CSC 413 Project Documentation

Fall 2018

Interpreter

Name: Kilan Rai

Student ID: 916002781

Class.Section: CSC413-01

GitHub Repository Link

https://github.com/csc413-01-fa18/csc413-p2-kilanrai.git

Table of Contents

1 Introduction

1.1 Project Overview: This project is about implementing “the interpreter” program for the mock language X. We can think of the mock language X as a simplified version of Java. The interpreter is responsible for processing byte codes that are created from source code files with the extension x. The interpreter and the Virtual Machine will work together to run a program written in the Language X. We will program to calculate the recursive call, i.e. our program or this program should be able to calculate the Fibonacci number and factorial numbers. So, the two programs are a recursive version of computing the nth Fibonacci number and recursively finding the factorial of the number. And these files have the extension x.cod.

1.2 Technical Overview: First, the ByteCodeLoader class loads the bytecodes from the source code file into a data-structure that stores the entire program. Then the Program class will store all the bytecodes read from the source file. And the Virtual Machine will execute them one by one in the order it gets from the Program class.

1.3 Summary of Work Completed: In this project, I was asked to implement four classes, namely- BytecodeLoader, Program, RuntimeStack and Virtual Machine. In addition, in the package bytecode, I have created ByteCode classes as follows: HaltCode, PopCode, FalseBranchCode, GotoCode, StoreCode, LoadCode, LitCode, ArgsCode, CallCode, ReturnCode, BopCode, ReadCode, WriteCode, LabelCode, DumpCode, and JumpCode.

2 Development Environment: I used IntelliJ IDEA 2018.2.3x64

3 How to Build/Import your Project: First close any running project on IDE IntiliJ. Once accepted the project iLearn, go to github repo, then clone or download zip. On IDE click on import project from existing source, then on the top Project name : csc413-p2-kilanrai-master; Project Location: C:Users\Kilan\Desktop\csc413-p2-kilanrai-master; then choose format: idea(directory based); click next, next, finish.

4 How to Run your Project: On IntelliJ IDE, on top left corner, we can see configuration menu, click on Edit Configurations and put the files names that we want to run : factorial.x.cod and fib.x.cod. Then we can specify any file name to run.

5 Assumption Made: The first thing, the ByteCodeLaoder Classs loads bytecodes from the source file into a data-structure that stores all bytecodes in an ArrayList contained inside of a Program object. But adding and getting any bytecodes has to go through the Program Class. And the Program Class will store all the bytecodes read from the source file into an ArrayList which has a designated type of ByteCode. The RunTimeStack class records and processes the stack of active frames. In addition, this class contains two data structures used to help the VirtualMachine execute the program. Finally, the Virtual Machine executes the given program and is the controller of this program. All operations need to go through this VirtualMachine class.

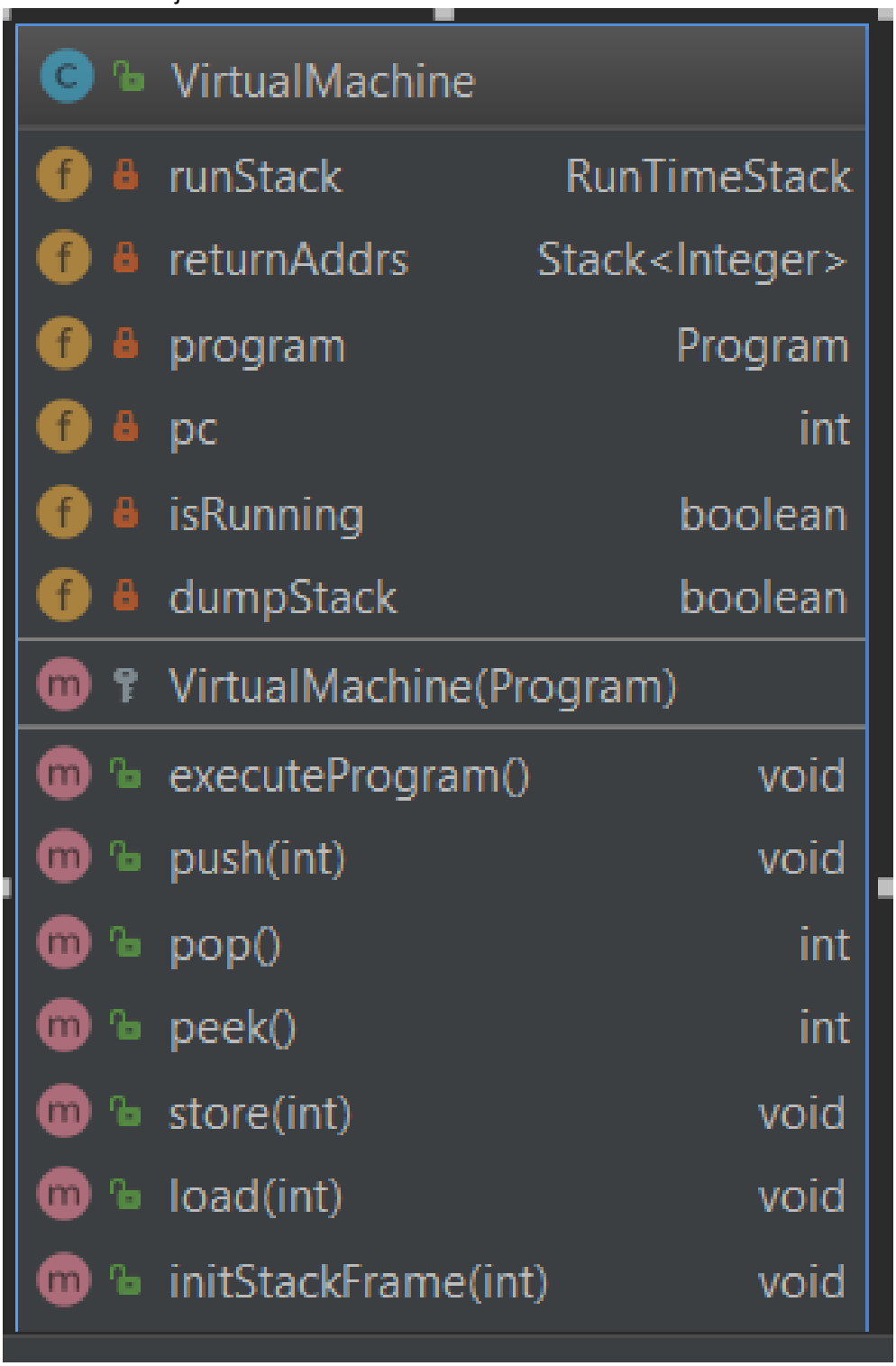
6 Implementation Discussion:

6.1 Class Diagram:

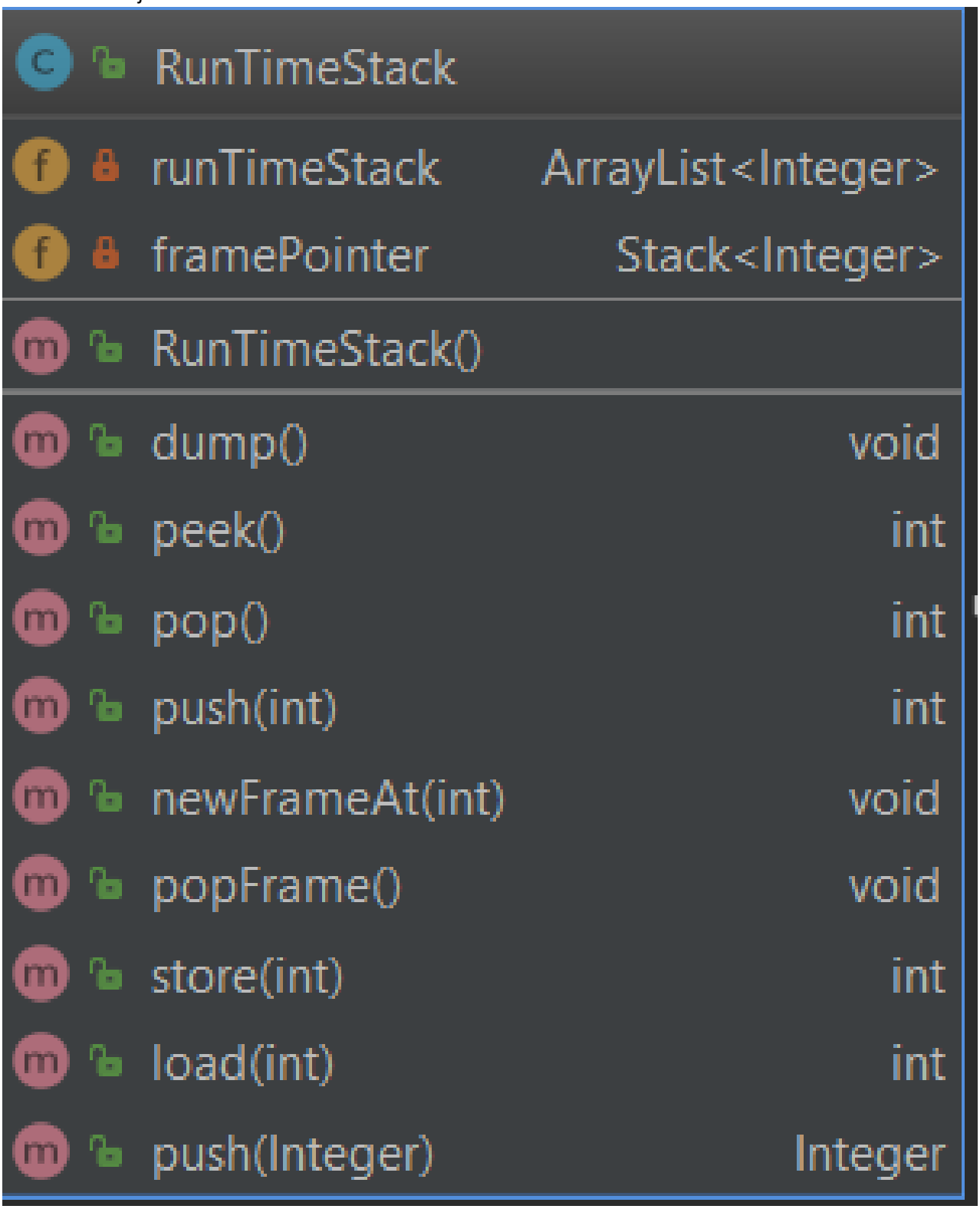
Forder Name : csc413-p1-kilanrai

Package Name: interpreter:

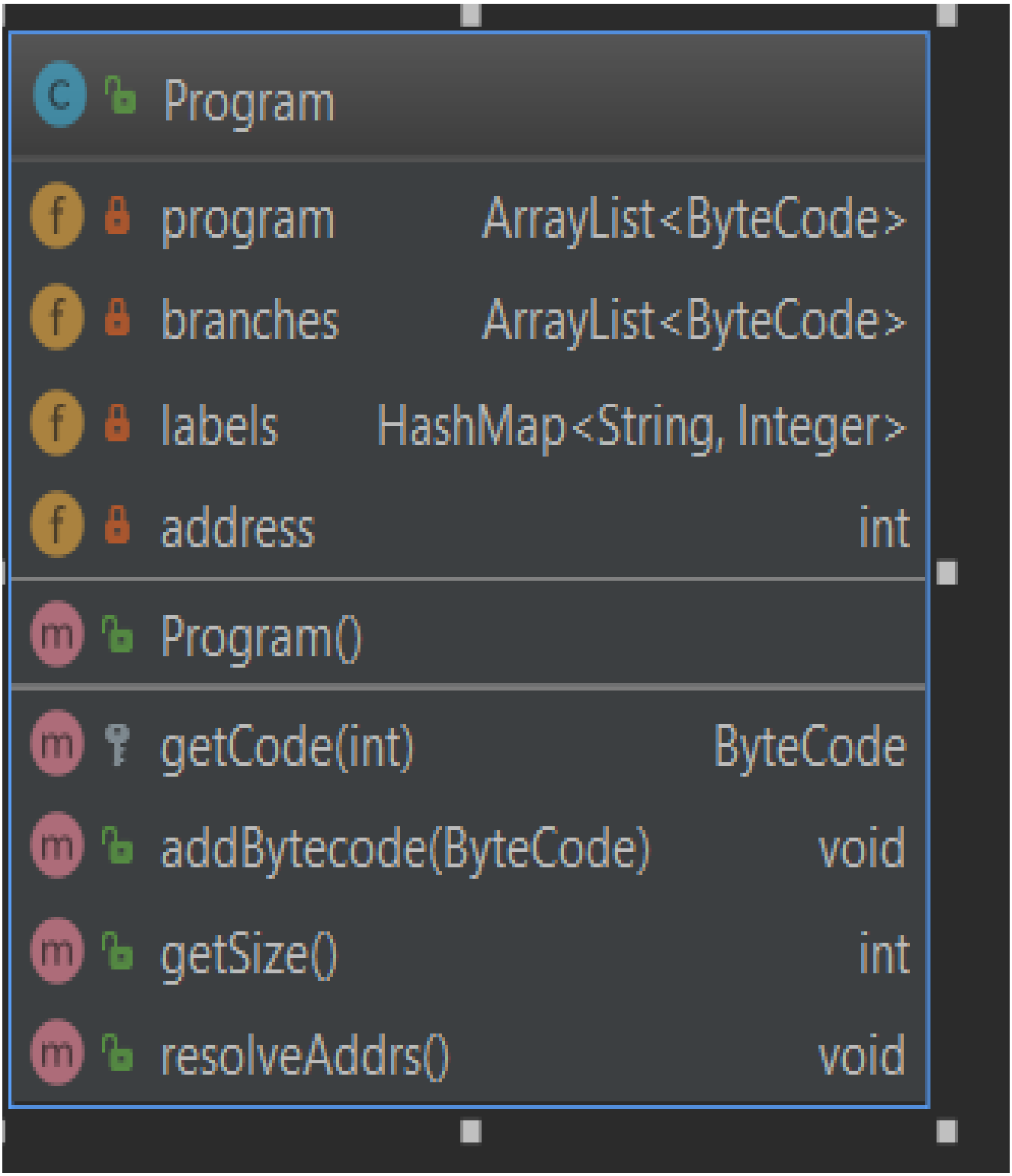
1. VirtualMachine.java



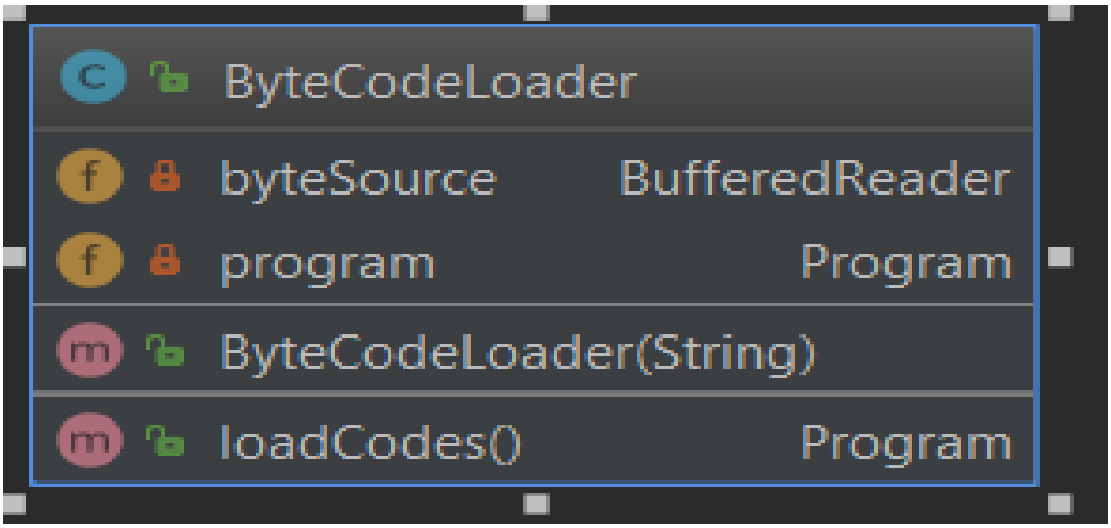
2. RunTimeStack.java



3. Program.java

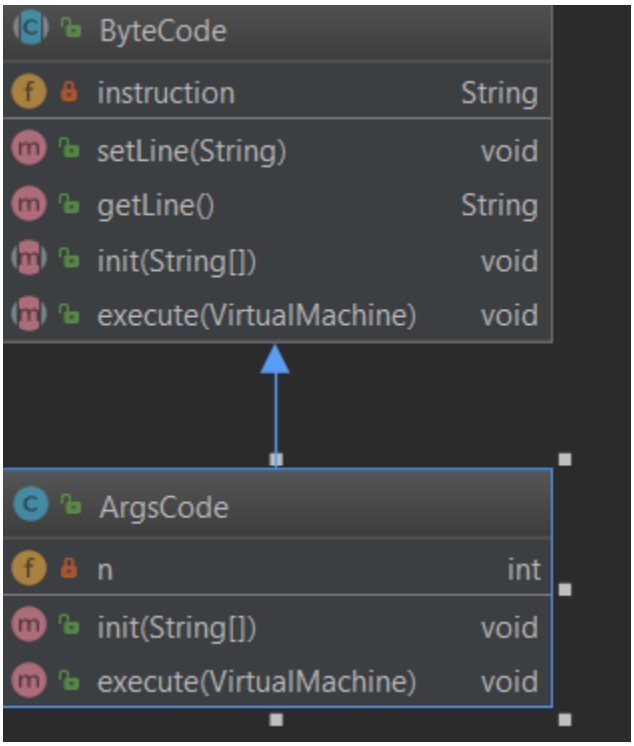


4. ByteCodeLoader.java

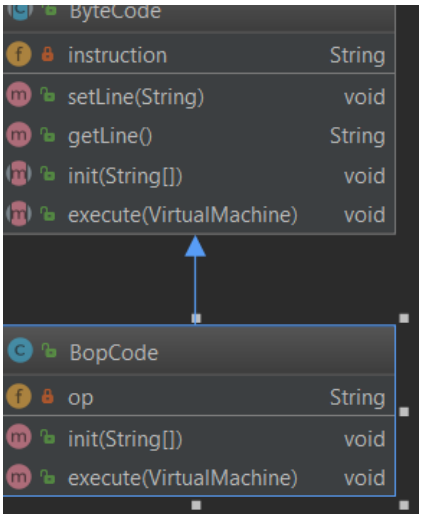


bytecode:

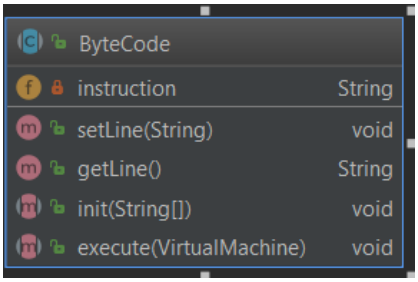
(i)ArgsCode.java



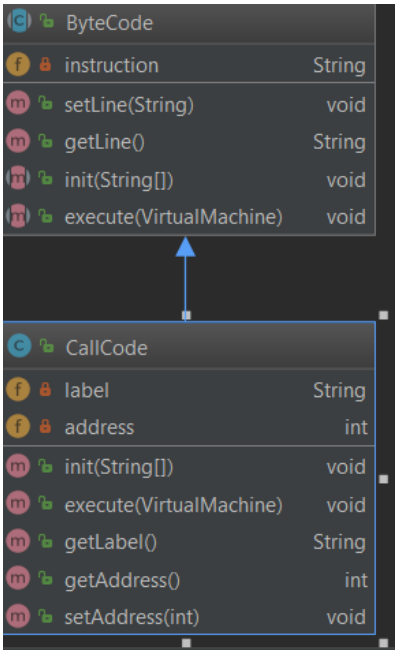
(ii)BopCode.java



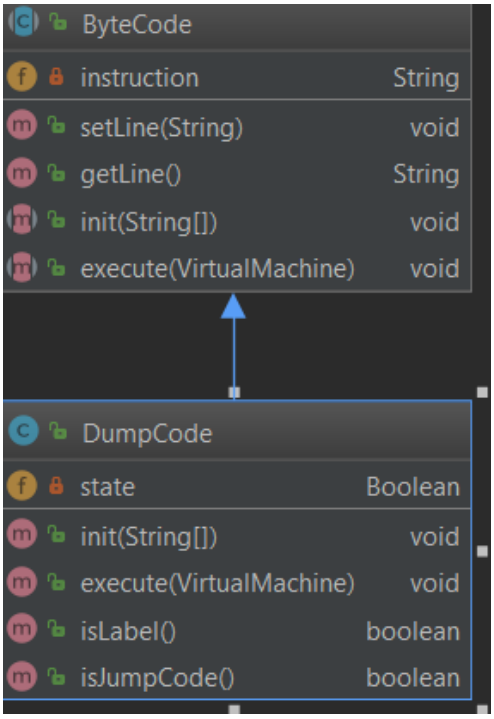
(iii)ByteCode.java



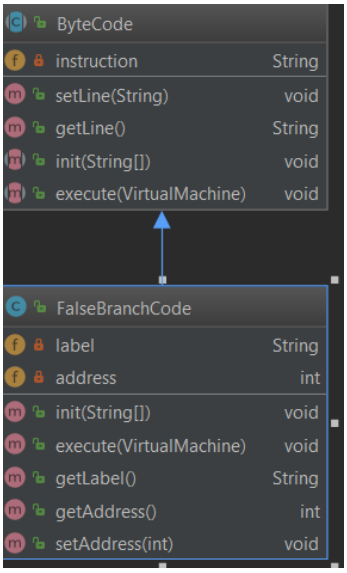
(iv)CallCode.java



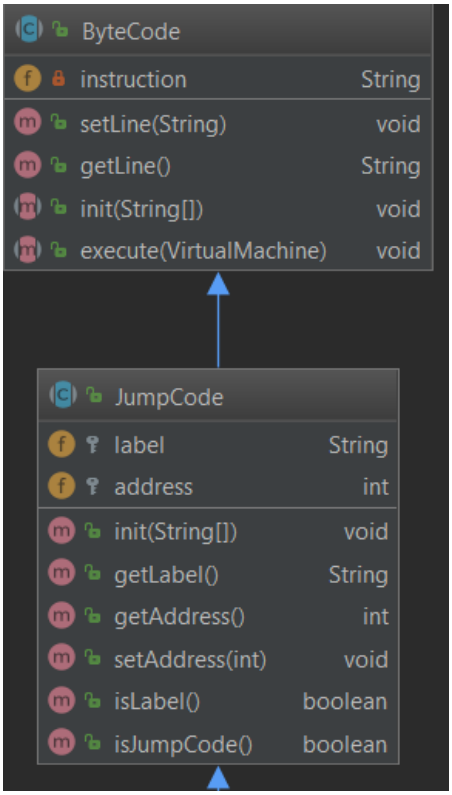
(v) DumpCode.java



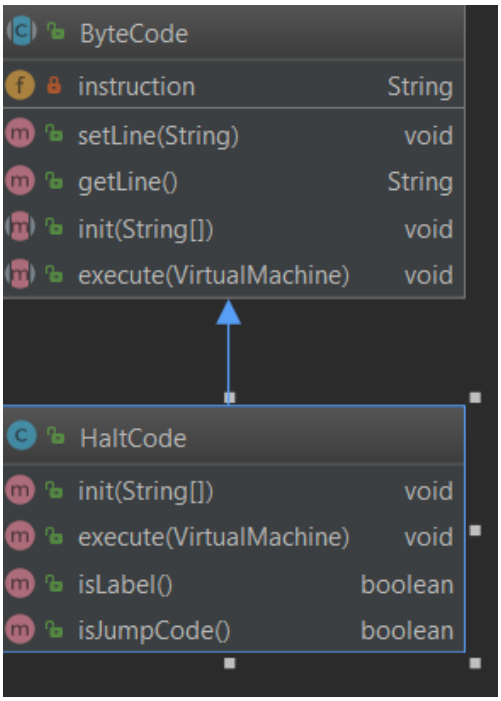
(vi) FalseBranchCode.java



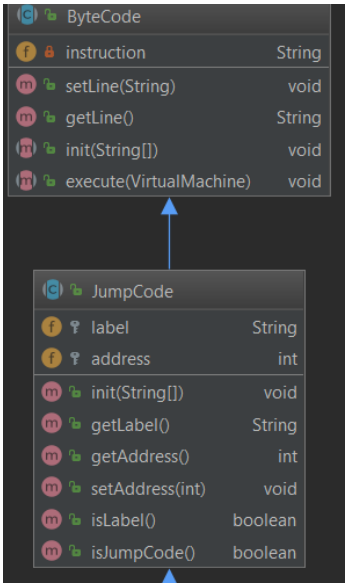
(vii) JumpCode.java



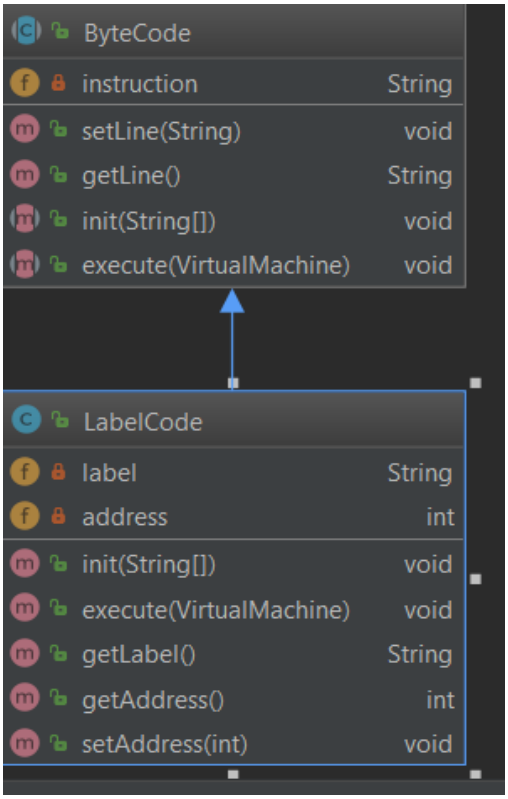
(viii) HaltCode.java



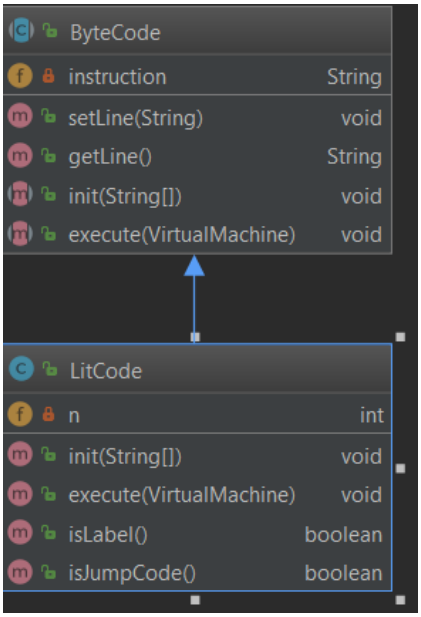
(ix) GotoCode.java



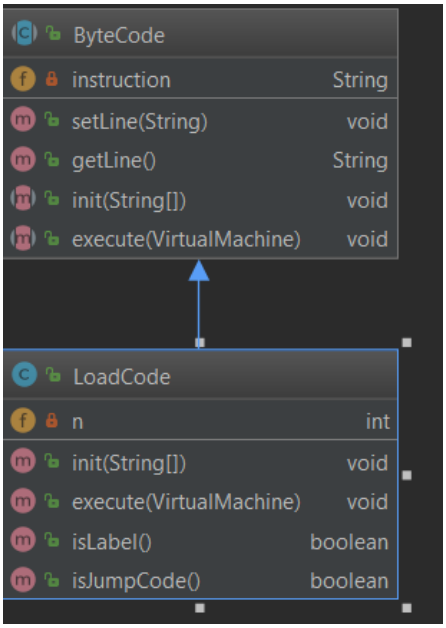
(x)LabelCode.java



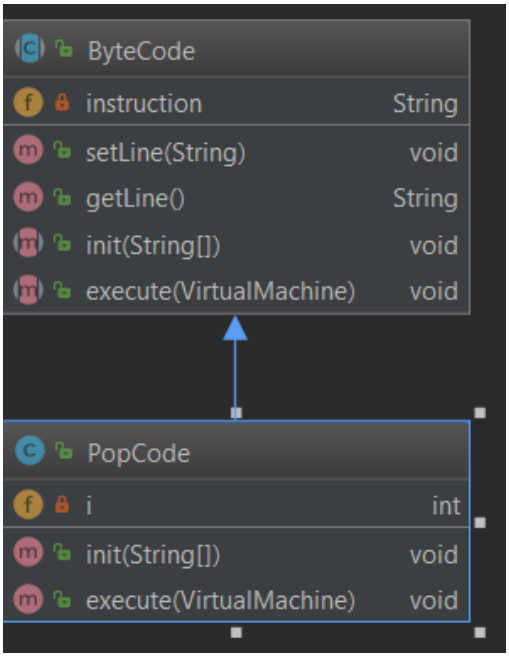
(xi)LitCode.java



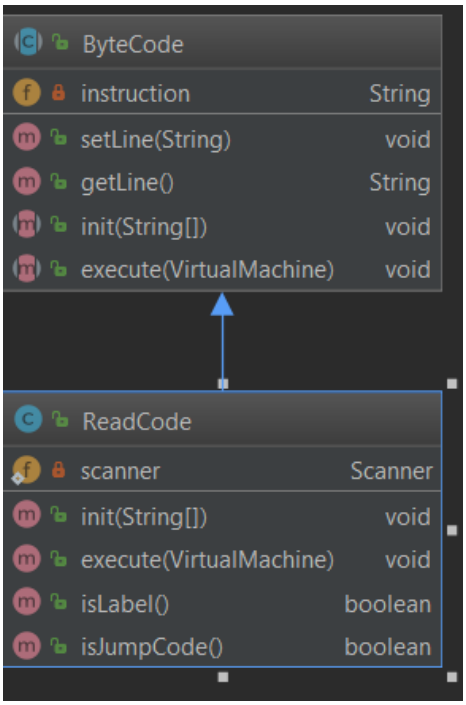
(xii) LoadCode.java



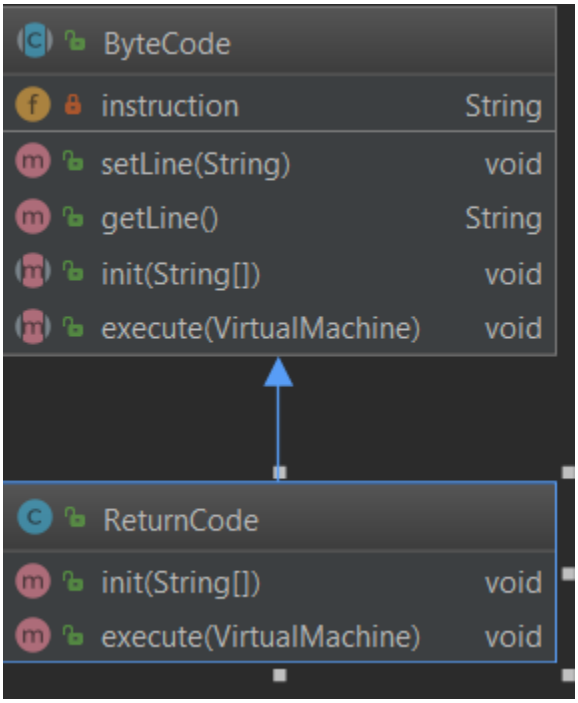
(xiii) PopCode.java



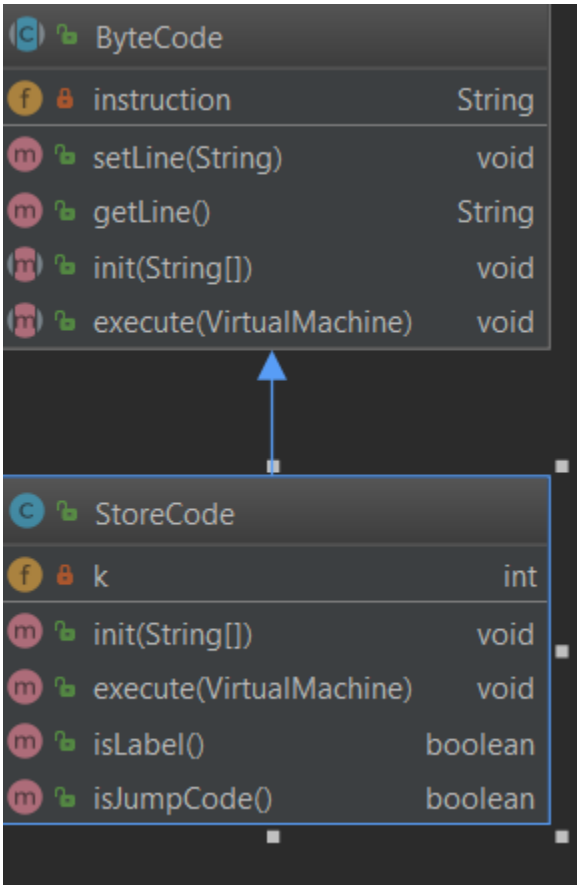
(xiv) ReadCode.java



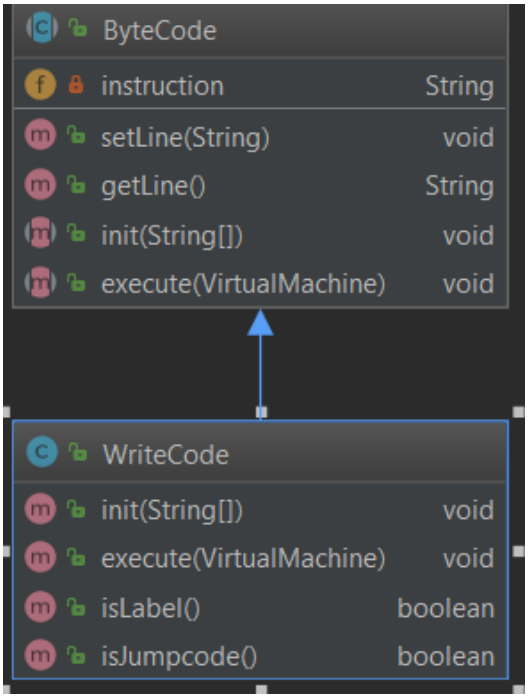
(xv) ReturnCode.java



(xvi)StoreCode.java



(xvii) WriteCode.java



7 Project Reflection: I was totally excited to do this project because I got to a chance how to work JMV in bytecode level and how the computer works in low level. Initially, I was lost what to do and how to tackle the problem. But when I started implementing the program, I got the idea how instruction works in Virtual machine. It was a little bit challenging for me to solve the task or problems. I worked with Fatma Khan since she wants to work together sometimes with me.

8 Project Conclusion/Results: Run with fib.x.cod:

For input 3:

3

[5,0] [3]

RETURN

[5,0,3]

ARGS 1

[5,0] [3]

CALL fib<<2>>

[5,0] [3]

LABEL fib<<2>>

[5,0] [3]

LOAD 0 n

[5,0] [3,3]

LIT 1

[5,0] [3,3,1]

BOP <=

[5,0] [3,0]

FALSEBRANCH else<<4>>

[5,0] [3]

LABEL else<<4>>

[5,0] [3]

LOAD 0 n

[5,0] [3,3]

LIT 2

[5,0] [3,3,2]

BOP ==

[5,0] [3,0]

FALSEBRANCH else<<6>>

[5,0] [3]

LABEL else<<6>>

[5,0] [3]

LOAD 0 n

[5,0] [3,3]

LIT 2

[5,0] [3,3,2]

BOP -

[5,0] [3,1]

ARGS 1

[5,0] [3] [1]

CALL fib<<2>>

[5,0] [3] [1]

LABEL fib<<2>>

[5,0] [3] [1]

LOAD 0 n

[5,0] [3] [1,1]

LIT 1

[5,0] [3] [1,1,1]

BOP <=

[5,0] [3] [1,1]

FALSEBRANCH else<<4>>

[5,0] [3] [1]

LIT 1

[5,0] [3] [1,1]

RETURN fib<<2>>

[5,0] [3,1]

LOAD 0 n

[5,0] [3,1,3]

LIT 1

[5,0] [3,1,3,1]

BOP -

[5,0] [3,1,2]

ARGS 1

[5,0] [3,1] [2]

CALL fib<<2>>

[5,0] [3,1] [2]

LABEL fib<<2>>

[5,0] [3,1] [2]

LOAD 0 n

[5,0] [3,1] [2,2]

LIT 1

[5,0] [3,1] [2,2,1]

BOP <=

[5,0] [3,1] [2,0]

FALSEBRANCH else<<4>>

[5,0] [3,1] [2]

LABEL else<<4>>

[5,0] [3,1] [2]

LOAD 0 n

[5,0] [3,1] [2,2]

LIT 2

[5,0] [3,1] [2,2,2]

BOP ==

[5,0] [3,1] [2,1]

FALSEBRANCH else<<6>>

[5,0] [3,1] [2]

LIT 1

[5,0] [3,1] [2,1]

RETURN fib<<2>>

[5,0] [3,1,1]

BOP +

[5,0] [3,2]

RETURN fib<<2>>

[5,0,2]

ARGS 1

[5,0] [2]

CALL Write

[5,0] [2]

LABEL Write

[5,0] [2]

LOAD 0 dummyFormal

[5,0] [2,2]

WRITE

2

[5,0] [2,2]

RETURN

[5,0,2]

STORE 1 k

[5,2]

LIT 0 x

[5,2,0]

LIT 7

[5,2,0,7]

STORE 2 x

[5,2,7]

LIT 8

[5,2,7,8]

STORE 2 x

[5,2,8]

POP 1

[5,2]

POP 2

[]

HALT

[]

Process finished with exit code 0

Run with factorial.x.cod

For 3 input: result is 6

It seems like it is working .