CSC 413 Project Documentation

Fall 2018

Sunminder Sandhu

916034306

CSC 413.01

https://github.com/csc413-01-fa18/csc413-p1-ssandhu2

Table of Contents

1) Introduction:

a) **Project Overview:** The purpose of this project is to implement an interpreter for the mock language X. We design program that translates file with the extension x into bytecode, stores it in an Arraylist and runs via the Virtual Machine.

b) **Technical Overview:** In this project we were supposed to take a file with bytecodes and process it. We use bytecodeloader class to load bytecodes into an arraylist use program class to process it. Runtimestack maintains activation frames. The VM and interpreter that we implement work together to run a program that has been written in language X.

c) **Summary of Work Completed:** In total we were given 6 classes and had to make 15 more subclasses to get the interpreter running. ByteCodeLoader reads the .x.cod files and translates in into bytecodes. We store bytecodes in the instance of program. In program class we resolve the address. In Virtual Machine implemented more methods for dump, to connect with our classes and maintain encapsulation. The runtimestack class had methods like pop peek etc. The subclasses had three methods each execute init and toPrint.

2) Development environment:

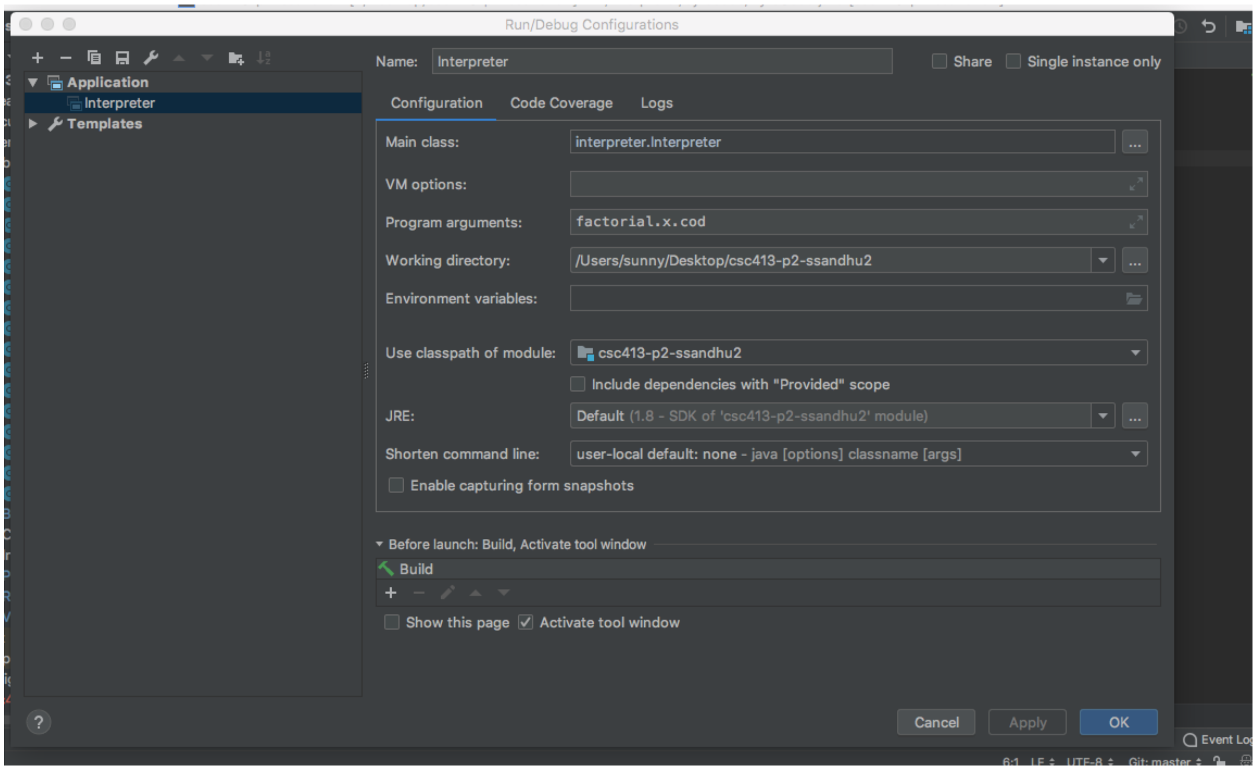
a) **Version of Java used:** 1.8.0_181

b) **IDE used:** IntelliJ IDEA

3) How to build/import the project:

- The first step is to clone the project to the desired location using terminal with the git clone "link" command and the link can be copied from the GitHub repository.
- Next, I opened up intelliJ which is the IDE I used and clicked import project -> import project from existing sources and hit next -> the name the project and hit next -> select all the files, lib and jdk hit next-> Finish

4) How to run your project: We can run the project using run which is the green button shaped like a triangle. But before running you need to add the x file into the program arguments which can be done by clicking on interpreter next to the green button and clicking edit configurations. Next put the factorial.x.cod or any .x.cod file you want to test for in the program aruguemnts hit apply and ok. Next hit the run button to run the file.

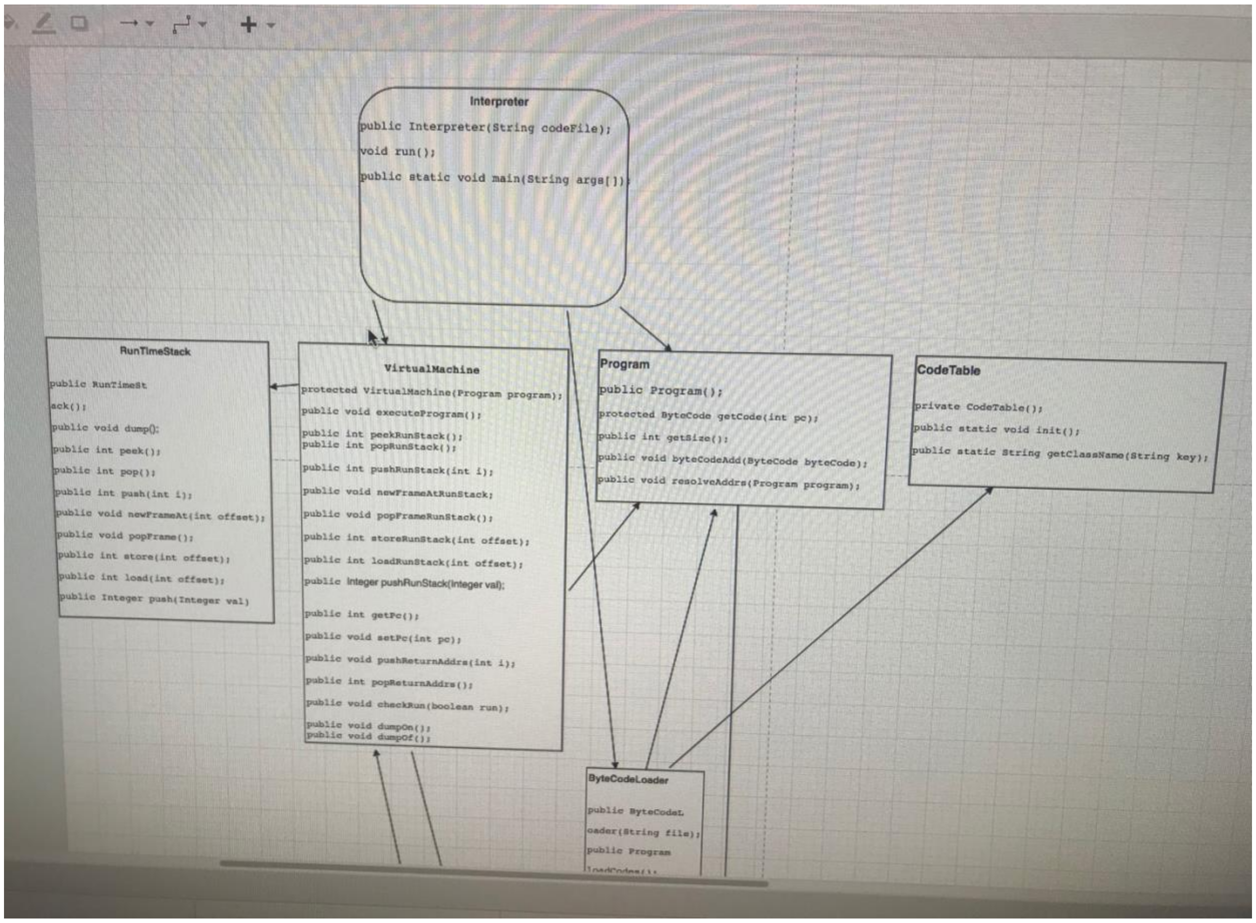


Name: Interpreter
Share
Single instance only
Main class: interpreter.Interpreter
VM options:
Program arguments: factorial.x.cod
Working directory: /Users/sunny/Desktop/csc413-p2-ssandhu2
Environment variables:
Use classpath of module: csc413-p2-ssandhu2
Include dependencies with "Provided" scope
JRE: Default (1.8 - SDK of 'csc413-p2-ssandhu2' module)
Shorten command line: user-local default: none - java [options] classname [args]
Enable capturing form snapshots
Before launch: Build, Activate tool window
Build
Show this page [✓ Activate tool window]

5) Assumptions made when designing and implementing your project: The first assumption I made was that to resolve the symbolic address we wouldn't need to use instanceof keywor. I didn't know what the keyword instanceof meant when I started then I checked slack and saw the professor told us we could use the keyword instanceof. I also assumed that we wouldn't need to make a hashmap in the program class to resolve address. In the subclasses at first, I did all of them without a toPrint method which prints and tried to do the printing in the execute method. But I felt making a new method to print would be more efficient. Before reading the pdf I assumed the VM class was almost done and only had to fill the while loop in the method but then I read the pdf which said VM will have many methods. I also assumed the user would provide valid input.

6) Implementation Discussion:

Hierarchy Diagram: I made my diagram on draw.io and when imported it as jpeg the diagram came out blurry. So I took pictures with my phone and also put the actual diagram(blurry one) after the pictures taken on my iphone.



Interpreter

```
public Interpreter(String codeFile);
void run();
public static void main(String args[])
```

RunTimeStack

```
public RunTimeSt
ack();
public void dump();
public int peek();
public int pop();
public int push(int i);
public void newFrameAt(int offset);
public void popFrame();
public int store(int offset);
public int load(int offset);
public Integer push(Integer val)
```

VirtualMachine

```
protected VirtualMachine(Program program);
public void executeProgram();
public int peekRunStack();
public int popRunStack();
public int pushRunStack(int i);
public void newFrameAtRunStack;
public void popFrameRunStack();
public int storeRunStack(int offset);
public int loadRunStack(int offset);
public Integer pushRunStack(Integer val);

public int getPc();
public void setPc(int pc);
public void pushReturnAddrs(int i);
public int popReturnAddrs();
public void checkRun(boolean run);
public void dumpOn();
public void dumpOf();
```

Program

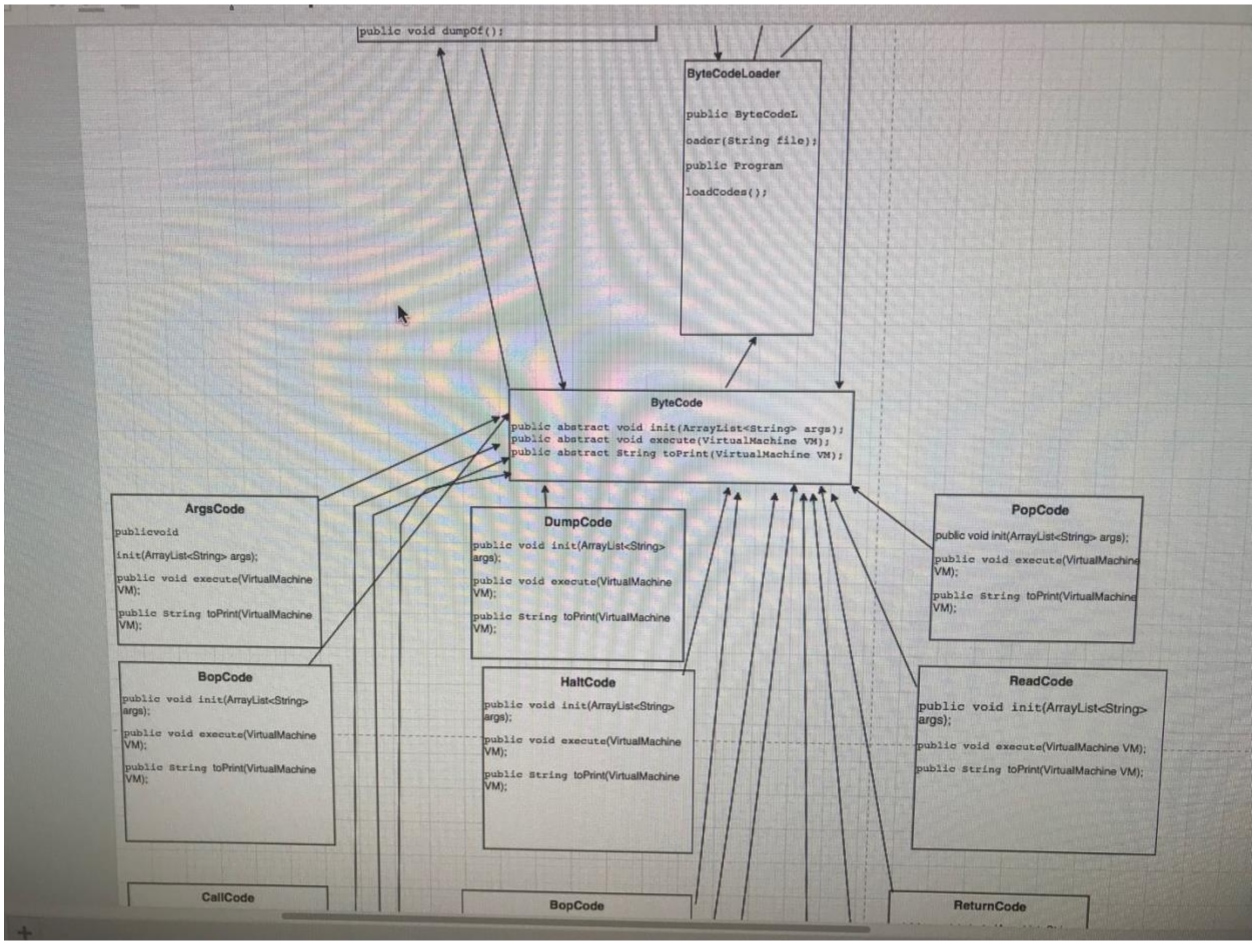
```
public Program();
protected ByteCode getCode(int pc);
public int getSize();
public void byteCodeAdd(ByteCode byteCode);
public void resolveAddrs(Program program);
```

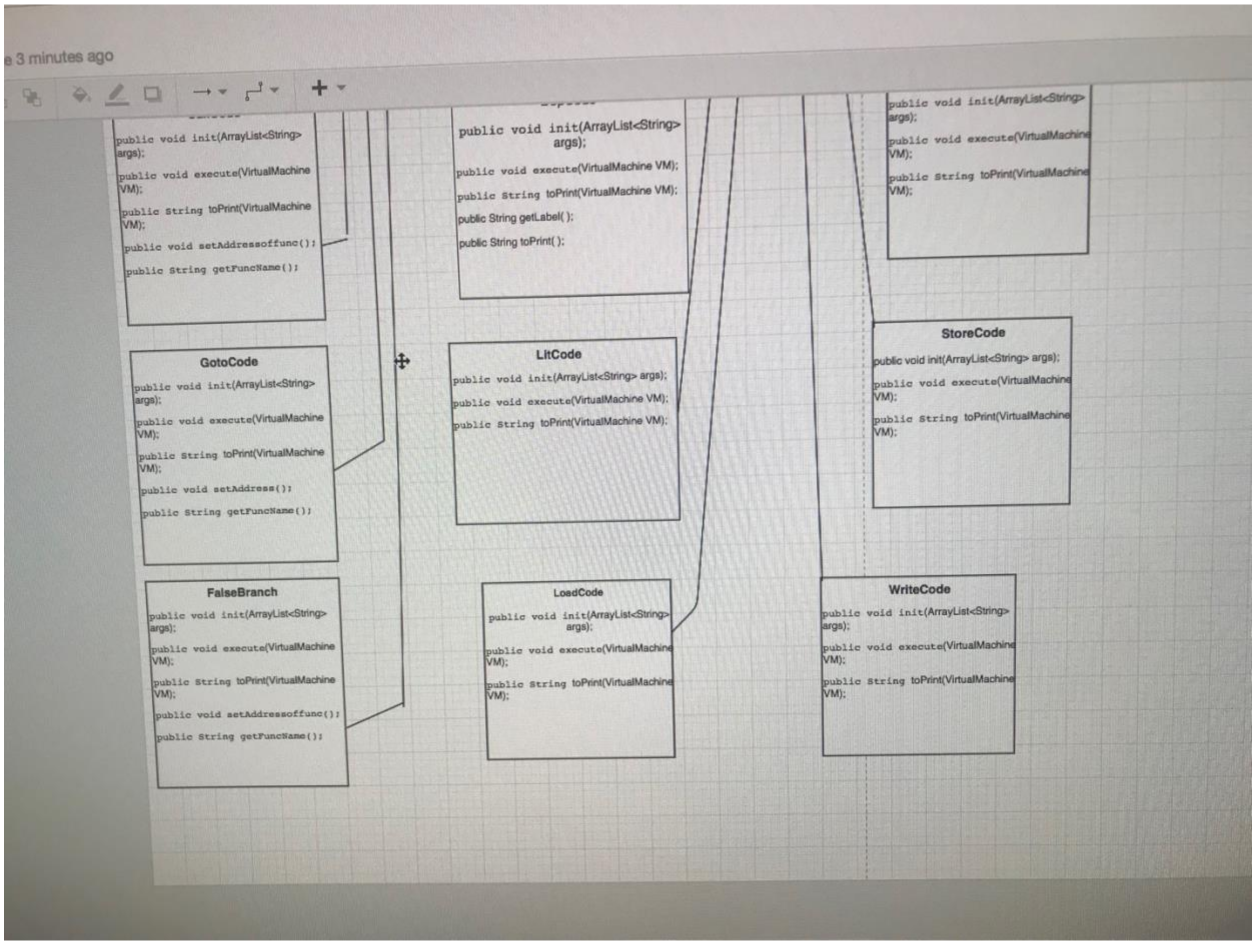
CodeTable

```
private CodeTable();
public static void init();
public static String getClassName(String key);
```

ByteCodeLoader

```
public ByteCodeL
oader(String file);
public Program
loadCodes();
```


public void dumpOf();
ByteCodeLoader
public ByteCodeLoader(String file);
public Program loadCodes();
ByteCode
public abstract void init(ArrayList<String> args);
public abstract void execute(VirtualMachine VM);
public abstract String toPrint(VirtualMachine VM);
ArgsCode
publicvoid init(ArrayList<String> args);
public void execute(VirtualMachine VM);
public String toPrint(VirtualMachine VM);
DumpCode
public void init(ArrayList<String> args);
public void execute(VirtualMachine VM);
public String toPrint(VirtualMachine VM);
PopCode
public void init(ArrayList<String> args);
public void execute(VirtualMachine VM);
public String toPrint(VirtualMachine VM);
BopCode
public void init(ArrayList<String> args);
public void execute(VirtualMachine VM);
public String toPrint(VirtualMachine VM);
HaltCode
public void init(ArrayList<String> args);
public void execute(VirtualMachine VM);
public String toPrint(VirtualMachine VM);
ReadCode
public void init(ArrayList<String> args);
public void execute(VirtualMachine VM);
public String toPrint(VirtualMachine VM);
CallCode
BopCode
ReturnCode



```
public void init(ArrayList<String> args);
public void execute(VirtualMachine VM);
public String toPrint(VirtualMachine VM);
public void setAddressoffunc();
public String getFuncName();
```

```
public void init(ArrayList<String> args);
public void execute(VirtualMachine VM);
public String toPrint(VirtualMachine VM);
public String getLabel( );
public String toPrint( );
```

```
public void init(ArrayList<String> args);
public void execute(VirtualMachine VM);
public String toPrint(VirtualMachine VM);
```

GotoCode

```
public void init(ArrayList<String> args);
public void execute(VirtualMachine VM);
public String toPrint(VirtualMachine VM);
public void setAddress();
public String getFuncName();
```

LitCode

```
public void init(ArrayList<String> args);
public void execute(VirtualMachine VM);
public String toPrint(VirtualMachine VM);
```

StoreCode

```
public void init(ArrayList<String> args);
public void execute(VirtualMachine VM);
public String toPrint(VirtualMachine VM);
```

FalseBranch

```
public void init(ArrayList<String> args);
public void execute(VirtualMachine VM);
public String toPrint(VirtualMachine VM);
public void setAddressoffunc();
public String getFuncName();
```

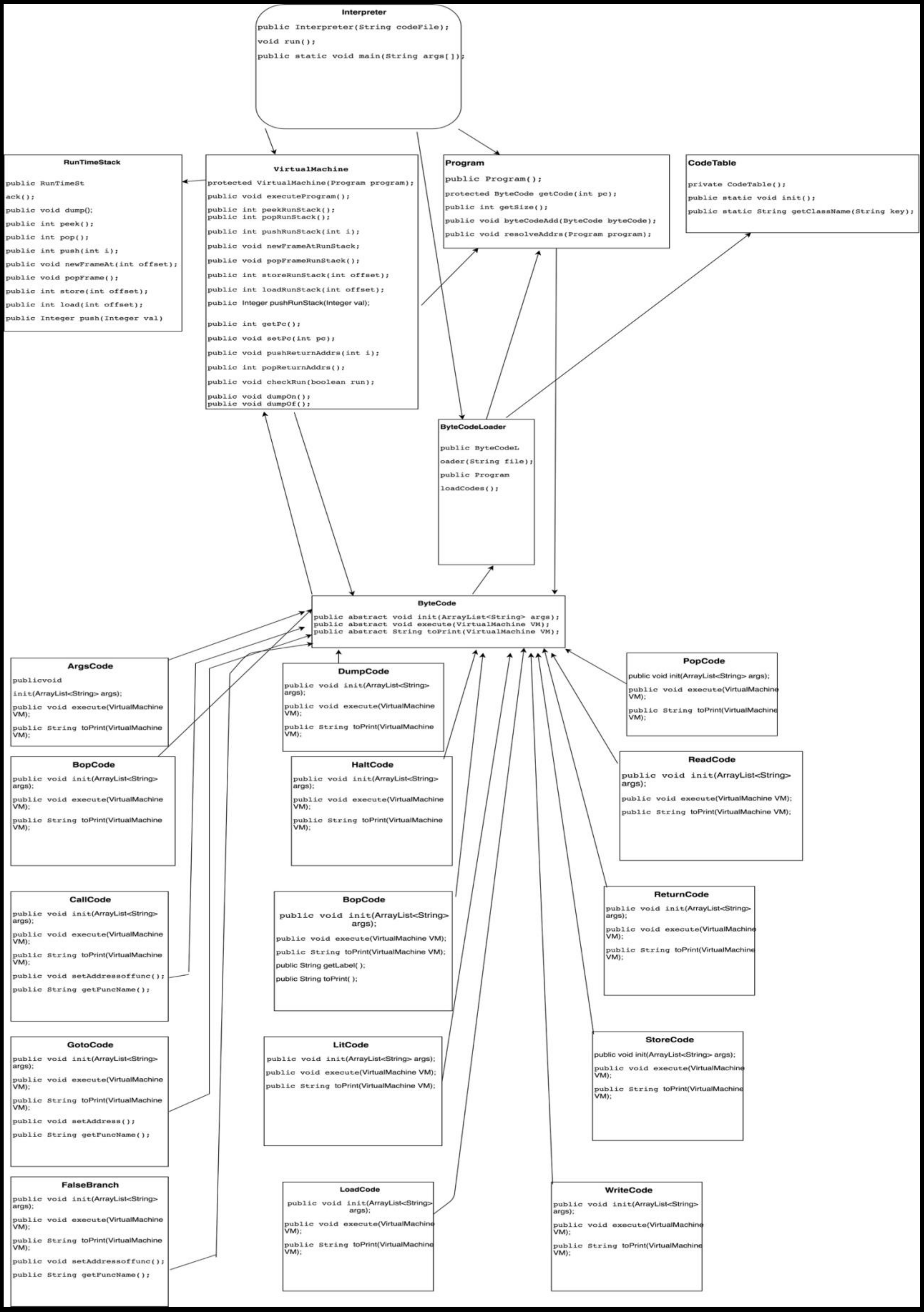
LoadCode

```
public void init(ArrayList<String> args);
public void execute(VirtualMachine VM);
public String toPrint(VirtualMachine VM);
```

WriteCode

```
public void init(ArrayList<String> args);
public void execute(VirtualMachine VM);
public String toPrint(VirtualMachine VM);
```

Actual diagram made on draw.io



```
Interpreter
public Interpreter(String codeFile);
void run();
public static void main(String args[]);
```

```
RunTimeStack
public RunTimeSt
ack();
public void dump();
public int peek();
public int pop();
public int push(int i);
public void newFrameAt(int offset);
public void popFrame();
public int store(int offset);
public int load(int offset);
public Integer push(Integer val)
```

```
VirtualMachine
protected VirtualMachine(Program program);
public void executeProgram();
public int peekRunStack();
public int popRunStack();
public int pushRunStack(int i);
public void newFrameAtRunStack;
public void popFrameRunStack();
public int storeRunStack(int offset);
public int loadRunStack(int offset);
public Integer pushRunStack(Integer val);
public int getPc();
public void setPc(int pc);
public void pushReturnAddrs(int i);
public int popReturnAddrs();
public void checkRun(boolean run);
public void dumpOn();
public void dumpOf();
```

```
Program
public Program();
protected ByteCode getCode(int pc);
public int getSize();
public void byteCodeAdd(ByteCode byteCode);
public void resolveAddrs(Program program);
```

```
CodeTable
private CodeTable();
public static void init();
public static String getClassName(String key);
```

```
ByteCodeLoader
public ByteCodeL
oader(String file);
public Program
loadCodes();
```

```
ByteCode
public abstract void init(ArrayList<String> args);
public abstract void execute(VirtualMachine VM);
public abstract String toPrint(VirtualMachine VM);
```

```
ArgsCode
publicvoid
init(ArrayList<String> args);
public void execute(VirtualMachine
VM);
public String toPrint(VirtualMachine
VM);
```

```
DumpCode
public void init(ArrayList<String>
args);
public void execute(VirtualMachine
VM);
public String toPrint(VirtualMachine
VM);
```

```
PopCode
public void init(ArrayList<String> args);
public void execute(VirtualMachine
VM);
public String toPrint(VirtualMachine
VM);
```

```
BopCode
public void init(ArrayList<String>
args);
public void execute(VirtualMachine
VM);
public String toPrint(VirtualMachine
VM);
```

```
HaltCode
public void init(ArrayList<String>
args);
public void execute(VirtualMachine
VM);
public String toPrint(VirtualMachine
VM);
```

```
ReadCode
public void init(ArrayList<String>
args);
public void execute(VirtualMachine VM);
public String toPrint(VirtualMachine VM);
```

```
CallCode
public void init(ArrayList<String>
args);
public void execute(VirtualMachine
VM);
public String toPrint(VirtualMachine
VM);
public void setAddressoffunc();
public String getFuncName();
```

```
BopCode
public void init(ArrayList<String>
args);
public void execute(VirtualMachine VM);
public String toPrint(VirtualMachine VM);
public String getLabel( );
public String toPrint( );
```

```
ReturnCode
public void init(ArrayList<String>
args);
public void execute(VirtualMachine
VM);
public String toPrint(VirtualMachine
VM);
```

```
GotoCode
public void init(ArrayList<String>
args);
public void execute(VirtualMachine
VM);
public String toPrint(VirtualMachine
VM);
public void setAddress();
public String getFuncName();
```

```
LitCode
public void init(ArrayList<String> args);
public void execute(VirtualMachine VM);
public String toPrint(VirtualMachine VM);
```

```
StoreCode
public void init(ArrayList<String> args);
public void execute(VirtualMachine
VM);
public String toPrint(VirtualMachine
VM);
```

```
FalseBranch
public void init(ArrayList<String>
args);
public void execute(VirtualMachine
VM);
public String toPrint(VirtualMachine
VM);
public void setAddressoffunc();
public String getFuncName();
```

```
LoadCode
public void init(ArrayList<String>
args);
public void execute(VirtualMachine
VM);
public String toPrint(VirtualMachine
VM);
```

```
WriteCode
public void init(ArrayList<String>
args);
public void execute(VirtualMachine
VM);
public String toPrint(VirtualMachine
VM);
```

a) ByteCodeLoader: In this class we were supposed to read the x file and store it into an ArrayList which was contained in the program class.

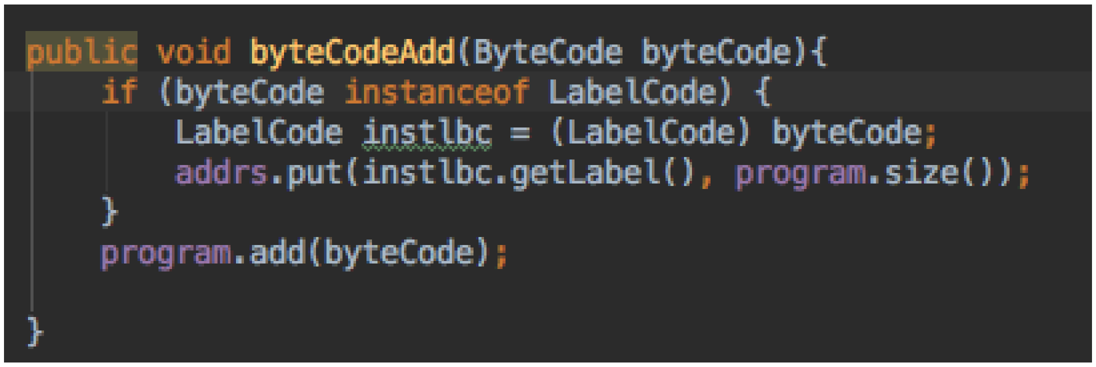
Steps for ByteCodeLoader:

1) Used the .readLine method to read the .x.cod file which had bytecodes.
2) Made sure while the file wasn't empty tokenize the file.
3) Also initialized another ArrayList to add in case the file had more tokens.
4) Then to create instances of I used the code given on the pdf by the professor.
5) Called the init method and passed ArrayList arr in the parameter, called the byteCodeAdd method in program class all this was done in a try catch block to handle multiple exceptions.
6) At the end called the resolveAddress method in the program class and return program.

b) Program: Once the bytecodes are stored we use the program class to resolve the symbolic addresses.

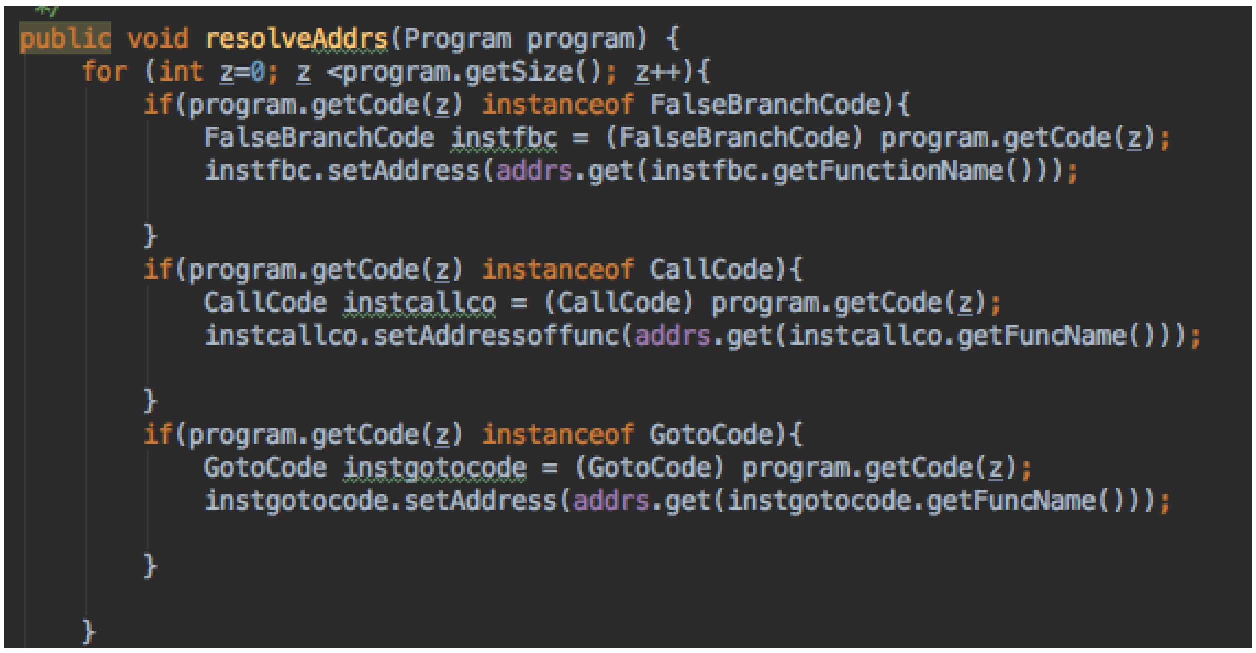
Steps for Program:

1) First, I created a hashmap and used a method called byteCodeAdd to see if it was an instance of the LabelCode subclass then cast it and put it to the hashmap.



```
public void byteCodeAdd(ByteCode byteCode){
    if (byteCode instanceof LabelCode) {
        LabelCode instlbc = (LabelCode) byteCode;
        addrs.put(instlbc.getLabel(), program.size());
    }
    program.add(byteCode);

}
```

2) In the resolveAddrs method I ran a for loop that runs the Arraylist that has all the bytecodes and see if it is an instance of Goto, FalseBranchCode or CallCode.



```
public void resolveAddrs(Program program) {
    for (int z=0; z <program.getSize(); z++){
        if(program.getCode(z) instanceof FalseBranchCode){
            FalseBranchCode instfbc = (FalseBranchCode) program.getCode(z);
            instfbc.setAddress(addrs.get(instfbc.getFunctionName()));

        }
        if(program.getCode(z) instanceof CallCode){
            CallCode instcallco = (CallCode) program.getCode(z);
            instcallco.setAddressoffunc(addrs.get(instcallco.getFuncName()));

        }
        if(program.getCode(z) instanceof GotoCode){
            GotoCode instgotocode = (GotoCode) program.getCode(z);
            instgotocode.setAddress(addrs.get(instgotocode.getFuncName()));

        }

    }
```

3) After checking if it's an instance of the following subclass it will cast and set the address to what was found in the HashMap. Giving us resolved symbolic address.

c) **CodeTable:** This class was done for us it has a Hashmap that maps the source code bytecode with their respected classes.

d) **RuntimeStack:** This class is responsible for maintaining the stack of active frames.

The class is used heavily by the VM class.

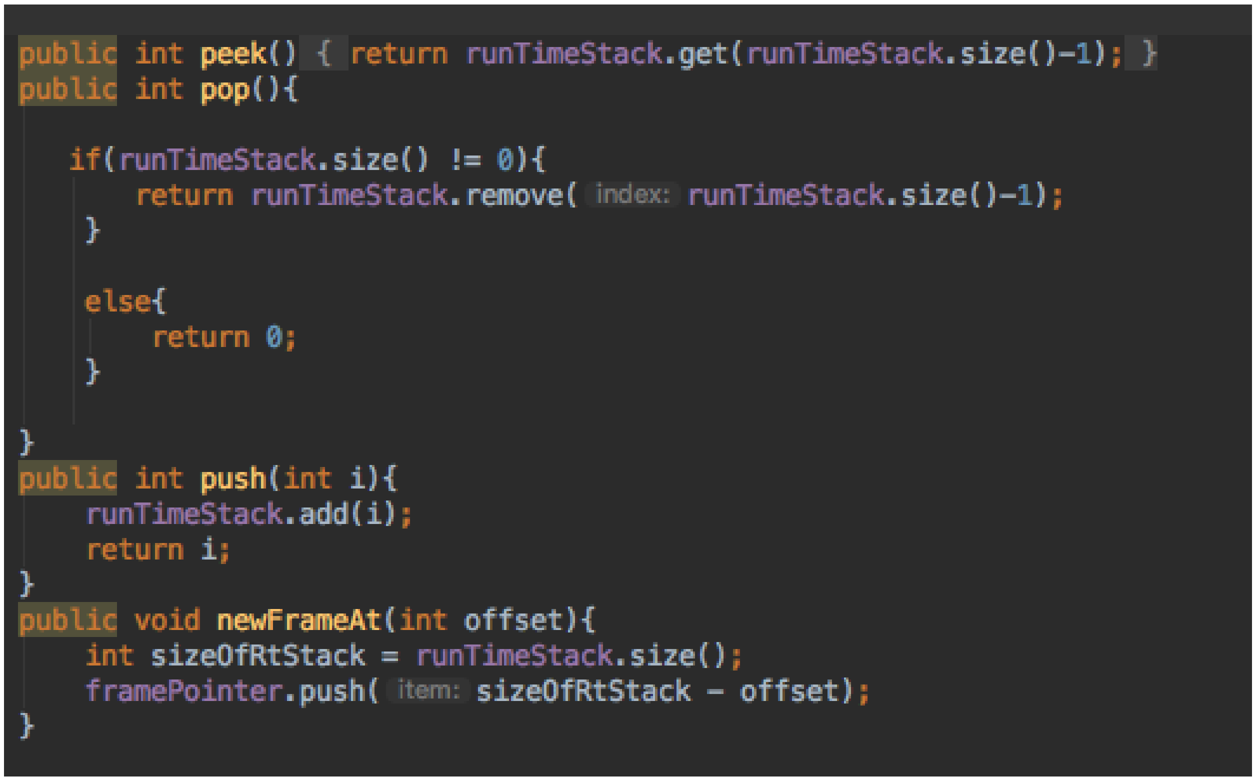
Methods of RuntimeStack:

Peek: used .get to get what's at the top of the stack.

Pop: used if statement to check if the stack isn't empty and remove the item at the top of the stack. If stack is empty return 0.

Push: To add to the top of the RuntimeStack.

NewFrameAt: used to create a new frame. Offset tells us by how many spots we down by from the top to start a new frame.



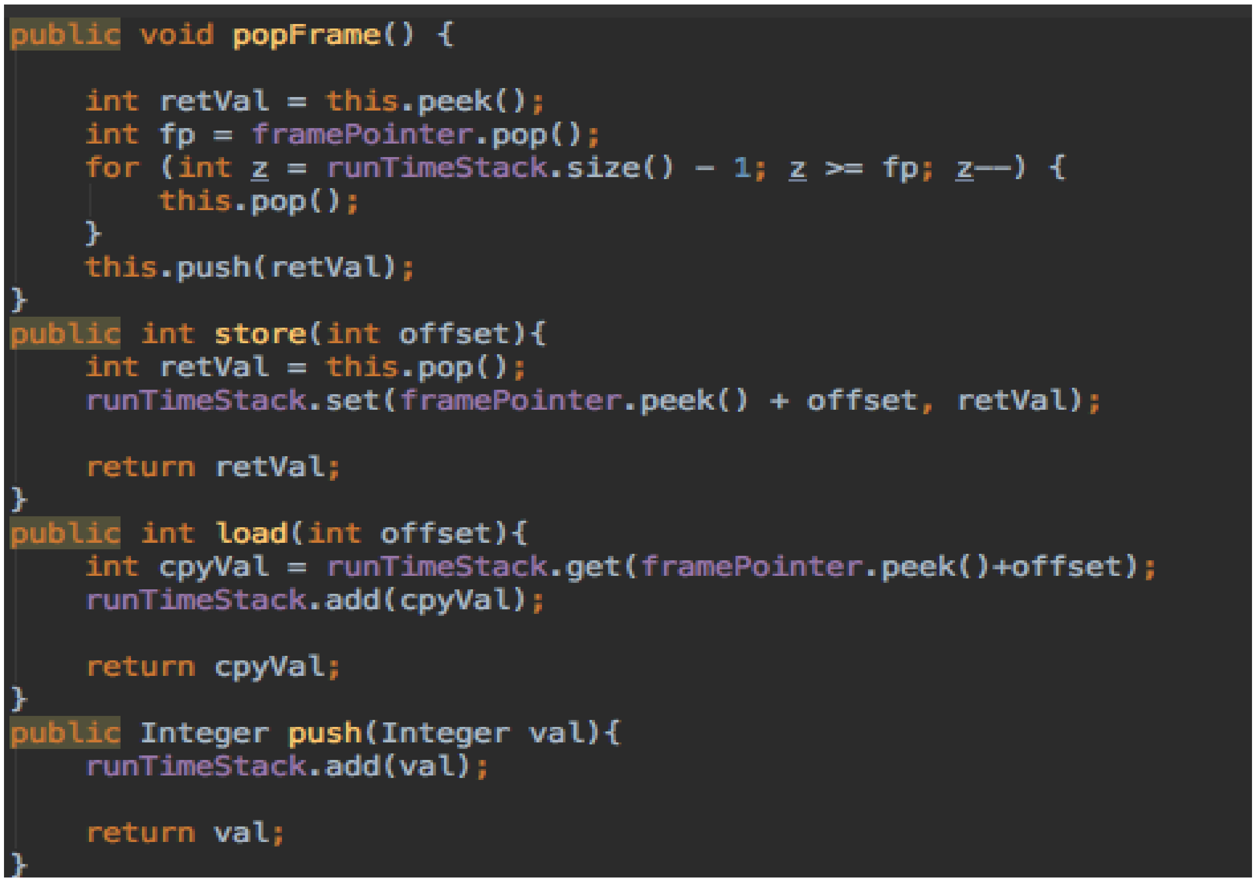
```
public int peek() { return runTimeStack.get(runTimeStack.size()-1); }
public int pop(){

   if(runTimeStack.size() != 0){
       return runTimeStack.remove( index: runTimeStack.size()-1);
    }

    else{
        return 0;
    }

}
public int push(int i){
    runTimeStack.add(i);
    return i;
}
public void newFrameAt(int offset){
    int sizeOfRtStack = runTimeStack.size();
    framePointer.push( item: sizeOfRtStack - offset);
}
```

PopFrame: retVal gives the top of the stack. Variable fp pops the framePointer and stores it. Then I ran a for loop that starts with the top of the stack, pop's the and push it to the top of the stack.

Store: pop's the top value of the stack stores it in retVal and then replace that value by using .set with the given offset.

Load: cpyVal get's the value in the current frame at the given offset and then add it to the top of the stack.

Push: For subclass LIT to add to the stack.

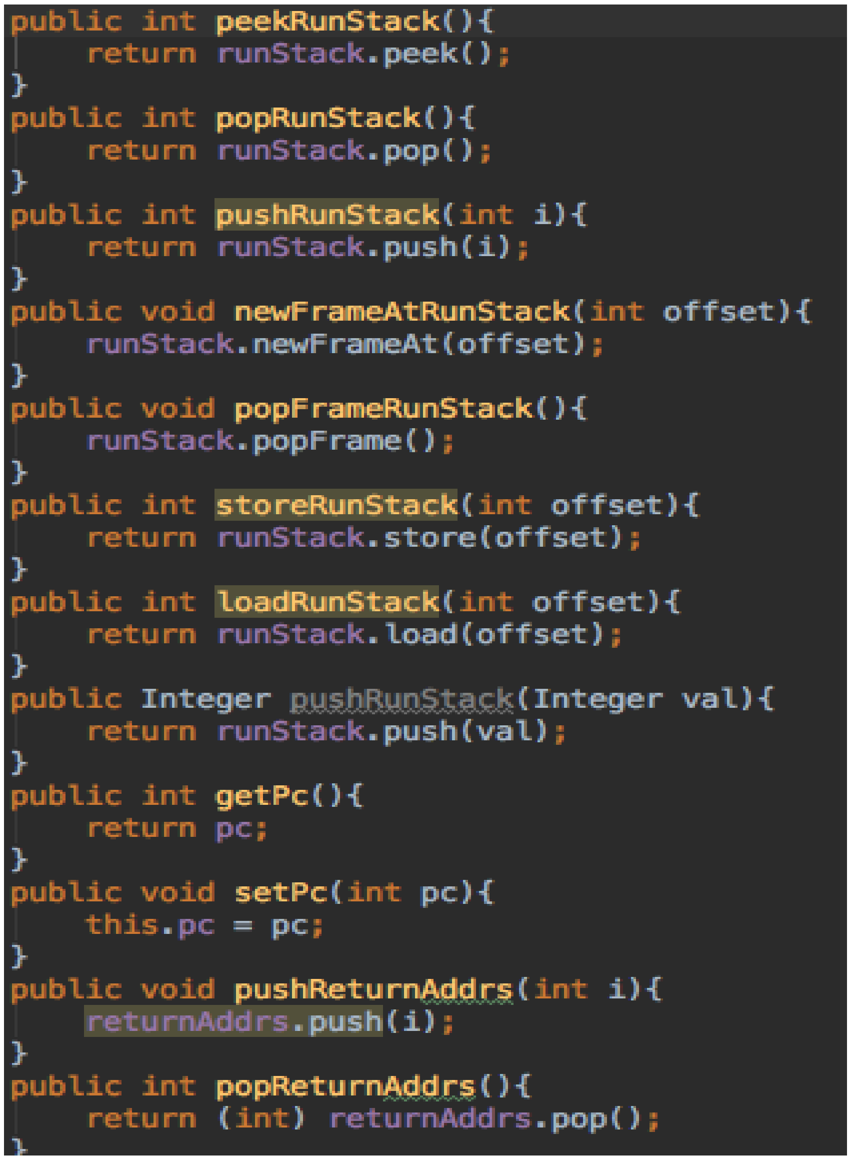


```
public void popFrame() {

    int retVal = this.peek();
    int fp = framePointer.pop();
    for (int z = runTimeStack.size() - 1; z >= fp; z--) {
        this.pop();
    }
    this.push(retVal);
}
public int store(int offset){
    int retVal = this.pop();
    runTimeStack.set(framePointer.peek() + offset, retVal);

    return retVal;
}
public int load(int offset){
    int cpyVal = runTimeStack.get(framePointer.peek()+offset);
    runTimeStack.add(cpyVal);

    return cpyVal;
}
public Integer push(Integer val){
    runTimeStack.add(val);

    return val;
}
```

e) **Virtual Machine:** This class is used to run program. It behaves like a controller.

VM uses program class to get the bytecode that were loaded to the byteLoaderClass. The while loop in the executeProgram method makes sure the VM can execute the bytecodes.

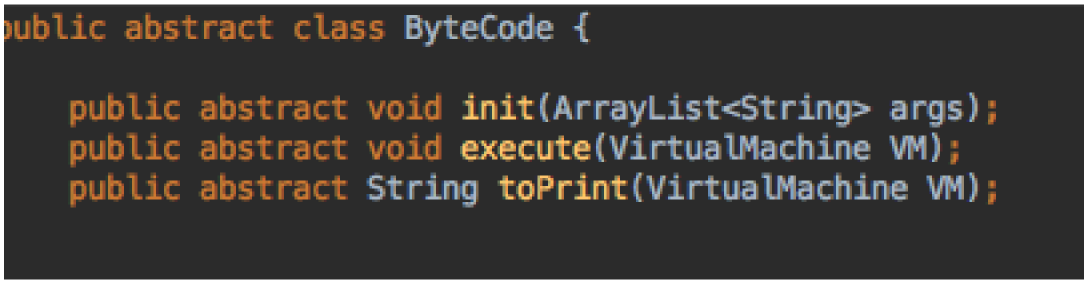
The methods below are used to access the RunTimeStack class to avoid breaking encapsulation. To be used in the bytecode subclasses.



```
public int peekRunStack(){
    return runStack.peek();
}
public int popRunStack(){
    return runStack.pop();
}
public int pushRunStack(int i){
    return runStack.push(i);
}
public void newFrameAtRunStack(int offset){
    runStack.newFrameAt(offset);
}
public void popFrameRunStack(){
    runStack.popFrame();
}
public int storeRunStack(int offset){
    return runStack.store(offset);
}
public int loadRunStack(int offset){
    return runStack.load(offset);
}
public Integer pushRunStack(Integer val){
    return runStack.push(val);
}
public int getPc(){
    return pc;
}
public void setPc(int pc){
    this.pc = pc;
}
public void pushReturnAddrs(int i){
    returnAddrs.push(i);
}
public int popReturnAddrs(){
    return (int) returnAddrs.pop();
```

f) **Interpreter:** This class was also done for us and is the starting point of the project.

g) **ByteCode Subclasses:** First I made an abstract subclass called bytecode and made three abstract methods. All the other subclasses inherited these methods helping with encapsulation.



```
ublic abstract class ByteCode {

   public abstract void init(ArrayList<String> args);
   public abstract void execute(VirtualMachine VM);
   public abstract String toPrint(VirtualMachine VM);
```

7) Reflection: This was definitely the hardest project I have done as a CS student. I spent a day on reading the pdf and had to read atleast five times before it made any sense. I felt the byteCodeLoader class was the least amount of work. I spent a lot of time on program class to figure out how to resolve the symbolic address. Runtimestack class was the one I spent the most time on and after I ran my code most of the debugging errors were in this class as well. I enjoyed doing the VM except for the dump part. I feel it would've been better if the professor gave us a test file that had dump on and dump off. One of the dumbest mistakes I made after I finished coding was that I set dump to false in the VM which didn't print anything I struggled a good hour before I realized where the error was. Also, I am very grateful that the professor extended the deadline. If the project was due on Friday I wouldn't have finished.

8) Conclusion and Results: Overall, I believe I learned a lot from this project. I've never worked on a project where I've had to keep track of so many files. As for the results, they were consistent with the output file the professor gave. I tested the output individually and the number entered by the user gave the right Fibonacci and factorial answer.