CSC 413 Project 2 Documentation

2023

Seyoung Kim

923109262

CSC413-02

GitHub Repository Link

https://github.com/csc413-SFSU-Souza/csc413-p2-pyoumg

Table of Contents

1 Introduction

1.1 Project Overview

This program can run a program written in language X. The file for this interpreter has the extension x.cod.

1.2 Technical Overview

This is an interpreter for the mock language X. Each ByteCode command implements interface ByteCode. Before executing, this program resolves the address using Hashmap. Each byte code in a program is stored in a list. In VirtualMachine, runTimeStack is implemented as a List, and framePointer is implemented as a Stack.

1.3 Summary of Work Completed

This program can run a program written in language X. This reads the code, resolves the address, and performs the appropriate command for each command.

2 Development Environment

a. Version of Java Used: Oracle OpenJDK version 19.0.2

b. IDE Used: IntelliJ IDEA 2022.3.2 (Ultimate Edition)

3 How to Build/Import your Project

You will need to create the run configurations and set the command line arguments. The Interpreter is only able to handle the .x.cod files. Press 'Build project' button to build the project.

When importing the interpreter project, you will use the root of your repository as the source. Do not use the interpreter folder as the root.

4 How to Run your Project

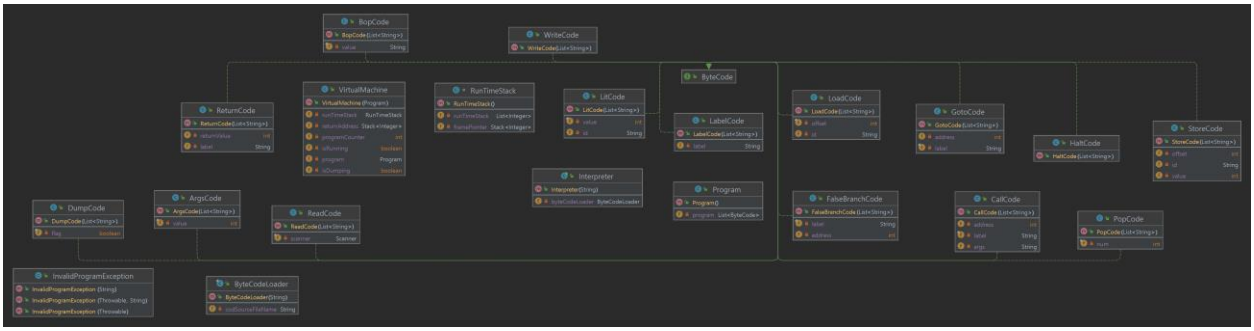
You will need to create the run configurations and set the command line arguments. The interpreter is only able to handle the .x.cod files. Press 'Run Interpreter.main()' to run the project.

5 Assumption Made

- The given ByteCode source programs (.cod files) for testing are generated correctly and contain no errors.
- No ByteCode can pop past any frame boundary.
- Encapsulation should not be broken.
- Each ByteCode class should have fields for their specific arguments.

6 Implementation Discussion

6.1 Class Diagram



6.2 Design Choice

a. VirtualMachine

- If private Boolean isDumping is true, the program dumps after executing the code.
- In resolveAddress(), the program resolves the address for Goto, Call, and FalseBranch bytecodes. In pass 1, the program stores the labelcode address, and in pass 2, it finds the address stored in the array and sets the addresses of Goto, Call, and FalseBranch bytecodes.
- In dump() function, it connects each element in the runtimeStack in order to make a single string and then return it. This function uses get() to get the element of the index.
- In VirtualMachine, functions that are called at the ByteCode are implemented. These functions call runTimeStack methods. And each functions are for encapsulation and are not duplicate.

b. loaders

- In loadCodes(), it reads each line, splits the key and arguments, stores them, and gets a new instance.

c. bytecodes

- ByteCode is an interface, and each ByteCode command class implements ByteCode.
- In PopCode, execute function checks if this operates across frame boundaries.
- Falsebranch, Goto, Call bytecodes have instance variable address to store resolved addresses.
- All Bytecode classes can’t detect dumping flag, so tostring() method is implemented in each class.

7 Project Reflection

It was a great chance to learn about the implementation of encapsulation and inheritance in JAVA, and I gained insight into how OOP is applied in Java.

8 Project Conclusion/Results

This program accurately executes factorial.x.cod and fib.x.cod.

factorial 12

```
Please enter an integer: 12
479001600

Process finished with exit code 0
```

fib 10

```
Please enter an integer: 10
55

Process finished with exit code 0
```

This program dumps only if the dump flag is on.

factorial 5

```
LOAD 0 dummyFormal   <load dummyFormal>
[] [120, 120]
120
WRITE
[] [120, 120]
RETURN
[120]
POP 3
[]
```