- Then I tried to implement each ByteCode. When we implement the abstract ByteCode, we have to import the RunTimeStack class to the VirtualMachine class to access the related function. Then we import the VirtualMachine to the abstract ByteCode classes. We have to import the RunTimeStack class to the VirtualMachine class then import the VirtualMachine class to the abstract class because the abstract classes can to access the RunTimeStrack directly.
- Finally, I implement the Dump ByteCode.

6 Implementation Discussion

- Implement the RunTimeStack class.
- Then I create the ByteCode classes which listed in the CodeTable class but there is no complete implementation yet.
- After that I tried to implement the ByteCodeLoder class and at the same time I had to implement the Program class because I have to call the "resolveAddress function" from the Program class to the ByteCodeLoder class to get the ByteCode's address.
- Then I implement the VirtualMachine class by calling the functions from the RunTimeStack class to complete the abstract ByteCode classes.
- After I can compile without any error, I tried to implement the dump function in the RunTimeStacke class and implement the Dump ByteCode.

6.1 Class Diagram



Load()
newFrameAt(int offset)
popFrame()

7 Project Reflection

- After I watched the introduction video for the project, I had to read the whole assignment question and tried to understand the requirement. I took so much time to understand the requirement and to think the program design. Implementation all the abstract ByteCode classes is the most difficult part for me. Slack is also helping me a lot to have done the project.

8 Project Conclusion/Results

- To conclude, I think my project is perfectly working. As a result, I got the correct answer for the factorial.x.cod and fid.x.cod that the provided in the project.