CSC 413 Project Documentation

Fall 2019

Erik Chacon

920768287

CSC 413.01

https://github.com/csc413-01-SU2020/csc413-p2-SaintGemini

Table of Contents

1 Introduction

1.1 Project Overview

The goal of this project was to create an Interpreter for the mock language “X”. It is supposed to read and run code that is like the code given in the fib.x.cod and factorial.x.cod files.

1.2 Technical Overview

The goal of the interpreter was to read and process byte codes so the computer could execute the given program. The interpreter will parse through each line and process each byte code individually. Each byte code behaves differently and will have a different implementation of the same functions. Each byte code is a child class of a parent ByteCode class.

1.3 Summary of Work Completed

A ByteCode parent class was created for all byte codes specified to be given to us. There is only one other parent class, BranchCode, which is a subclass of the ByteCode parent class. BranchCode deals with the byte codes that have branch instructions (CallCode, FalseBranchCode and GotoCode).

All ByteCodes were implemented to specifications.

The ByteCodeLoader class contains the loadCodes() function. Thankfully, about 80% of the code was given to us in a video lecture but the remaining 20% was implemented with comments to explain.

The VirtualMachine and RunTimeStack class have the majority of the added functions which were not specified in the requirements. Both the executeProgram() function in the Virtual Machine class and the dump() function in the RunTimeStack class were finished.

The CodeTable and Interpreter class were not to be touched.

2 Development Environment

I used Java 12.0.2 and IntelliJ IDEA 2020.1.2 (Ultimate Edition) for building, debugging and testing this project.

3 How to Build/Import your Project

1) In the project repository on github, download the zip or clone the project to your desktop.
    a. ZIP: unzip the contents of the folder into your desktop.
    b. Clone: Using git bash, change into desktop directory and type the command (do not include square brackets for link): git clone [http or ssh link given by github]
2) In IntelliJ, import project from existing code and select the root folder of the project as the root directory for the project.
3) In the “Build” tab in IntelliJ, press the “Build Project” button.

4 How to Run your Project

From within IntelliJ:

1) In the “Edit Configurations” tab, click on edit configurations
2) Click on the “+” buton and click on “Application”
3) Name the configuration “Interpreter”
4) In the main class section, put “interpreter.Interpreter”
5) In the program arguments section, either put “fib.x.cod” or “factorial.x.cod”. If left blank, there is no code for the interpreter to interpret.
6) The classpath of module tab should be set to the root folder of the project.
7) For JRE, use the latest version of java installed on your computer. It is recommended to use Java version 12 or later.
8) Click “Apply” and then “OK”
9) Hit the run button next to the Edit Configurations button.

5 Assumption Made

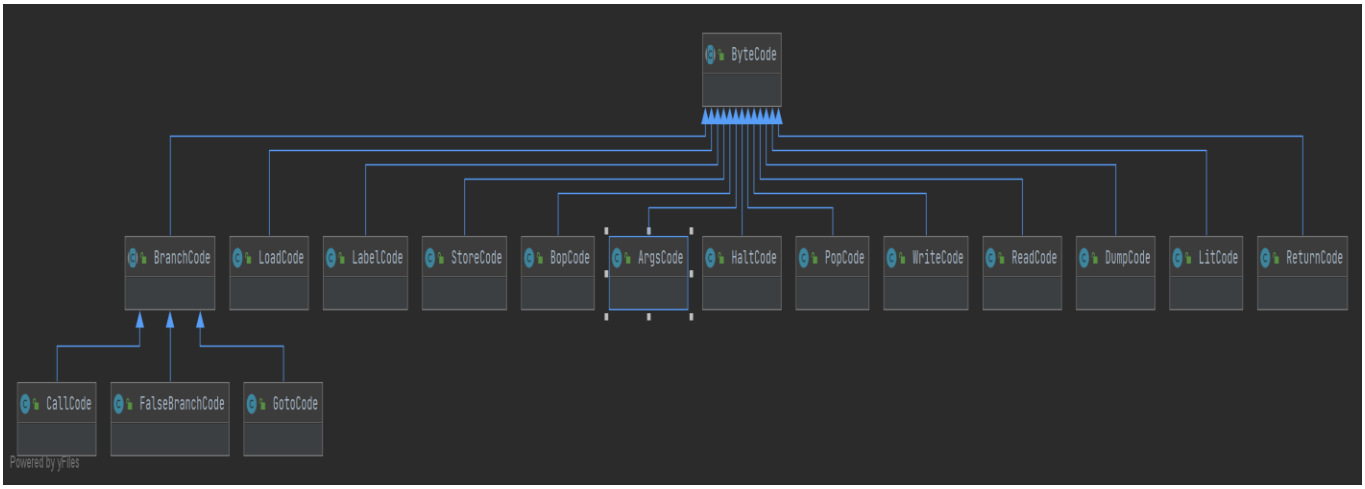
I assume all integers passed through the interpreter are non-negative.

6 Implementation Discussion

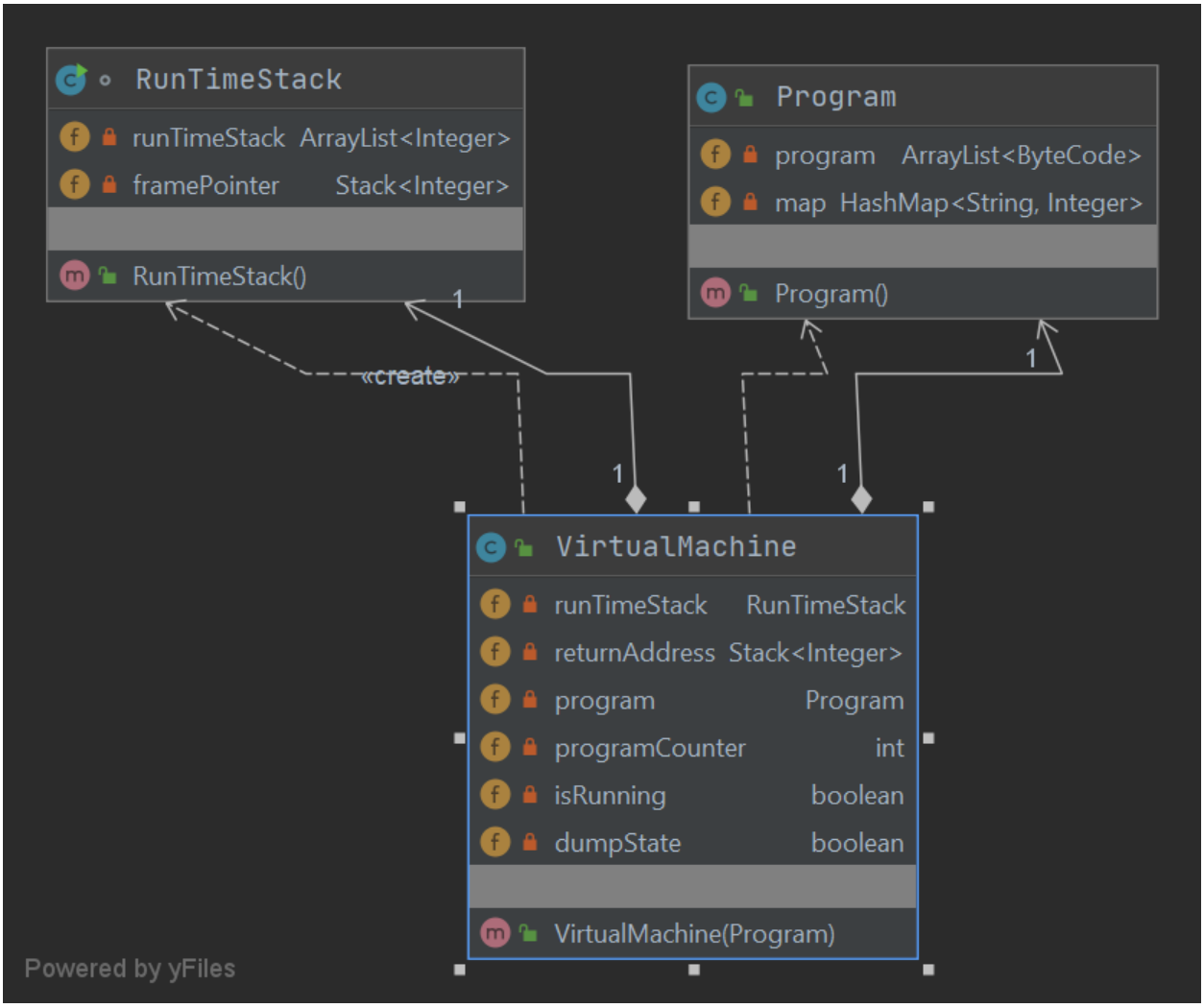
When implementing all of the ByteCodes, the idea was given that we should have a parent class for all byte codes with abstract methods. I decided to make another one for the byte codes with branch instructions. BranchCode is a subclass of ByteCode but a parent class to CallCode, FalseBranchCode, and GotoCode. They only feature two more functions, getByteCode() and setIndex(). It was not completely necessary to create the parent class BranchCode for a smaller interpreter project with only 3 byte codes adding two functions each, but it is good programming practice to by dynamic and scalable. The function getByteCode() returns a string and literally returns the name of the bytecode. The setIndex() function can be used to set the index (location) of a byte code. The index is used in the execute method for each of these branch byte codes.

6.1 Class Diagram

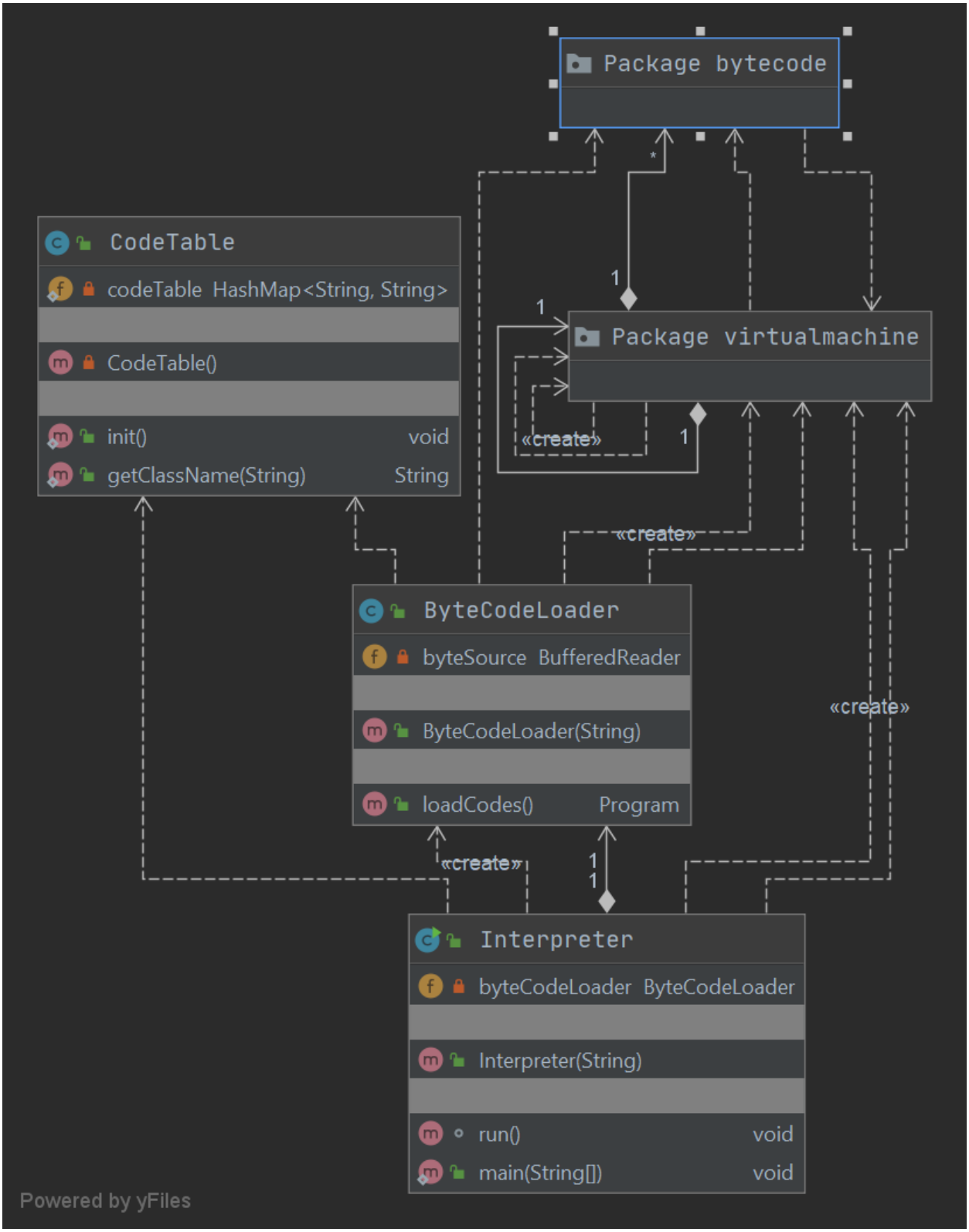
ByteCode Package



Virtual Machine Package



Interpreter Package



7 Project Reflection

For the entire first week of the project I was lost. I did not really understand how to go about building this project or where I should even start. It took a lot of trial and error and I had to watch the video lectures multiple times over. I believe I ended in a pretty good spot but no where near what I wanted to be. I felt like every class implementation is a but generic but fulfils the requirements. I could not get my program to run through the command line which is discouraging but it works perfectly through the IDE. As with the last project, I wish I had more time to work on it like a normal semester, so I do feel that this one was a bit rushed.

8 Project Conclusion/Results

The outputs for both the factorial.x.cod and the fib.x.cod are correct but not complete. I noticed when comparing my output to the professors output (posted in slack) that mine differed slightly from his. I was not sure if this was because my implementation is different (and probably not as good) or whether my bytecodes were not processed properly. For example, when executing factorial.x.cod, the first few lines of the professors output are as follows:

```
GOTO start<<1>>
[]
LABEL start<<1>>
[]
GOTO continue<<3>>
[]
LABEL continue<<3>>
[]
ARGS 0
[] []
CALL Read   Read()
[] []
LABEL Read
[] []
Please enter an integer:
6
```

My output:

GOTO start<<1>>

[]

GOTO continue<<3>>

[]

ARGS 0

[] []

CALL Read

[] []

Please enter an integer:

6

While the ending of the professors output is:

```
ARGS 1
[] [720]
CALL Write   Write(720)
[] [720]
LABEL Write
[] [720]
LOAD 0 dummyFormal
[] [720, 720]
720
WRITE
[] [720, 720]
RETURN
[720]
POP 3
[]
```

And my ending output is:

```
ARGS 1
[] [720]

CALL Write
[] [720]

LOAD 0 dummyFormal
[] [720, 720]

720
WRITE
[] [720, 720]

RETURN
[720]

POP 3
[]

HALT
```

[]

After testing a few different integers, it was clear that the program itself runs fine with the correct answer being outputted every time. My output just differs slightly from the professors. I wish there was more time to debug and figure out where I could improve but overall I believe this project is satisfactory.