Lecture 17: 0xCAFEBABE (Virtual Machines)

Java™: Programming Language

“A simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, and dynamic language.” [Sun95]

Java: int is 32 bits
C: int is >= 16 bits
Java Virtual Machine

- Small and simple to implement
- All VMs will run all programs the same way
- “Secure”

Java Ring (1998)

Implementing the JavaVM

- load class into memory
- set the instruction pointer to point to the beginning of main
- while not finished:
  - fetch the next instruction
  - execute that instruction

Some other issues we will talk about Wednesday:
- Verification – need to check byte codes satisfy security policy

Java Byte Codes

- Stack-based virtual machine
- Small instruction set: 202 instructions (all are 1 byte opcode + operands)
  - Intel x86: ~280 instructions (1 to 17 bytes long!)
- Memory is typed
- Every Java class file begins with magic number 3405691582 = 0xCAFEBABE in hex

Stack-Based Computation

- **push** – put something on the top of the stack
- **pop** – get and remove the top of the stack

Stack

<table>
<thead>
<tr>
<th>push 2</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>push 3</td>
<td>3</td>
</tr>
<tr>
<td>add</td>
<td>5</td>
</tr>
</tbody>
</table>

Does 2 pops, pushes sum

Some JVML Instructions

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Mnemonic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>nop</td>
<td>Does nothing</td>
</tr>
<tr>
<td>1</td>
<td>aconst_null</td>
<td>Push null on the stack</td>
</tr>
<tr>
<td>3</td>
<td>icost_0</td>
<td>Push int 0 on the stack</td>
</tr>
<tr>
<td>4</td>
<td>icost_1</td>
<td>Push int 1 on the stack</td>
</tr>
</tbody>
</table>

Why do we need both aconst_null and icost_0?

Load Constant

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Mnemonic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>ldc &lt;value&gt;</td>
<td>Push a one-word (4 bytes) constant onto the stack</td>
</tr>
</tbody>
</table>

Constant may be an int, float or String

- ldc "Hello"
- ldc 216

The String is really a reference to an entry in the string constant table!
### Arithmetic

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Mnemonic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>96</td>
<td>iadd</td>
<td>Pops two integers from the stack and pushes their sum</td>
</tr>
<tr>
<td>97</td>
<td>iadd</td>
<td>Pops two long integers from the stack and pushes their sum</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>106</td>
<td>fmul</td>
<td>Pops two floats from the stack and pushes their product</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>119</td>
<td>dneg</td>
<td>Pops a double from the stack, and pushes its negation</td>
</tr>
</tbody>
</table>

### Java Byte Code Instructions

- 0: nop
- 1-20: putting constants on the stack
- 96-119: arithmetic on ints, longs, floats, doubles

- 1 byte opcode: 146 left
- What other kinds of instructions do we need?

### Other Instruction Classes

- Control Flow (~20 instructions)
  - if, goto, return
- Loading and Storing Variables (65 instructions)
- Method Calls (4 instructions)
- Creating objects (1 instruction)
- Using object fields (4 instructions)
- Arrays (3 instructions)

### Control Flow

- ifeq <label>
  - Pop an int off the stack. If it is zero, jump to the label. Otherwise, continue normally.
- if_icmple <label>
  - Pop two ints off the stack. If the second one is <= the first one, jump to the label. Otherwise, continue normally.

### Referencing Memory

- iload <varnum>
  - Pushes the int in local variable <varnum> (1 bytes) on the stack
- istore <varnum>
  - Pops the int on the top of the stack and stores it in local variable <varnum>
Referencing Example

Method void main(java.lang.String[])
0 iconst_2
1 istore_1
2 iconst_3
3 istore_2
4 iload_1
5 iload_2
6 iadd
7 istore_3
8 getstatic #2 <Field java.io.PrintStream err>
11 new #3 <Class java.lang.StringBuffer>
14 dup
15 invokespecial #4 <Method java.lang.StringBuffer()>
20 invokevirtual #5 <Method java.lang.StringBuffer append(java.lang.String)>
23 iload_3
24 invokevirtual #6 <Method java.lang.StringBuffer toString()>
27 invokevirtual #7 <Method void println(java.lang.String)>
30 return

Example

```java
class Sample1 { 
    static public void main (String args[]) { 
        System.err.println("Hello!"); 
        System.exit (1); 
    } 
}
```

Charge

- PS5: Due Wednesday
- Question 2 is a "tricky" question
- Focus on correctness: implement something simple for questions 7-9 first
- You can describe clever designs for question 6, simplicity should be the main factor in deciding what to implement