CSC 413 Project Documentation

Spring 2022

Beatrice Aragones

920614978

CSC413.02

https://github.com/csc413-SFSU-Souza/csc413-p2-bearagones.git

Table of Contents

1 Introduction

1.1 Project Overview

The purpose of the project was to implement an interpreter for a mock language X, which we can think of as a simplified version of Java. It allows the user to run three different files to test the program itself, which are “factorial.x.cod”, “fib.x.cod”, and “functionArgsTest.cod”. For the first two files, when the user runs the program, it allows the user to input an integer and have the program output the factorial or the Fibonacci series of the integer. In all three outputs, the user will be able to see what is going on in the program through a series of instructions.

1.2 Technical Overview

The purpose of the project was to practice programming using our knowledge of various data structures, as well as practice our knowledge of object-oriented programming once again, such as focusing on not breaking encapsulation. The program implements the use of various data structures, such as Stacks, ArrayLists, and HashMap. A HashMap was used in the CodeTable class in order to store the data that was being read in line-by-line by the ByteCodeLoader class. The VirtualMachine class was the primary controller of the program that enabled its execution, and it implemented many different functions to communicate between other classes. The purpose of the Program class is to store and hold the ByteCodes that were loaded from one of the three test files. The RunTimeStack class was used to process the stack of active frames in the program. Lastly, we implemented an interface of ByteCode and used it to support various kinds of different ByteCodes and its usages.

1.3 Summary of Work Completed

Overall, I believe that I was able to complete the entire program. After running the three different test files, the program can compile each of them and get the expected output. When typing in an integer for “factorial.x.cod” and “fib.x.cod”, I can see the process of the calculations, and it matches with the sample outputs that were provided in the Discord channel. When testing out “functionArgsTest.cod”, I also get an output that resembles the sample outputs. I also made sure to read the Dumping Formats and check if my output followed the directions, which I think may be correct.

2 Development Environment

This program was implemented in IntelliJ IDEA 2021.2.1 (Ultimate Edition) on Java 14.0.2

3 How to Build/Import your Project

To build and import my project from GitHub to your IDE such as IntelliJ, you first need to clone the repository. First, click the green “Clone” button on the right side of the page and copy the link. In the terminal, then type “git clone [insert report URL link here]”. The repo will be placed in a folder that you have to locate when importing the project to your IDE. After opening your IDE, select “New…” → “Project from existing sources…”, then locate the folder where the repo was cloned. Here, be sure to click on the repo folder itself and select it as the source root of the project. This is key for it to build and run properly. When creating the new project, you can leave the everything as is. When selecting the JDK version, I chose 14. Now, the importing process is finished, and the project is imported into your IDE.

4 How to Run your Project

The instructions to run my project are a bit different compared to how my project was run for the previous calculator assignment that I had done. This time, you are going to need to make sure that you have java installed, then type in the command line the following commands (although, this may be unnecessary if you are already able to see the classes directory in the GitHub repository):

```
cd csc413-p2-bearagones
mkdir classes
javac -d classes -sourcepath . -cp .classes/ interpreter/Interpreter.java
cp *.cod classes
cd classes
java interpreter.Interpreter [insert .cod file name here]
```

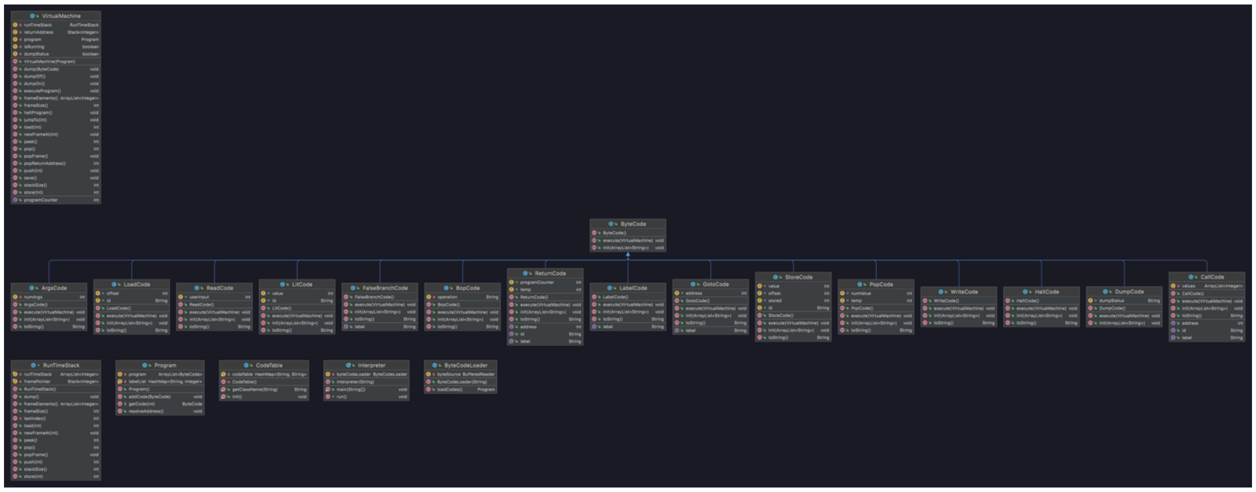
Once these commands have been entered in the terminal, you can then locate the “Interpreter” drop down menu at the top of your IDE (assuming you are using Intellij), and then click on “Edit Configurations…”. Once you have clicked on this, you can type in the name of the .cod file that you would like to test in the “Program Arguments” textbox. Also, make sure that the working directory is correct and is the source root of the project. Now, you should be able to get an output from either of the three files that you have chosen!

5 Assumption Made

When running the program, there are many assumptions to be made. First, assume that the .cod files are correct and contain no errors. Additionally, assume that there is no need to worry about errors such as division by zero as it is not included as one of the test cases in one of the .cod files. For testing “fib.x.cod” and “factorial.x.cod”, assume that you can only input integer values. If any other type of data were inputted, such as letters or random tokens like parentheses or commas, then there will be exceptions thrown. One last assumption that has to be made is after resolving the addresses in the program, the labels of the FalseBranchCode will change from labels to integers.

6 Implementation Discussion

6.1 Class Diagram



6.2 Design Choice

When working in the VirtualMachine class and the RunTimeStack class, I chose to implement similar or the same functions. The reason why I did this was to not break encapsulation. By implementing similar or the same classes, the VirtualMachine class can then request to perform operations so that the ByteCode classes and the RunTimeStack class do not interact with one another. This can be seen in the UML diagram that I included above, as well as in the documentation folder itself for easier viewing (if it helps). For the supported ByteCode classes, each class uses the interface of ByteCode.java as an abstract class (parent class). The supported ByteCode classes are then child classes with their own different executions and inits. Each child class also implements an Overrided toString() method in order to correctly print out the process and instructions that are happening in the program.

I also wanted to point out my implementation of BopCode.java, which is one of the supporting ByteCode classes. For this file, I used a HashMap to perform the operation because I felt that it was a lot more cleaner and easier to understand/read what was happening in the class with the different operations. Lastly, in the resolveAddress() method in the Program class, I used a label list HashMap where the keys were the String labels, and the indexes were the values. Then, if the ByteCode were either an instance of CallCode, ReturnCode, GotoCode, and FalseBranchCode, the program would set the label as the new address.

7 Project Reflection

To be quite honest, I found this assignment to be very overwhelming, since it was a bit confusing to me and hard to understand what was exactly going on in the program. It was also one of the first assignments that had a huge pdf attached to it, and it took me quite a while to go through the entire pdf the first time before looking over it again several more times. Like I said before, however, I found it really helpful to have all of the hints that were given in the various videos, as well as throughout the entire pdf. Without the detailed instructions, I simply wouldn't have known what to do or where to even start in the first place.

One of the biggest struggles I had in this assignment was just figuring out the dump() method in the RunTimeStack class. I ended up using a lot of if statements in order to get the correct output for this method. One last thing that I really struggled with was implementing the supporting ByteCode classes. I had some personal difficulties trying to understand what methods I should be implementing from the VirtualMachine classes in order to have them execute properly. However, with debugging, I was able to see where the problems were occurring and fixed them to the best of my ability.

One of the last struggles that I wanted to mention was time. I ended up inadvertently procrastinating on this assignment, not because I didn't feel like doing it, but because I had a lot of assignments going on at the same time that needed to be done. I ended up focusing on the assignments that I knew I could finish in a relatively short amount of time, but since I kept getting new assignments right after, I ended up working on this assignment close to the last minute. I learned that I shouldn't always push back working on larger assignments in favor of just finishing shorter assignments to cross them off my to-do list.

8 Project Conclusion/Results

Overall, although this second assignment was difficult, I did enjoy the challenge that came with it. Since this program involved making our very own interpreter for a mock language X, I felt that I was able to understand more about java, even though the mock language X is a simpler version of Java. I can't say that this was one of my favorite assignments to do, but I do feel that I learned a lot more as I progressed through the assignment. When the next assignment comes around, I'm definitely going to do my best to get some sort of work done every few days, even though it may be a very long or big assignment.