Description

In Parser that determines the validity of a SIMPLE program. In this section, we'll extend that project by generating SIMPLE Bytecode for each program and interpreting that Bytecode. There are two parts-- a code generator and a SIMPLE Bytecode interpreter that runs programs.

Code Generator

The job of the generator is to generate SIMPLE Bytecode. SIMPLE Bytecode has the following operations.

OPERATION	OP CODE	DESCRIPTION
LOAD	0	Add value at address in operand to Accumulator
LOADI	1	Add value of operand to Accumulator
STORE	2	Store value in Accumulator to address of operand Set Accumulator to 0

For this system, the *Accumulator* represents a lone CPU register. You will represent it as an integer.

Each operation has a single operand. So the operation "0 0" adds the value at address 0 into the accumulator. Operation "1 9" adds the number 9 to the value in the accumulator. "2 0" stores the value of the accumulator to address 0.

The following bytecode should be generated for the program “x = 5 y = x+3”:

code	generated bytecode	meaning
x = 5	1 5 2 0	load 5 in accumulator, store it to x’s address (0)
y= x+3	0 0 1 3 2 1	load address 0 (x) in accumulator, add 3, store in address 1 (y)

Interpreter

The Interpreter “runs” SIMPLE Bytecode programs, modifying memory which is represented with an array of integers.

Sample

The program “x = 5 y = x+3” generates the following Bytecode:

```
1 5 2 0 0 0 1 3 2 1
```

When that Bytecode program is interpreted, memory will appear as:

5	8	0	0	0	0	0	0	0	0

Variable “x” is stored at address 0, variable y is stored at address 1.

Memory is limited

The Interpreter should print “Run-time error: Address out of bounds” if an address outside of memory is referenced.

Java Specification

Place the code for generating Bytecode and executing it in a single class, “BytecodeInterpreter”. You’ll also modify the Parser class extensively for this project.

The BytecodeInterpreter should have the following:

- a constructor with a parameter specifying the size of memory.
- an ArrayList data member *bytecode* to hold the code for a program.
- an ArrayList data member *memory* to hold the main memory array.
- constants for the Bytecode commands LOAD, LOADI, and STORE.
- a private int data member for the *accumulator*.
- a private int data member for the size of memory.
- generate method-- this method takes in a command and an operand as parameters and adds the command to the bytecode being generated.
- run method-- this method runs the code in bytecode, modifying memory
- private methods for each command type, *run* will call these.
- get methods providing public access to appropriate data members.

Your program, including the classes from the Lexer-Parser, should define all non constant data members as private, and each class should provide appropriate toString and equals methods.

Design and specify extensions to the language, and implement them including updates to the Lexer, Parser, Code Generator, and Interpreter. Be creative, but run your extension ideas by Wolber. Simple ideas: add print, -, other math operators.

Instruction

pass provided Tests

pass tests with compiler errors (no output for symtab/bytecode/mm)
pass valid program tests that generate code and interpret
pass valid program test with address out of bounds
Java Specs
x6
data members in BytecodeInterpreter for bytecode and MM
constants for the Bytecode commands LOAD,LOADI,STORE.
private methods to run each command type, e.g., load
all data members as private in all classes
toString and equals in all classes
Style Guidelines
Javadoc-compliant comment for each class
Javadoc-compliant comment for each method

Testing:

program should be able to pass these tests.

test.txt

```
xyz = 33  zz12=99+88+xyz
```

Output:

```
Valid Program
Symbol Table: {xyz=0, zz12=1}
ByteCode:[1, 33, 2, 0, 1, 99, 1, 88, 0, 0, 2, 1]
Memory:[33, 220, 0, 0, 0, 0, 0, 0, 0, 0]
```

textExpectingId2.txt

```
32=54
```

Output:

```
Error: Expecting identifier, line 1
invalid Program
```

testExpectingAssignmentOp.txt

```
x32 54 =87
```

Output

```
Error: Expecting assignment operator, line 1
invalid Program
```

testExpectingIdOrInt2.txt

```
x = 32 32
```

Output

```
Error: Expecting identifier line 1
invalid Program
```

testMultiplePlus.txt

```
x32=77
yyy9=x32+5 + 4 +x32
```

Output

```
Valid Program
Symbol Table: {x32=0, yyy9=1}
ByteCode:[1, 77, 2, 0, 0, 0, 1, 5, 1, 4, 0, 0, 2, 1]
Memory:[77, 163, 0, 0, 0, 0, 0, 0, 0, 0]
```

testWhiteSpace.txt

```
   x32   =
83   rt=2
```

Output

```
Valid Program
Symbol Table: {rt=1, x32=0}
ByteCode:[1, 83, 2, 0, 1, 2, 2, 1]
Memory:[83, 2, 0, 0, 0, 0, 0, 0, 0, 0]
```

NEW--RUN TIME ERROR SAMPLE

Note: Please set your memory size to 10 in the code you submit. This app will have 11 items and should cause the error: Run-time error: Address out of bounds

testOutOfBounds.txt

```
a=1
b=2
c=3
d=4
e=5
f=6
g=7
h=8
i=9
```

```
j=10
k=11
```

Output

```
Valid Program
Symbol Table: {a=0, b=1, c=2, d=3, e=4, f=5, g=6, h=7, i=8, j=9, k=10}
Error: Address out of range
ByteCode:[1, 1, 2, 0, 1, 2, 2, 1, 1, 3, 2, 2, 1, 4, 2, 3, 1, 5, 2, 4, 1, 6, 2, 5, 1, 7, 2, 6, 1, 8, 2, 7, 1, 9, 2, 8, 1, 10, 2, 9, 1, 11, 2, 10]
Memory: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```