Brandon Watanabe
921988506
21 April, 2022

Assignment 4 Documentation

Github Repository

https://github.com/sfsu-csc-413-fall-2022-roberts/assignment-4---interpreter-BrandonWatanabeStudentAccount.git

Project Introduction and Overview

For this project I had to create an interpreter that reads a byte code file and performs the necessary operations for each line.

Summary of Technical Work

The Interpreter class takes in a file and runs the program. The ByteCode class is an abstract class that is the foundation for all of the different ByteCode subclasses. These subclasses are created for each different type of bytecode and implements the byte code's behavior. The CodeTable class stores the mapping between byte codes and the byte code class that is associated with it. The ByteCodeLoader class reads each line from the file and creates instances of ByteCode subclasses and stores them in a Program object. The Program class stores the byte codes in order and resolves any symbolic addresses. The RunTimeStack class records the data stored in the current runtime stack and tracks the activation records in a framePointers Stack. The VirtualMachine class executes the commands for each byte code.

Scope of Work

Task		Completed
Implemented in bytecode package		X
	ArgsCode	X
	BopCode	X
	ByteCode	X
	CallCode	X
	DumpCode	X
	FalseBranchCode	X
	GoToCode	X
	HaltCode	X
	LabelCode	X
	LitCode	X

	LoadCode	X
	PopCode	X
	ReadCode	X
	ReturnCode	X
	StoreCode	X
	WriteCode	X
Implemented and updated in ByteCodeLoader.java		X
	public ByteCodeLoader(String byteCodeFile)	X
	public Program loadCodes()	X
Implemented and updated in CodeTable.java		X
	public static void init()	X
	private static void populateByteCodeMap()	X
	public static String get(String code)	X
Implemented and updated in Program.java		X
	public void addCodes(HashMap<Integer,ByteCode> inputMap, ArrayList<String[]> argsList)	X
	public ByteCode getCode(int programCounter)	X
Implemented and updated in RunTimeStack.java		X

	public RunTimeStack()	X
	public String dump()	X
	public int peek()	X
	public int pop()	X
	public void popN(int n)	X
	public int push(int item)	X
	public void newFrameAt(int offset)	X
	public void popFrame()	X
	public void store(int offset)	X
	public void load(int offset)	X
	public void bop(String op)	X
	public void write()	X
Implemented and updated in VirtualMachine.java		X
	public void executeProgram()	X
	public void handleDump(ByteCode code)	X
	public void newFrameAt(int n)	X
	public void bop(String op)	X

	public void storeCurrentPC()	X
	public void setPC(int n)	X
	public void setDumpState(boolean b)	X
	public int popRunStack()	X
	public void haltProgram()	X
	public void push(int n)	X
	public void load(int n)	X
	public void popN(int n)	X
	public int getReturn()	X
	public void popFrame()	X
	public void store(int n)	X
	public void write()	X
Ran using the following file		X
	GOTO start<<1>> LABEL Read READ RETURN LABEL Write LOAD 0 dummyFormal WRITE RETURN LABEL start<<1>> DUMP ON LIT 0 i LIT 0 j	X

	LOAD 0 i LOAD 1 j BOP + LIT 7 BOP + STORE 0 i DUMP OFF LOAD 0 i ARGS 1 CALL Write STORE 1 j POP 2 HALT	

Execution and Development Environment

I used Java 17 on the IntelliJ IDE on my Windows computer to complete this assignment and tested in both the IDE and shell.

Compilation Result

Using the instructions provided in the assignment one specification:

```
LIT 0 i                  int i

[0][0]

LIT 0 j                  int j

[0, 0][0, 0]

LOAD 0 i                 <load i>

[0, 0, 0][0, 0, 0]

LOAD 1 j                 <load j>

[0, 0, 0, 0][0, 0, 0, 0]

BOP +

[0, 0, 0][0, 0, 0]

LIT 7

[0, 0, 0, 7][0, 0, 0, 7]

BOP +

[0, 0, 7][0, 0, 7]
```

```
STORE 0 i               i = 0

[7, 0][7, 0]

Please enter an integer:

1
```

No error messages or warnings were displayed, and the application ran as expected.

Assumptions

I assumed that all of the provided skeleton code was correct and did not alter it.

Implementation

ByteCode

The ByteCode class is an abstract class that is the parent class to all of the different ByteCode subclasses in the bytecode package. The purpose of these classes is to implement the behavior of each type of bye code, which includes any member data that it requires to run. First, I created all of the different abstract methods that I thought I might need for each subclass. Then I filled in each method for all the different subclasses. During this process I realized that some subclasses would need extra methods that the others did not need so I implemented those as well.

CodeTable

The CodeTable class stores the relations between the bytecode names and their respective ByteCode subclass. To do this, first, I created an init method that runs the private method to populate the HashMap that stores this data. This method reads a file with all of the different types of bytecodes and the name of the subclass. It then takes these and assigns them to the HashMap. I also implemented a get method to access the HashMap.

ByteCodeLoader

The ByteCodeLoader class reads the bytecode file from Interpreter.java and creates instances of each bytecode and stores them in a Program object. First, I made a method to read the file and process each line. I split up the line by whitespace and assign it to an array. The first element in the array is used to create the instance of the ByteCode subclass by referencing the CodeTable and using reflection. The new ByteCode object is then stored in a hashmap with its line number to keep them in order. This hashmap and the array are then passed into a Program object for processing.

Program

The Program class is responsible for storing the ByteCodes in order and resolving any symbolic references. It takes in a Hashmap and string array from the ByteCodeLoader class. It then copies the HashMap and assigns the line number of the symbolic reference to its correct address. It also initialized the different ByteCode objects using the init method.

Code Organization

I implemented all of the new classes and methods in the same style as the ones provided so that it was easy to read and understand what additions were made.

Class Diagram

Results and Conclusion

This project was a good challenge for me and really helped me understand object oriented programing in a way that I did not before. I got to see how all of these different classes had different roles and all intertwined with each other.

Challenges

The most challenging part of this assignment for me was creating the Program class. Specifically, resolving the symbolic addresses.

Future Work

A way to build on this would be to add functionality for more ByteCodes.