Java Virtual Machine

Operating System

Hardware Platform
Multiple Platforms
What is the “Machine”? [James Smith, VEE’05]
Java
A few words of history

• **1985** - James Gosling designs NeWS - Network/extensible Window System for Sun Microsystems. NeWS is a portable dynamically-typed object-oriented language, with garbage collection, a portable code format, dynamic loading.

• **1992** - James Gosling designs Oak - a statically-typed object-oriented language for embedded devices. Oak has inheritance, garbage-collection, a portable intermediate representation, type-safe, but a syntax like C.

• **1995** - Java, aka Oak, is introduced. It runs on devices with >20MB of main memory.
From Source Code to Running Program

Java Program

class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}

HelloWorldApp.java

Compiler

JVM

Win32

UNIX

MacOS
Java Virtualizes…

• ... locale
• ... threading
• ... endianness
• ... memory models
• ... operating system
• ... program extension
• ... data representation
• ... memory management

A Main Reference Source

The Java™ Virtual Machine Specification (2nd Ed) by Tim Lindholm & Frank Yellin
Addison-Wesley, 1999

The book is on-line and available for download:
http://java.sun.com/docs/books/vmspec/
Implementing programming languages

Usual Programming Language Implementation

Another Programming Language Implementation

And Another Implementation
An Overview

• Source code is translated into an intermediate representation, (IR)
• The IR can be processed in these different ways:
  1. compile-time (static) translation to machine code
  2. emulation of the IR using an interpreter
  3. run-time (dynamic) translation to machine code = JIT (Just-In-Time) compiling

What is IR?

IR is code for an idealized computer, a virtual machine.
A Small IR and its Interpreter (1/3)

We need a representation scheme for the bytecode. A simple one is:

- to use one byte for an opcode,
- four bytes for the operand of LDI,
- two bytes for the operands of LD, ST, JMP and JMPF.

As well as 0 for STOP, we will use this opcode numbering:

<table>
<thead>
<tr>
<th>LDI</th>
<th>LD</th>
<th>ST</th>
<th>ADD</th>
<th>SUB</th>
<th>EQ</th>
<th>NE</th>
<th>GT</th>
<th>JMP</th>
<th>JMPF</th>
<th>READ</th>
<th>WRITE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>

The order of the bytes in the integer operands is important. We will use \textit{big-endian} order.
It emulates the fetch/decode/execute stages of a computer.

```c
for( ; ; ) {
    opcode = code[pc++];
    switch(opcode) {
    case LDI:
        val = fetch4(pc);  pc += 4;
        push(val);
        break;
    case LD:
        num = fetch2(pc);  pc += 2;
        push( variable[num] );
        break;
    ...
    case SUB:
        right = pop();  left = pop();
        push( right-left );
    ...
```
case JMP:
    pc = fetch2(pc);
    break;

case JMPF:
    val = pop();
    if (val)
        pc += 2;
    else
        pc = fetch2(pc);
    break;

...
The Java Classfile

[Diagram of Java classfile structure with labels such as Class Header Info, Constant Pool, Interfaces, Fields, Misc. Attributes, Method, Method #1, Local Variable Table, Line Number Table, Exception Table, Bytecode Array, Method Header, Method #1]
The internal runtime structure of the JVM consists of:

- One: (i.e. shared by all threads)
  - method area
  - heap
- For each thread, a:
  - program counter (pointing into the method area)
  - Java stack
  - native method stack (system dependent)
Runtime structure

16

- class loader subsystem
- method area
- heap
- Java stacks
- pc registers
- native method stacks
- execution engine
- native method interface
- native method libraries
- class files

runtime data areas
Datatypes

**Datatypes of the JVM (Venners 5-4)**

**Primitive Types**
- Byte
- Short
- Char
- Int
- Long
- Float
- Double

**Reference Types**
- returnValue
- reference
- Class types
- Interface types
- Array types

**Numeric Types**
- F.P. Types
- Integral Types

**two words**
```java
int bar(int i) {
    try {
        if (i == 3) return this.foo();
    } finally {
        this.ladida();
    }
    return i;
}
```

### Java Bytecode

```
01 iload_1 // Push i
02 iconst_3 // Push 3
03 if_icmpne 10 // Goto 10 if i does not equal 3
// Then case of if statement
04 aload_0 // Push this
05 invokevirtual foo // Call this.foo
06 istore_2 // Save result of this.foo()
07 jsr 13 // Do finally block before returning
08 iload_2 // Recall result from this.foo()
09 ireturn // Return result of this.foo()
// Else case of if statement
10 jsr 13 // Do finally block before leaving try
// Return statement following try statement
11 iload_1 // Push i
12 ireturn // Return i
// finally block
13 astore_3 // Save return address in variable 3
14 aload_0 // Push this
15 invokevirtual ladida // Call this.ladida()
16 ret 3 // Return to address saved on line 13
// Exception handler for try body
17 astore_2 // Save exception
18 jsr 13 // Do finally block
19 aload_2 // Recall exception
20 athrow // Rethrow exception
// Exception handler for finally body
21 athrow // Rethrow exception
```
Virtualizing Memory

- Memory is a set of objects with fields, methods and a class + local variables of a method
- Memory is read by accessing a field or local variable
- Memory is modified by writing to a field or local variable
- Location and size of data are not exposed
- Memory allocation is done by call in new

public class Main {
    static public
    void main(String[] a){
        Cell c1, c2 = null;
        while (true) {
            c1 = new Cell();
            c2 = c1;
        }
    }
}

class Cell {Cell next; }
Virtualizing Memory

- Memory is a set of objects with fields, methods and a class + local variables of a method
- Memory is read by accessing a field or local variable
- Memory is modified by writing to a field or local variable
- Location and size of data are not exposed
- Memory allocation is done by call in new

Question:
- Does main() terminate?

```java
public class Main {
    static public void main(String[] a){
        Cell c1, c2 = null;
        while (true) {
            c1 = new Cell();
            c1.next = c2;
            c2 = c1;
        }
    }
}

class Cell { Cell next; }
```
Virtualizing Memory

The semantics of `new` is as follows:

- Allocate space for the object’s fields and metadata fields
- Initialize the metadata fields
- Set all fields to null/zero/false
- Invoke the user defined constructor method
Virtualizing memory

- Garbage collection is the technology that gives the illusion of infinite resources
- Garbage collection or GC is implemented by the programming language with the help of the compiler
  - Though for some well-behaved C programs it is possible to link a special library that provides most of the benefits of GC

- Question:
  - How does GC work?
Garbage Collection

Phases

- Mutation
- Stop-the-world
- Root scanning
- Marking
- Sweeping
- Compaction
Garbage Collection

Phases
- Mutation
- Stop-the-world
- Root scanning
- Marking
- Sweeping
- Compaction
Garbage Collection

Phases
- Mutation
- Stop-the-world
- Root scanning
- Marking
- Sweeping
- Compaction
Garbage Collection

Phases
- Mutation
- Stop-the-world
- Root scanning
- Marking
- Sweeping
- Compaction
Garbage Collection

Phases
- Mutation
- Stop-the-world
- Root scanning
- Marking
- Sweeping
- Compaction
Garbage Collection

Phases
- Mutation
- Stop-the-world
- Root scanning
- Marking
- Sweeping
- Compaction
Garbage Collection

Phases
- Mutation
- Stop-the-world
- Root scanning
- Marking
- Sweeping
- Compaction
Garbage Collection

Phases
- Mutation
- Stop-the-world
- Root scanning
- Marking
- Sweeping
- Compaction
Conclusion

- The Java Virtual Machine provides a level of abstract over the hardware and the operating system that hides their specificities.
- Java source code is compiled to bytecode (javac).
- Bytecode is either interpreted or JIT-ed to execute by the JVM (java).