Sandesh Basnet

https://github.com/csc413-03-sp18/csc413-p3-SandeshOnRails.git

Project Introduction and Overview

The goal of this project was to build an Interpreter for a language. The project skeleton was provided with some of the files already implemented for us while we had to work on to create an abstract class Bytecode, that is extended and implemented by the different types of Bytecode, implement Program, Virtual Machine, and the RunTimeStack class. The BytecodeLoader class reads the bytecode from a file and loads it into the program and also resolves the addresses with symbols. The Program class uses an Array List of Bytecodes that is loaded from the BytecodeLoader class. The RunTimeStack Class uses frame stack and runtime stack to push a new frame whenever a function Is invoked, and removing it after it is processed. The virtual machine executes the Program by creating an instance of the RunTimeStack and returnAddrs, and executes every bytecode until the program Is running.

Instructions to Compile as Jar and Execute

Java –jar Interpreter.jar

Development Environment

NetBeans IDE 8.2 with JDK version 8.

Assumptions

Upon reading the instructions for creating classes for abstract type bytecode, I assumed extending the abstract class and implementing them would be the most challenging part. It also was a hard enough task how to model the bytecode classes while maintaining a good object oriented design.

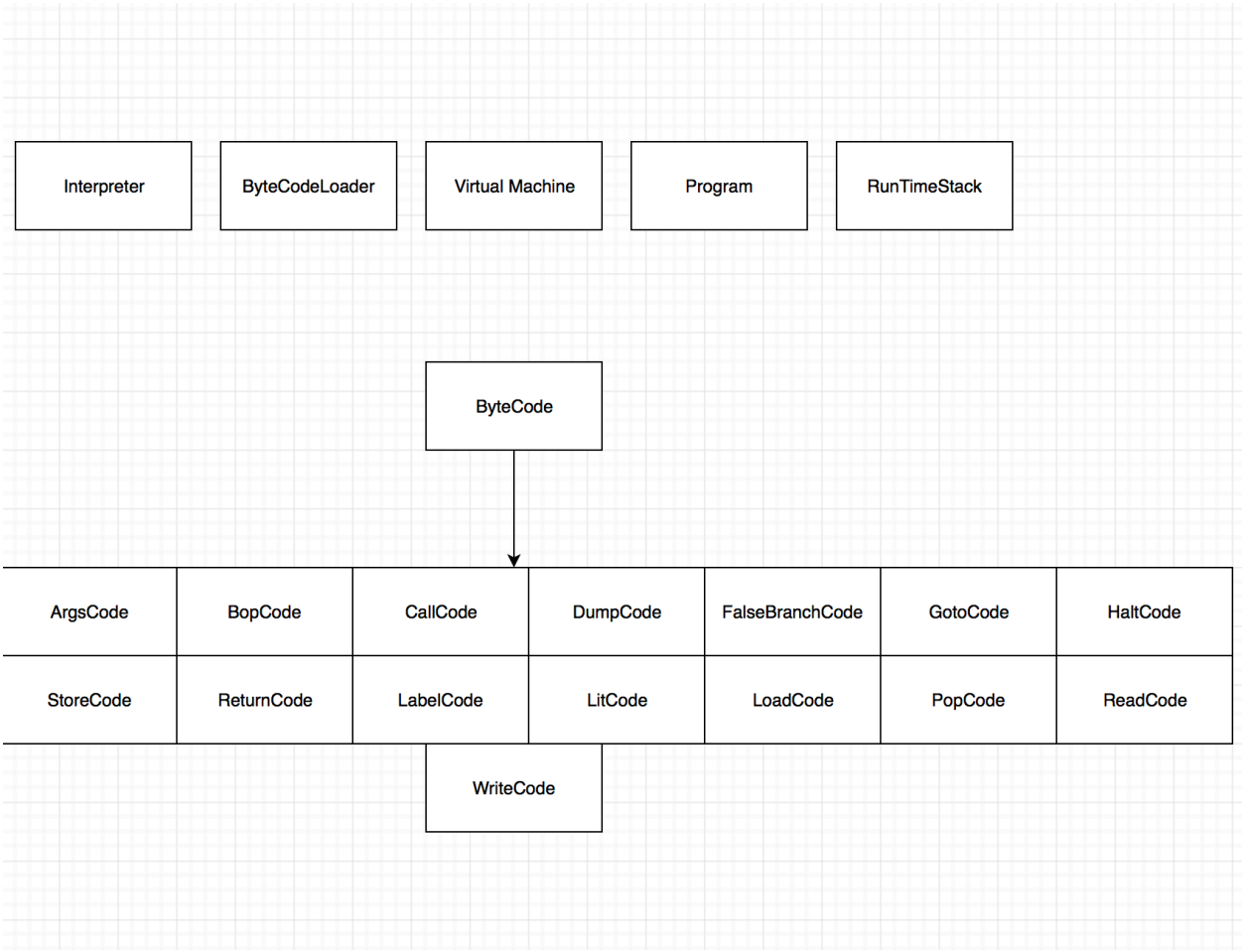
Implementation

Abstract Class ByteCode

- Class Popcode: Pops the top of the runTimeStack.
- Class FalseBranchCode: Pops the top of the stack and branches the label if 0, otherwise proceed execution.
- Class GoToCode: Goes to the given label.
- Class StoreCode: Pops the stack and loads it onto the offset.
- Class LoadCode: Pushes offset values in the frame.

- Class LitCode: Loads the literal value ‘n’ adds it in order to initialize it.
- Class ArgsCode: Initializes the instance of the frame stack.
- Class CallCode: Transfers control to the indicated function.
- Class ReturnCode: Return from the current la given a function name as parameter.
- Class BopCode : Pops the top 2 values of the stack and does the necessary operations.
- Class ReadCode: Reads an integer, prompts the user for input and adds the value to the stack
- WriteCode: Writes the value on top of the stack.

Below is the class hierarchy



Results and Conclusion:

I was successfully able to interpret the bytecode files factorial.x.cod and fib.x.cod. The most challenging part for me was to figure out how to model the instructions into code structures and at which point to approach the project. At first I was having trouble compiling the Interpreter, and I figured out it was due to an error in the loadCodes() in the ByteLoader Class. This project helped me see processing of code files on a system level.