Mak, Ho Yin
913340927
CSC413-01, Summer2019
https://github.com/csc413-01-summer2019/csc413-p2-hoyin0929

Introduction:

The purpose of this program is implementing an interpreter to interpret a mock language called "X". This project needs to be complete with a partially complete program, and it requires us to extend different subclass to implement different bytecodes under the sources file. Each subclass we created will implementing different x-machine bytecodes so that the program can handle operation in x code. Then we need to edit the Bytecode Loader class to load bytecode, Program class to store all the bytecodes read from source file, Run Time Stack to help the virtual machine run, and Virtual Machine to let all operation to go through. After all these have completed, it will work together to run a program written in the language X.

Development environment:

Java Version: 10.0.2
IDE: IntelliJ IDEA 2018.3.4 x64

Build:

- Inside IntelliJ, click File on the top left corner then click New -> Project From Existing Sources
- Choose the directory of the saved program, then click OK
- Keep click next until you click finish
- When the project shows up on IntelliJ, click Build on the top taskbar, then click Build Project. The Project can build successfully if there is no error.

Run:

- After the project has successfully built, we can run the Interpeter.java to see if the program can run appropriately with finding the nth Fibonacci number or the factorial of a number.
- We must edit the configuration before we run Interpeter.java. In the configuration, we need to enter "fib.x.cod" or "factorial.x.cod" in a field called "Program arguments:", for this is a program written for x language.
- When the program is successfully run, the program will ask user to enter a number, and the output will be displayed after the input has been received. The output will be the nth Fibonacci number or the factorial of a number depends on the program arguments which entered by user.

Assumption:

I made a wrong assumption that I can access some classes directly. However, during the process I worked, I found out that it will end up break the encapsulation

Implementation Discussion:

I use the following steps to implement my program as the following. First, I create all bytecode class with empty methods in the sources file named bytecode which allows all other class can access them. After the empty bytecode classes have been created, we can finish the BtyeCodeLoader by completing the loadcodes() methods. The loadcodes method use local buffer reader object to make an instance of bytecodes subclasses. Then, I implement the Program class which is mainly about the resolve address. This class uses array list to store all the bytecodes read from sources file. After I finish modified the Program class, I implement the RunTimeStack class by adding the dump function to print out the stack with brackets and commas. After that, I complete the Virtual Machine by making it to run methods from RunTimeStack. Virtual Machine is the controller of this program. To do this, I add many other functions to allow all operation can pass through this class. Lastly, I can finish the bytecode class by adding code to the empty method bodies. I attached a class hierarchy at the end of the documentation to understand the implementation better and easier.

Reflection:

When I just started the project, it is very challenging for me compared to all my computer science projects, for its complexity and size. It makes me have a lot of confusions. To understand better of this project, I read the pdf few more times later, and it helps me a lot. The pdf includes a lot of details on what is the responsibility for each class, and it provides the programming steps to follow.

Conclusion and Results:

Each struggle I made during this project definitely grows me a lot as a programmer. It helps me understand the consequences on minor change in a large size program. I am glad that we have a slack channel, for it solves most of my confusion.

Class Hierarchy Diagram