CS216: Program and Data Representation
University of Virginia Computer Science
Spring 2006 David Evans

Lecture 17: 0xCAFEBABE (Virtual Machines)

http://www.cs.virginia.edu/cs216

CS216 Roadmap

Data Representation Program Representation

Real World Problems

"Hello"
['H','i',\0]
0x42381a,
3.14,
'x'

01001010

Objects
Arrays
Addresses,
Numbers,
Characters

Bits

Python code
C code
JVML
x86

High-level language
Low-level language
Virtual Machine language
Assembly

Real World Physics

Java Virtual Machine

JVML is a detour:
everything else we have seen is part of running the PS1 Python program

Java™: Programming Language

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

Wednesday's class

Properties of language implementations, not languages

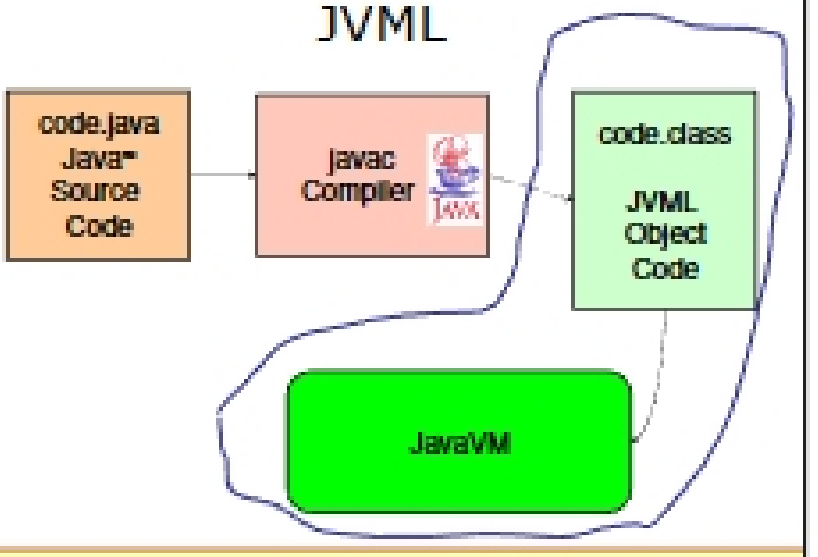
Java™: Programming Language

compared to C++, not to C sort of

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

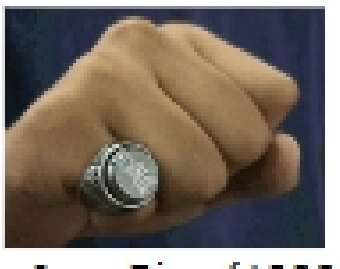
Java: int is 32 bits
C: int is >= 16 bits

JVML



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

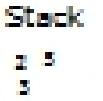
- Stack-based virtual machine
- Small instruction set: 202 instructions (all are 1 byte opcode + operands)
  - Intel x86: ~280 instructions (1 to 17 bytes long!)
- Memory is typed
- Every Java class file begins with magic number 3405691582
  = **0xCAFEBABE** in hex

Stack-Based Computation

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

```
push 2
push 3
add
```

Stack



Does 2 pops, pushes sum

Some JVML Instructions

Opcode	Mnemonic	Description
0	nop	Does nothing
1	aconst_null	Push null on the stack
3	iconst_0	Push int 0 on the stack
4	iconst_1	Push int 1 on the stack
...		

Why do we need both aconst_null and iconst_0?

Load Constant

Opcode	Mnemonic	Description
18	ldc <value>	Push a one-word (4 bytes) constant onto the stack

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

Opcode	Mnemonic	Description
96	iadd	Pops two integers from the stack and pushes their sum

```
iconst_2
iconst_3
iadd
```


Arithmetic

Opcode	Mnemonic	Description
96	iadd	Pops two integers from the stack and pushes their sum
97	ladd	Pops two long integers from the stack and pushes their sum
...		
106	fmul	Pops two floats from the stack and pushes their product
...		
119	dneg	Pops a double from the stack, and pushes its negation

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>