Name: Thanh Le
Class: CSC 413

Computer Science Department
San Francisco State University
CSC 413
Spring 2018

Assignment 3 - The Interpreter

Link to github repository: https://github.com/csc413-02-sp18/csc413-p3-tle25.git

Project introduction and overview:

- For this project, I am required to implement all the bytecodes and uncompleted files under the mock language X.
- I have implemented all bytecode classes in the table codes.
- I also create an abstract class named ByteCode.java , with two abstract methods for the bytecode classes.
- There are other four classes that I implemented;
- ByteCodeLoader.java which instantiated bytecode classes and push them to Program arrayList
- Program.java which holds all instance classes. This program also helped to resolve all the address on its array.
- VirtualMachine.java is a executor branch. It asks an object bytecode from program and execute it one by one.
- RunTimeStack.java is a receiver class. It will do as VM command ask it do.

The general idea about how this program works:

- First, Interpreter takes file.x.cod and passes it to ByteCodeLoade. After all bytecode are modified and pushed to ArrayList in Program.

- In Program class, it has reSolveAddress function. This function allow the program go through al lthe bytecode classes. It will convert all the addresses. What it means is that when Call, Goto, or FalseBranch class wants to jump to specific location. It must set the Program Counter to the address it wants to jump. Doing that, Label will be the target it will want to jump to, everytime Call, Goto, falseBranch send thei arguments to label hashmap, they will get the address from that location.

- In order to do that, I create a hashmap label<String, Int> arg as the key, and its current address as value. Whenever classes CallCode, GotoCode, FalseBranchCode are recognized, send it value to label and get the return address. Set this this address to PC so that VM will know where to go.

- For the VirtualMachine class, it works as a leader that load all bytecode from program and execute them one by one.

- Each of the bytecode class has a specific task. When bytecode is executed, it will request VM to do the operation. However, I am not allowed to break encapsulation rule. The reason we use encapsulation in this project is to give user be able to use its methods and but hide all detail information from its class.

- When VM is received requests from bytecodes, it will call RunTimeStack class to the push, pop, peek, or peek, FramePointer, pushFramePointer, restore, load, etc. For this class, it has two different arguments runTimeStack and framePointer. RuntimeStack is used to keep all the data from the functions, and framePointer is used to store all the starting frames at on the stack. These two arugment are the based to do the Dump() output.

The DUMPing statement will be writen in RunTimeStack class. When dump is off, it will continue execute the program, but hide output.

It is all about the program and how it works.

- Bellow is what bytecode classes supposed to do.

Bytecode	Description	Examples
HALT	*halt* execution	HALT
POP	*POP n*: Pop top *n* levels of runtime stack	POP 5 POP 0
FALSEBRANCH	*FALSEBRANCH <label>* - pop the top of the stack; if it's *false* (0) then branch to <label> else execute the next bytecode	FALSEBRANCH xyz<<3>>
GOTO	*GOTO <label>*	GOTO xyz<<3>>
STORE	*STORE n <id>* - pop the top of the stack; store value into the *offset n from the start of the frame*; <id> is used as a comment, it's the variable name where the data is stored	STORE 2 i
LOAD	*LOAD n <id>*; push the value in the slot which is *offset n from the start of the frame* onto the top of the stack; <id> is used as a comment, it's the variable name from which the data is loaded	LOAD 3 j
LIT	*LIT n* - load the *literal value n* *LIT 0 i* – this form of the Lit was generated to load 0 on the stack in order to initialize the variable *i* to 0 and reserve space on the runtime stack for *i*	LIT 5 LIT 0 i
ARGS	*ARGS n* ;Used prior to calling a function: n = #args this instruction is *immediately followed* by the *CALL* instruction; the function has *n args* so *ARGS n* instructs the interpreter to set up a new frame *n down from the top*, so it will include the arguments	ARGS 4
CALL	*CALL <funcname>* - transfer control to the indicated function	CALL f CALL f<<3>>
RETURN	*RETURN <funcname>* ; Return from the current function; *<funcname>* is used as a comment to indicate the current function *RETURN* is generated for intrinsic functions	RETURN f<<2>> RETURN
BOP	*bop <binary op>* - *pop top 2 levels* of the stack and perform indicated operation – operations are + - / * == != <= > >= < | & *| and & are logical operators, not bit operators* lower level is the *first operand*: e.g. *<second-level> + <top-level>*	BOP +
READ	*READ* ; Read an integer; prompt the user for input; put the value just read on top of the stack	READ
WRITE	*WRITE* ; Write the value on top of the stack to output; *leave the value on top of the stack*	WRITE
LABEL	*LABEL <label>*; target for branches; (see *FALSEBRANCH, GOTO*)	LABEL xyz<<3>> LABEL Read

The task of this project is to practice with Java Encapsulation. There are so many different classes I need to implement. Luckily, I completed this Assignment on time. This project works for all input

Development Environment

- The program is developed in my laptop and written on Netbeans IDE 8.2
- I use JDK8.
- There many method/object/functions I don't know from Java library, so I google and use stackoverFlow to get hints about them.
-

Instruction to compile and execute

First, download the all the scr files in Assignment #3

- **You must clone my program. Using this link to access to my repo**
  https://github.com/csc413-02-sp18/csc413-p3-tle25.git

When you cloned it, I was using netbean with JDK 8. So I don't know you can run it with other IDE or not, except JDK 8 and netbean 8.2.

- Open the repo that you cloned. Right click on the project and set the .x.cod file you want to execute. Also change the working directory. Then click play button and run it.
When you run the program, it will ask you enter the number that you want to execute.

This program is created with dump method. This project is running under the file factorial.x.cod. This class I modified with some extra lines: DUMP ON and DUMP OFF. It will only output the line when DUMP is turn ON.

If you want to see all the output from the program, you can go to DumpCode.java, set its flag to true. It will display all the output. I did not modify the fib.x.cod, you can test this class.

Assumption

There is a lot of things for this project. I read the pdf many times, googling and asking other students for help. When I finish the program, I feel like I completed understand its process.

Implementation discussion

Interpreter project:

>>>Interpreter.java ← main class

Constructor: Interpreter(String codeFile)

main(String[] args)

Run()

This is the main class. It will and send x.cod file to ByteCodeLoader. Then run Execute with an object parameter

>>>TableCode.java

Tablecode has a hashmap, whenever code is called, it will send the bytecode class to the function calling it. It has implet

>>>VirtualMachine.java

This class will execute all bytecode stored in Program ArrayList, one by one

Each bytecode will have different task. So when execute is called, it will return different value.

This class will also call write dump(), this will write all the argument from runStack.

>>>Program.java

The class will have has an ArrayList program and a hashmap label

The hashmap label will store all label argument as key, and its current address as value. This method is used to get the address from label when CALL, GOTO, FALSEBRACH is called

The method addCode will add all instance bytecodes to arrayList program. The ArrayList will hold all the bytecode class.

The method resolveAddress will check if arraylist program contains classes CALL, FALSEBRANCH, GOTO. When these classes are found in array, they will be converted into correct addresses so the VirtualMachine knows what to set the Program Counter(PC). To do this, the Hashmap label will takes their aruments where it wants to jump to, then return the address to it. They will take the addresses and set to PC.

>>>RunTimeStack.java

RunTimeStack: constructor, create object of RTS and FramePoninter. Initialize framePointer at 0.

dump(): this will display all output from stack
push(int): push int value on top of RTS
push(Integer): push Integer value on top of RTS. like lit 5 ← is an Integer
peek() : peek from the top of stack
getSize(): return stack size
pop(): get value on the top and pop
popFrame(): pop the the current frame
newFrameAt(int): creating a new frame.
load(int) : load the value from n index of current frame
store(int): store value at n index of current frame

>>>>ByteCode.java

Abstack class with 2 abstract methods
Abstrack public abstract void execute(VirtualMachine vm);
Abstrack public abstract void init(ArrayList list);

Subclassess

- ArgsCode.java
  exectue(VirtualMachine)
  inti(ArrayList)
  toString() : output all the object

- BopCode.java
  exectue(VirtualMachine): operate the first two top of runStack, with an operator
  When it is done, push the value on top of Stack

inti(ArrayList)
toString() : output all the object

- ArgsCode.java
  exectue(VirtualMachine) : it call VM method to create a new frame with arg n
  inti(ArrayList)
  toString() : output all the object

- CallCode.java
  exectue(VirtualMachine) : set PC to where address from updatePC() method its want to jump, save it current address so that it will be able return back to this
  inti(ArrayList)
  toString() : output all the object
  getString(int) : send the arg to resolveProgram
  updatePC(void) : contains the label address and it current address.

- DumpCode.java
  exectue(VirtualMachine)
  inti(ArrayList)
  toString() : output ON or OFF,

- FalseBranchCode.java set PC to where address from updatePC() method its want to jump
  exectue(VirtualMachine)
  inti(ArrayList)
  toString() : output all the object
  getString(int) : send the arg to resolveProgram
  updatePC(void): contains the label address

- GotoCode.java
  exectue(VirtualMachine)
  inti(ArrayList)
  toString() : output all the object

- HaltCode.java
  exectue(VirtualMachine) : set isRunning to false
  inti(ArrayList)
  toString() :

- LabelCode.java
  exectue(VirtualMachine) :
  inti(ArrayList)
  toString() : output all the object

- LitCode.java
  exectue(VirtualMachine) : call the VM to push argument on top of Stack
  inti(ArrayList)
  toString() : output all the object

- LoadCode.java
  exectue(VirtualMachine) call VM to load the value at n argument of a top frame
  inti(ArrayList)
  toString() : output all the object

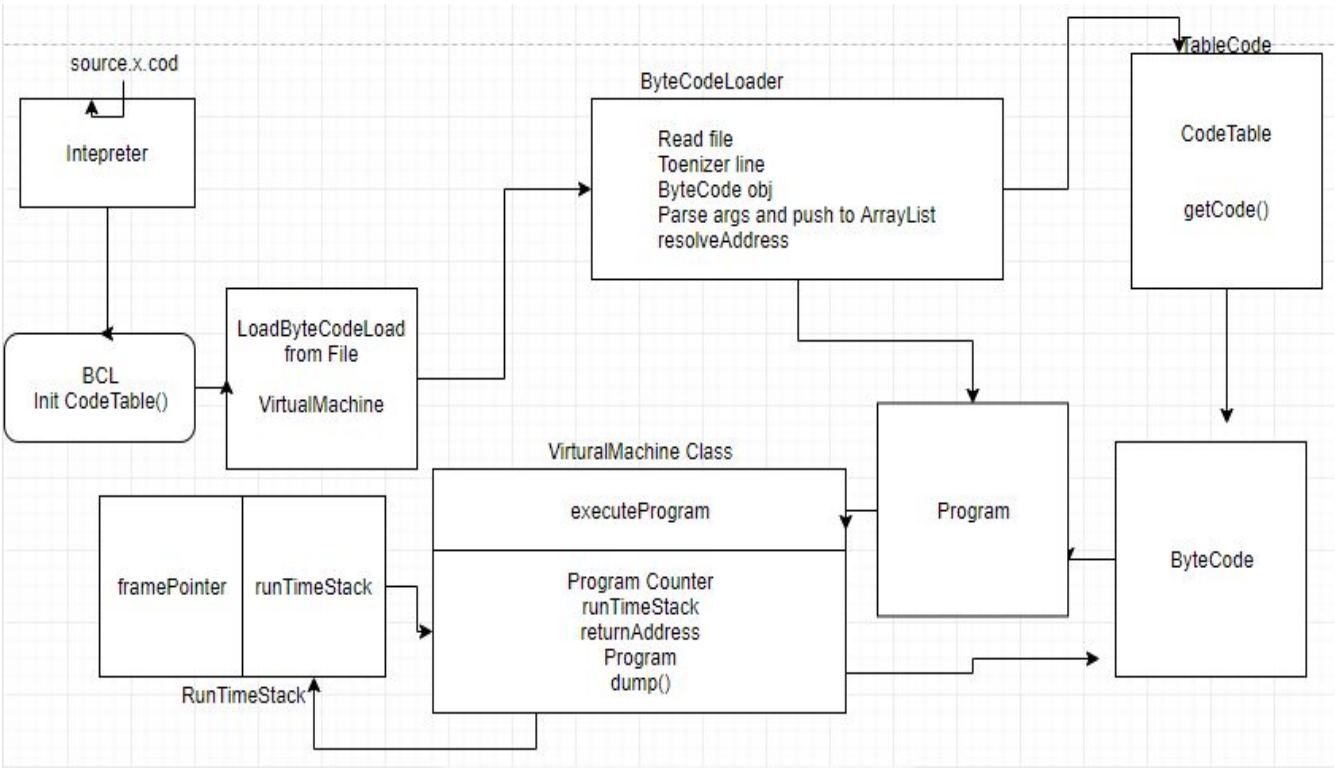
- PopCode.java
  exectue(VirtualMachine) call VM to pop n level(time) of RTS
  inti(ArrayList)
  toString() : output all the object

- ReadCode.java
  exectue(VirtualMachine) read the input, and call VM to push this input on top of Stack
  inti(ArrayList)
  toString() : output all the object

- ReturnCode.java :
  exectue(VirtualMachine) call Stack from VM, peek() it arug, it will return the address of CallClass, setPC to this address so that it will VM will jump back to the CallCode.
  inti(ArrayList)
  toString() : output all the object

- StoreCode.java
  exectue(VirtualMachine) : Store the current value returns from Call function. Call VM to push that value to the index of the current runStack.
  inti(ArrayList)
  toString() : output all the object

- WriteCode.java
  exectue(VirtualMachine) Call the VM to peek top of RTS
  inti(ArrayList)
  toString() : output all the object



Results and conclusions

The program works fine. I tested it many times and it worked for all inputs. From this project, I learned a basic encapsulation in java. I understand more how and when we should use abstract methods. This project also give me another chance to practice with hashmap and creating newInstance. I also learned the instanceof. This operator is new to me. What I know about this instanceof is it will check if a object or instance is a subtype of a given type or not. For example. Salary is an instance of Employee or Car is an instance of Vehicle.
I think these benefits will help a lot for my next project, Tank War.

In conclusion, I spent a lot of time for this project, more than 4 other classes combined. Most of it from understanding the process and its logic. At first, I got mad about this project because I did not understand I was supposed to do. It confused me a lot. I planed to give up for this one because I did not think I could do it. I only knew this project when Professor Anthony sketched the diagram how this program work. Luckily, I can finish this Assignment on time. Even though I completed this Assignment on time, and it works, I think there is some error that I still can not figure out. About myself, I think I am a bit behind from this field. I think there is a lot of thing about Java I don't know. This assignment gives me another chance to practice with it.