CSC 413 Project Documentation

Spring 2019

Carlos Lopez

918559153

413.02

https://github.com/csc413-02-spring2019/csc413-02-carlosmcodes

Table of Contents

1 Introduction

1.1 Project Overview

The main objective of this project was to create a real time compiler, A JVM. One involving a runtime stack to place variables/ functions and literals on as well as using a stack data structure to keep variables these variables and functions within the scope of that function. We were given a programming language named 'X' and our job was to break this down to interpret it into specific bytecodes.

1.2 Technical Overview

In terms of technicality, I found this project very difficult. Not so much the programming but understanding how the JVM works in and out. Diving into the technicality, this project was broken up into a series of classes, a good fraction of those being the bytecodes which came from an absract bytecode class. Each bytecode has its own components and works independently of each other but it does have one abstract bytecode that is universal for all. Followed by this is the class that loades the bytecodes, nothing special here. A class called program was created to resolve jumps in the program when jump functions were found. The runtime stack class was more so the 'meat' of the project. This was implemented using several functions where I intergrated both location of frames hand in hand with the runtime stack. Along with this function several other functions were created to fit the structure of how bytecodes work. Following the runtime stack class came the virtual machine. From how I understood encapsulation and the whole project I took the virtual machine as being the middle man of how the runtime stack and bytecodes 'speak'. This class had some small methods that simply passed over data through the runtime stack. Though specifically, every operation to bytecode had to pass through the virtual machine if the origin was the runtime stack.

1.3 Summary of Work Completed

In summation of what was completed, the bytecodes were all implemented with abstraction and the runtime stack, program, and virtual machine classes were all implemented by yours truly.

2 Development Environment

Version of Java: 8.0_171

IDE: Intellij IDEA 2018.3

3 How to Build/Import your Project

Assuming you've got VCS installed and integrated with git….

- Open Intellij
- close all projects
- click 'check out from Version Control'
- click 'Git'
- copy URL

- test to verify
- Click clone
- push ok till project opens

4 How to Run your Project

Next to the 'Play' button theres a configuration tab, this is where you configure the IDE to run from a starting point.

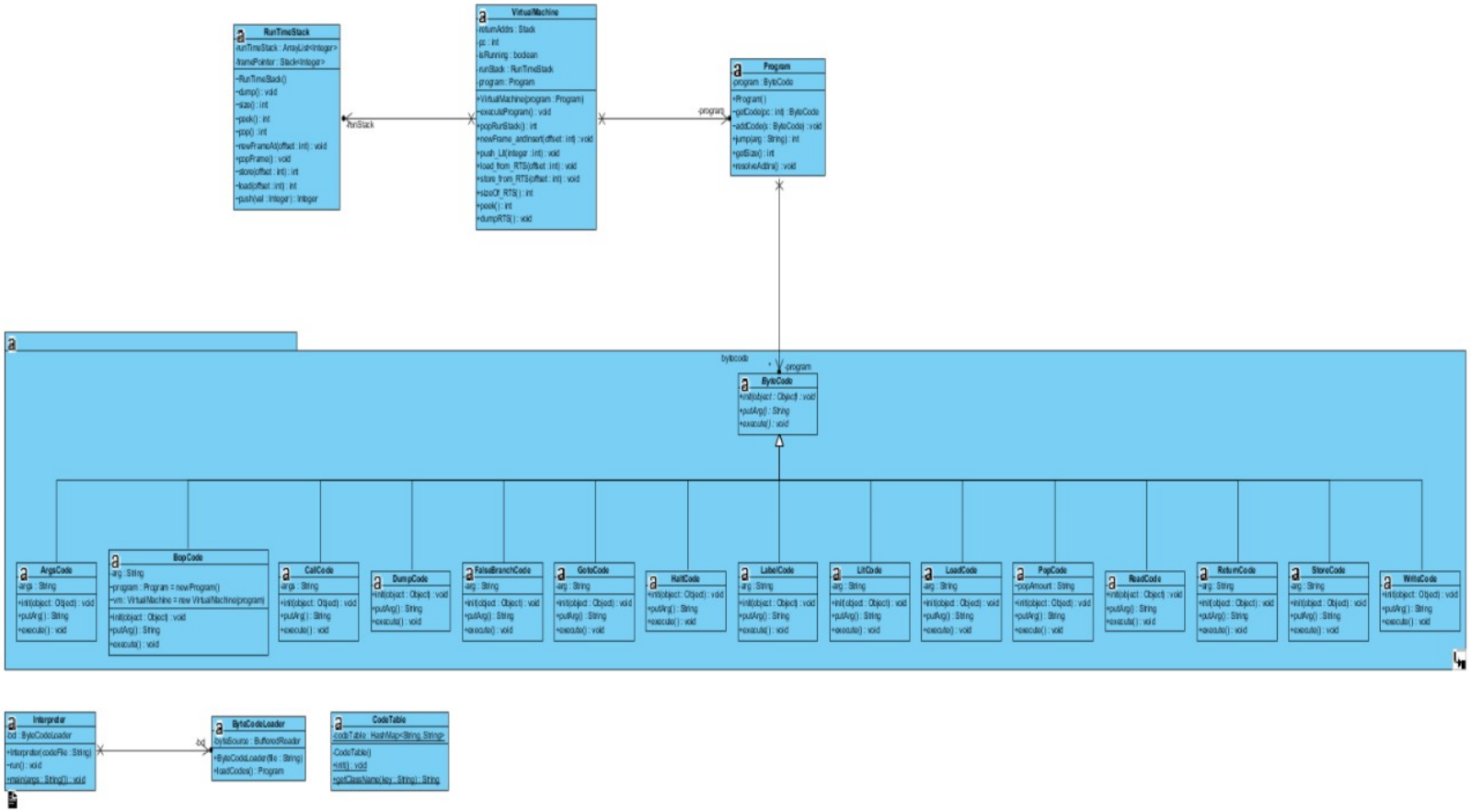
- Edit configuration
- + (top left)
- applications
- Main class (…)
- pick Interpreter.
- Ok
- Press 'Play' button

5 Assumption Made

My initial assumption was that I would struggle with encapsulation and abstraction. I knew what encapsulation meant and what it was but I had never seen it be used in a way the way the virtual machine passed the runtime stack arguments to the bytecodes. I thought that was a big step in terms of my programming toolbox. Abstraction I had done before, but just was a little hesitant on it. Though to my knowledge I implemented it correctly.

6 Implementation Discussion

6.1 Class Diagram


RunTimeStack
runTimeStack : ArrayList<Integer>
framePointer : Stack<Integer>
+RunTimeStack()
+dump() : void
+size() : int
+peek() : int
+pop() : int
-newFrameAt(offset : int) : void
+popFrame() : void
-store(offset : int) : int
-load(offset : int) : int
+push(val : Integer) : Integer
rtnStack
VirtualMachine
returnAddrs : Stack
pc : int
isRunning : boolean
runStack : RunTimeStack
program: Program
+VirtualMachine(program : Program)
+executeProgram() : void
+popRunStack() : int
+newFrame_andInsert(offset: int) : void
+push_LI(Integer :int) : void
+load_from_RTS(offset :int) : void
+store_from_RTS(offset : int) : void
+sizeOf_RTS() : int
+peek() : int
+dumpRTS() : void
program
Program
program : ByteCode
+Program()
+getCode(pc : int) :ByteCode
+addCode(b : ByteCode) : void
+jump(arg : String) : int
+getSize() : int
+resolveAddrs() : void
bytecode
*
program
ByteCode
+init(object : Object) : void
+putArg() : String
+execute() : void
ArgsCode
-arg : String
+init(object: Object) : void
+putArg() : String
+execute() : void
BopCode
-arg : String
-program : Program = newProgram()
-vm : VirtualMachine = new VirtualMachine(program)
+init(object : Object : void
+putArg() : String
+execute() : void
CallCode
-arg : String
+init(object: Object) : void
+putArg() : String
+execute() : void
DumpCode
+init(object : Object) : void
+putArg() : String
+execute() : void
FalseBranchCode
-arg : String
+init(object : Object) : void
+putArg() : String
+execute() : void
GotoCode
-arg : String
+init(object : Object) : void
+putArg() : String
+execute() : void
HaltCode
+init(object: Object) : void
+putArg() : String
+execute() : void
LabelCode
-arg : String
+init(object : Object) : void
+putArg() : String
+execute() : void
LitCode
-arg : String
+init(object : Object) : void
+putArg() : String
+execute() : void
LoadCode
-arg : String
+init(object : Object) : void
+putArg() : String
+execute() : void
PopCode
-popAmount : String
+init(object: Object) : void
+putArg() : String
+execute() : void
ReadCode
+init(object : Object) : void
+putArg() : String
+execute() : void
ReturnCode
-arg : String
+init(object : Object) : void
+putArg() : String
+execute() : void
StoreCode
-arg : String
+init(object : Object) : void
+putArg() : String
+execute() : void
WriteCode
+init(object: Object) : void
+putArg() : String
+execute() : void
Interpreter
bd : ByteCodeLoader
+Interpreter(codeFile : String)
+run() : void
+main(args : String[]) : void
bd
ByteCodeLoader
byteSource : BufferedReader
+ByteCodeLoader(file : String)
+loadCodes() : Program
CodeTable
codeTable : HashMap<String,String>
-CodeTable()
+init() : void
+getClassName(key : String) : String

7 Project Reflection

To reflect on what was done and what was learned. What was done was not my best work, my understanding of a JVM was enough to pass 340. Implementing that brought me a gang of fears and concerns. Though I will say, this project did help me understand how it works better than I had previously known. But I can truthfully say I do not know it enough to implement it. I did my best to understand but I did struggle with the concept.

In terms of programming skills, I thought encapsulation was learned, understood and implemented fully. Abstraction, same thing. I did also think using the 'forName()' method and the 'getDeclaredConstructor().newInstance()' method was a tool well attained. I always wondered how we formed classes from string literals and this showed me how.

8 Project Conclusion/Results

To conclude, working with 16+ classes was over whelming and it was by far the biggest project I have ever worked on but even though I struggled and did not complete it fully. I did feel this was beneficial to me and I did learn a lot even with the restriction of time given.