Homework 2 — Warm-up exercise in Java programming: Stack machine for integer list manipulation — assigned Tuesday 7 September, due Tuesday 14 September

Write an emulator for a hypothetical computer—a simple stack machine for integer list manipulation.

2.1 Specification: the list machine

2.1.1 General description

Consider a simple stack machine for integer list manipulation. The stack machine has a program *p*, a program counter *pc*, a stack *s* that may hold integers and lists, and a stack top pointer *sp*. The instructions of the machine and their effect on the stack are described by the following table:

Instruction	Stack before		Stack after	Effect
`CST` *i*	*s*	⇒	*i s*	Push integer constant i
`ADD`	$i_2\ i_1\ s$	⇒	$(i_1 + i_2)\ s$	Add integers
`SUB`	$i_2\ i_1\ s$	⇒	$(i_1 - i_2)\ s$	Subtract integers
`DUP`	*v s*	⇒	*v v s*	Duplicate
`SWAP`	$v_2\ v_1\ s$	⇒	$v_1\ v_2\ s$	Swap
`POP`	*v s*	⇒	*s*	Pop
`GOTO` *a*	*s*	⇒	*s*	Jump to *a*
`IFNZRO` *a*	*i s*	⇒	*s*	Jump to *a* if $i \neq 0$
`CALL` *a*	*v s*	⇒	*v r s*	Call function at *a*, pushing return address *r*
`RET`	*v r s*	⇒	*v s*	Return: jump to *r*
`MKNIL`	*s*	⇒	`Nil` *s*	Push `Nil` (the empty list)
`MKCONS`	*t i s*	⇒	`Cons`(*i*,*t*) *s*	Push cons node
`LISTCASE` *a*	`Nil` *s*	⇒	*s*	If `Nil`, do not jump
`LISTCASE` *a*	`Cons`(*i*,*t*) *s*	⇒	*t i s*	If `Cons`, unpack components and jump to *a*
`PRINT`	*v s*	⇒	*v s*	Print *v* and keep it
`STOP`	*s*	⇒	_	Halt the machine

In the table, *i* stands for an integer, *a* for an address within the program, *r* for an address within the program, *t* for a list, *v* for an arbitrary value, and *s* for the remainder of the stack (0 or more values).

An instruction with an argument (`CST`, `GOTO`, `IFNZRO`, `CALL`, `LISTCASE`) takes up two positions in the program.

Under any circumstances not covered by the table, the machine will crash. For instance, if the instruction to be executed is an `ADD` but the top stack element is a list rather than an integer, the machine will crash.

2.1.2 Code completion (25pts)

Given the skeleton of a list stack machine interpreter written in Java, provided below, complete the rest of the code.

Your code must be placed in a package named
`edu.unm.cs.cs351.fall2004.`*yourusername*`.liststackmachine`.

According to the provided code, you are to complete the method `execcode`. You must also write a `main` method, which must accept command-line arguments as follows: the first argument on the command line, a string, is the name of a file containing the "assembly" text of a program (i.e., machine instructions in human-readable form, as in the examples provided below); the second argument on the command line, an integer, is the initial value of the program counter; the remaining arguments (if any) on the command line, which are integers, are the initial contents of the stack, from top to bottom. Presumably you will write a separate `load` method, and, for debugging purposes, a `disassemble` method.

For example, if you put the assembly text:

```
0: CST 3
2: ADD
3: PRINT
4: POP
5: STOP
```

in a file `add3.lsm`, then you should be able to invoke the interpreter as follows:

yourfavoritejvm `edu.unm.cs.cs351.fall2004.`*yourusername*`.liststack\`
`machine.ListStackMachine add3.lsm 0 5`

and expect to see 8 printed.

2.1.3 Code skeleton

(From P. Sestoft, IT Copenhagen.)

```
abstract class List
{
}
class Nil extends List
{
    public String toString ()
    {
        return "Nil";
    }
}
class ListStackMachine
{
    private static int execcode (int [] p, Object [] s, int pc, int sp)
    {
        for (;;)
            {
                switch (p [pc++])
                    {
                    case CST:
                        s [sp+1] = new Integer (p [pc++]);
```

```
            sp++;
            break;
        case ADD:
            s [sp-1] = new Integer (((Integer) s [sp-1]).intValue ()
                                    + ((Integer) s [sp]).intValue ());
            sp--;
            break;
        case MKNIL:
            s [sp+1] = new Nil ();
            sp++;
            break;
```

2.1.4 Code improvement (20pts)

The version of the interpreter you just completed is written essentially as corresponding C code might be written. Redesign and reimplement the interpreter, fully adhering to the principles and practices of object-oriented programming. Put it in a package named `edu.unm.cs.cs351.fall2004.`*yourusername*`.betterliststackmachine`.

You should be able to invoke the interpreter as follows:

yourfavoritejvm `edu.unm.cs.cs351.fall2004.`*yourusername*`.betterliststack\`
`machine.ListStackMachine add3.lsm 0 5`

2.2 Analysis and discussion

Having implemented the interpreter, answer the following questions.

2.2.1 (5pts)

Manually execute the code below on the stack machine. Show the stack contents after every instruction and show what is printed on the console:

```
0: CST 100
2: CST 11
4: ADD
5: MKNIL
6: MKCONS
7: PRINT
8: STOP
```

2.2.2 (5pts)

Write stack machine code to create and print the list `Cons(111,Cons(222,Nil))`.

2.2.3 (5pts)

The instructions `CALL` and `RET` can be used to implement simple functions. What does the following stack machine program do, assuming that the integer 42 is on the stack top when execution is started at instruction address 0?

```
 0: CALL 4
 2: PRINT
 3: STOP
 4: DUP
 5: DUP
 6: MKNIL
 7: MKCONS
 8: MKCONS
 9: MKCONS
10: RET
```

Show the contents of the stack after each instruction, in the order in which the instructions are executed.

2.2.4 (5pts)

Write a stack machine program that, given that integer $n \geq 0$ is on the stack top, builds and prints an n-element list of the form `Cons(`n`,Cons(`$n-1$`,...,Cons(1,Nil),...))`. Provide both an iterative and a recursive solution.

2.2.5 (5pts)

The instruction `LISTCASE` can be used to test whether the list on the stack top is `Nil` or a `Cons`. If the stack top element is `Nil`, execution simply continues with the next instruction. If the stack top element is `Cons(`i`,`t`)`, then the integer i and the list tail t are unpacked on the stack and execution continues at address a.

Consider the following stack machine code fragment:

```
21: LISTCASE 27
23: CST 999
25: GOTO 28
27: POP
28: PRINT
```

Assume that the list `Cons(111,Cons(222,Nil))` is on the stack top when the function at address 21 is called (by `CALL 21`). What is on the stack top, and is therefore printed, when control reaches instruction 28? What is printed if the empty list is on the stack top when `CALL 21` is executed?

2.2.6 (5pts)

Write a stack machine function that, given that a list is on the stack top, computes the sum of the list elements. Provide both an iterative and a recursive solution.

2.2.7 (5pts)

For some programs and some initial states of the machine, the machine will crash. Write some programs that cause the machine to crash. What kinds of conditions cause a crash?

We would like to know for certain if a given stack machine program will crash when run. If we are given a program, an initial program counter, and an initial stack, can we prove (without actually running the program) that the program will crash, or, more usefully, that it will not crash? What do you think?

2.2.8 (5pts)

Extend the machine with a new instruction `GETVAR` *i* that reads the contents of the *i*th stack element below the stack top and pushes that value onto the stack. Since the stack top is the 0th element below the stack top, `GETVAR` 0 should be equivalent to `DUP`.

2.2.9 (5pts)

Use the new `GETVAR` instruction to write a stack machine function that can create a list that contains n copies of v, like this:

$$\texttt{Cons(}v\texttt{,Cons(}v\texttt{,...,Cons(}v\texttt{,Nil)...))}$$

The function should be called with a stack of the form $n : v : s$, that is, with n on the stack top and v below it, and should return with a stack of the form $w : s$ where w is an n-element list all of whose elements are v. Provide both an iterative and a recursive solution.

2.2.10 (5pts)

Use the new `GETVAR` instruction to write a stack machine function that, given that integer $n \geq 0$ is on the stack top, builds and prints an n-element list of the form

$$\texttt{Cons(}2n\texttt{,Cons(}2n-2\texttt{,...,Cons(2,Nil),...))}.$$

Provide both an iterative and a recursive solution.

2.2.11 (5pts)

Without using the `GETVAR` instruction, write a stack machine function that, given that integer $n \geq 0$ is on the stack top, builds and prints an n-element list of the form

$$\texttt{Cons(}2n\texttt{,Cons(}2n-2\texttt{,...,Cons(2,Nil),...))}.$$

Provide both an iterative and a recursive solution.

How to turn in

For the Java code, follow the pattern of Homework 1. For the analysis and discussion, place your answers in a file `answers.txt` or `answers.ps`. For each exercise involving machine code, test the machine code you wrote by running it in your interpreter. Your machine code tests should be just as thorough as your Java code tests!