02 B The Java Virtual Machine

CS1102S: Data Structures and Algorithms

Martin Henz

January 22, 2010

Generated on Friday 22nd January, 2010, 09:46

Trade-off Between Interpretation and Compilation

- Compilation results in machine code, executed efficiently on the target hardware
- Drawbacks: security, portability
- Interpretation interprets one program piece at a time, using a portable and secure runtime system
- Drawbacks: slow execution

Java's Roots

- James Gosling and Bill Joy developed Oak at FirstPerson Inc, a Sun subsidiary
- Oak was intended for programming devices
- Sun realized the potential of Java for the web and tried to establish it for client-side computing ("applets")
- Security features, clean design and powerful libraries made Java attractive for server-side processing

What is Java?

- Java is not just the language
- Java stands for a combination of techniques and concepts:
  - Language syntax
  - Compiler
  - Java Virtual Machine
  - Java's standard APIs

Features of Java

Portable : runtime system needs to be ported to hardware platforms. Once runtime is available, any 100% pure Java application can run on the platform. **Write Once, Run Anywhere**

Efficient : Byte-code interpretation provides good efficiency trade-off. Classes are loaded on-demand, resulting in fast start-up times of applications

Secure : Extensive security model allows targeting various security requirements.

However, the Java license states: "not designed or intended for use in the design, construction, operation or maintenance of any nuclear facility."

Primary Design Goals

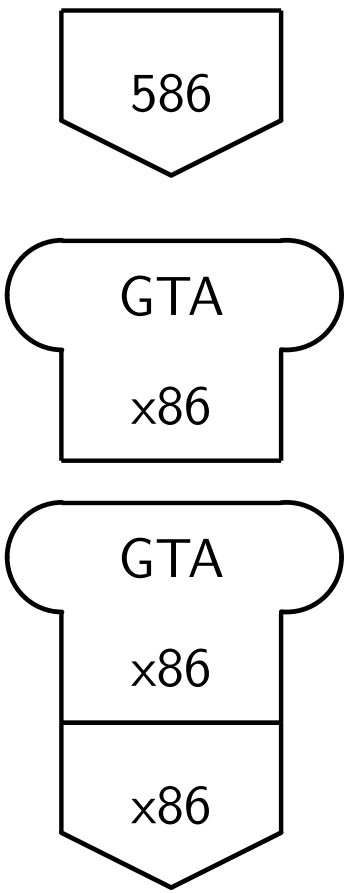
- Object-oriented
- Portable
- Supporting network computing
- Secure
- Easy to use/program

1. Motivation for JVM

2. Virtual Machine Overview
   - A Virtual Processor
   - Runtime System
   - Virtual Machine Features and Drawbacks
   - Security and Efficiency

3. A Quick Tour of the JVM

T-Diagrams for Processor and Machine Code



x86 Processor

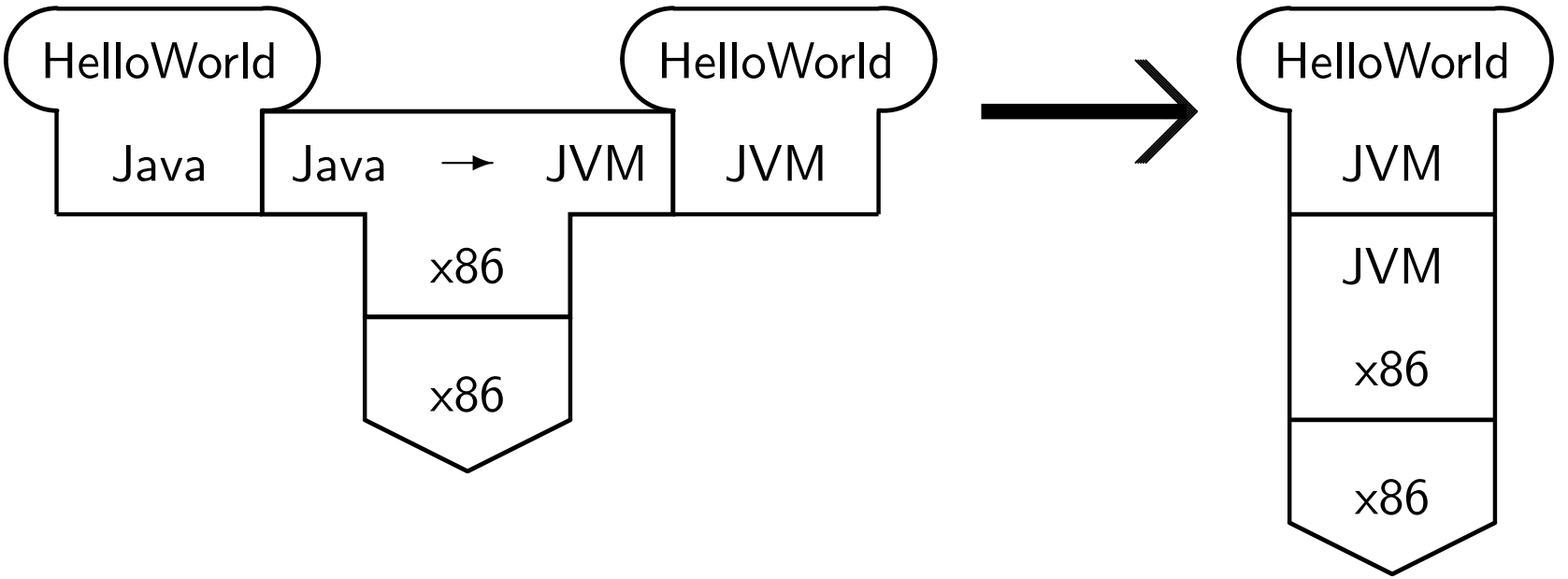
Program “GTA” (x86 code)

“GTA” running on x86

A Virtual Processor

- Java programs are not executed directly, or compiled to native machine code, but compiled to virtual machine code
- The virtual machine code is interpreted instruction-by-instruction by a program called the Java Virtual Machine (JVM)
- The JVM is a program that read in virtual machine code in form of class files (often dynamically) and iterprets the instructions one after the other

T-Diagrams for Execution of Jasmin Code



Compiling “HelloWorld” from Java to JVM code, and running the JVM code on a JVM running on an x86

“Physical” Processors

- Processor and main memory (often caches)
- Instruction counter points to memory location of instruction
- Registers keep track of intermediate computing results
- Instructions can write computing results to memory

Example

A typical x86 instruction:

```
mov AX, 10;  store the integer 10 in register AX
```

Corresponding JVM instruction:

```
bipush 10;   push integer 10 onto operand stack
istore_1;    store top of stack to variable 1
```

Another Example

Adding two integer on x86:

```
mov AX, 5;   put 5 into AX
mov BX, 10;  put 10 into BX
add;         add numbers, result in AX
```

Corresponding JVM instruction:

```
bipush 5;    push integer 5 onto operand stack
bipush 10;   push integer 10 onto operand stack
iadd;        add two numbers on top of stack
istore_1;    store top of stack to variable 1
```

JVM Instructions

Arithmetic operations : iadd, lsub, frem

Logical operations : iand, lor , ishl

Numeric conversions : int2short , f2l

Pushing constants : bipush, iconst−0

Stack manipulation : pop,dup

Flow control : goto, ifne

Managing local vars : astore, iload

Arrays : aastore,aaload

Objects : getfield , invokevirtual

Components of the Runtime System

Execution engine : virtual processor for executing byte code

Memory manager : allocating memory for objects and arrays and performing garbage collection

Error and exception manager : dealing with catching and throwing exceptions

Native methods : handling methods provided in native machine code

Thread interface : handling threads

Class loader : (dynamically) loading Java classes from class files

Security manager : verifying that classes are safe and resources are accessed in compliance with security policies

Main Advantages of Virtual Machines

Portability : Once compiled, the code runs on any architecture on which the virtual machine is available

Security : It is easy to handle security requirements in software, by programming them within the virtual machine

Interpretation (Tcl in Tcl)

```
proc interpreter() {
    while {1} {
        set line [get_next_line]
        set command [get_first_word $line]
        set arguments [all_but_first_word $line]
        if {$command == "puts"} {
            puts $arguments;
        }
        if ...
    }
}
```

Compilation (very simplified)

```
public static void compile(String lines[],outfile)
    Scanner scanner = new Scanner(lines);
    Parser parser = new Parser(scanner);
    SyntaxTree t = parser.getTree();
    Code c = compiler.compile(t);
    writeCodeToFile(outfile);
}
```

Virtual Machine (very simplified)

```
public static void execute(byte bytecode[]) {
    int pc = 0; // program counter
    while (true) {
        int opcode = bytecode[pc];
        switch (opcode) {
            case OPC_PUTS:
            ...
            case OPC_EXIT:
                return;
    }
}
```

Security

- Virtual machine can check code before it gets executed (byte code verification)
- Virtual machine can check arguments of operations before they are used (runtime checks)
- Particular components such as memory system are in full control of the virtual machine; any level of security can be achieved, if desired

Efficiency

- Slower than native code
- Much faster than interpretation
- Modern JVM implementations do just-in-time compilation
- If needed, Java can be compiled to native code

Disadvantages

- Complex instruction set to achieve security
- Hard to extend machine
- No parse tree! Compare Scheme's quoting and eval

1. Motivation for JVM
2. Virtual Machine Overview
3. A Quick Tour of the JVM
   - Hello World!
   - HelloWorld in Jasmin

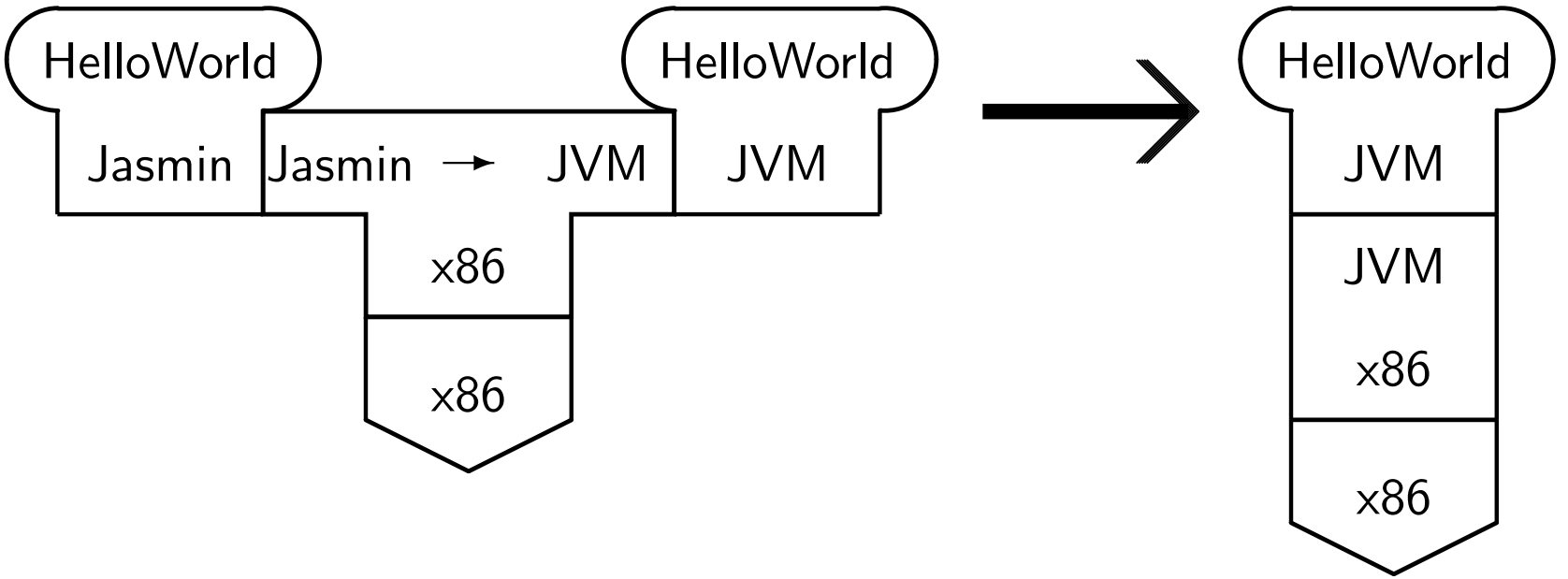
Example

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

An Assembler for JVM Code

- Jasmin is a compiler from a text format of JVM code to actual class files
- Provided free of change together with book “Java Virtual Machine”

T-Diagrams for Execution via JVM



Compiling “HelloWorld” from Jasmin to JVM code, and running the JVM code on a JVM running on an x86

What does a `.class` file look like?

```
^@^C^@-^@^N
^@^M^@^G^A^@^Pjava/lang/Object^A^@
SourceFile^A^@^F<init>^G^@^L^A^@^Dmain^L^@^D^@
^A^@^DCode^A^@^V([Ljava/lang/String;)V^A^@^C()V^A^@
HelloWorld^G^@^B^@!^@^E^@^M^@^@^@^@^@^B^@^A^@^D^@
^@^A^^@^@^@^Q^@^A^@^A^@^@^@^E*<B7>^@^A<B1>^@^@^@^@^
^B^@^B^@^@^@^Q^P^B:^@^P^C:^A^Y^@^Y^A:^@:^A<B1>^@^@^
```

Hex Dump of `HelloWorld.class` File

```
r-98-183-18-172:~/Documents/jasmin-2.3 henz$ xxd HelloWorld.class
0000000: cafe babe 0003 002d 000e 0a00 0d00 0701  .......-........
0000010: 0010 6a61 7661 2f6c 616e 672f 4f62 6a65  ..java/lang/Obje
0000020: 6374 0100 0a53 6f75 7263 6546 696c 6501  ct...SourceFile.
0000030: 0006 3c69 6e69 743e 0700 0c01 0004 6d61  ..<init>......ma
0000040: 696e 0c00 0400 0a01 0004 436f 6465 0100  in........Code..
0000050: 1628 5b4c 6a61 7661 2f6c 616e 672f 5374  .([Ljava/lang/St
0000060: 7269 6e67 3b29 5601 0003 2829 5601 000c  ring;)V...()V...
0000070: 4865 6c6c 6f57 6f72 6c64 2e6a 0100 0a48  HelloWorld.j...H
0000080: 656c 6c6f 576f 726c 6407 0002 0021 0005  elloWorld....!..
0000090: 000d 0000 0000 0002 0001 0004 000a 0001  ................
00000a0: 0008 0000 0011 0001 0001 0000 0005 2ab7  ..............*.
00000b0: 0001 b100 0000 0000 0900 0600 0900 0100  ................
00000c0: 0800 0000 1d00 0200 0200 0000 1110 023a  ...............:
00000d0: 0010 033a 0119 0019 013a 003a 01b1 0000  ...:.....:.:....
00000e0: 0000 0001 0003 0000 0002 000b            ............
r-98-183-18-172:~/Documents/jasmin-2.3 henz$
```

Readable Text from `.class` File

- Is there a way to get the JVM code for a given `.class` file?
- Yes, through a disassembler!

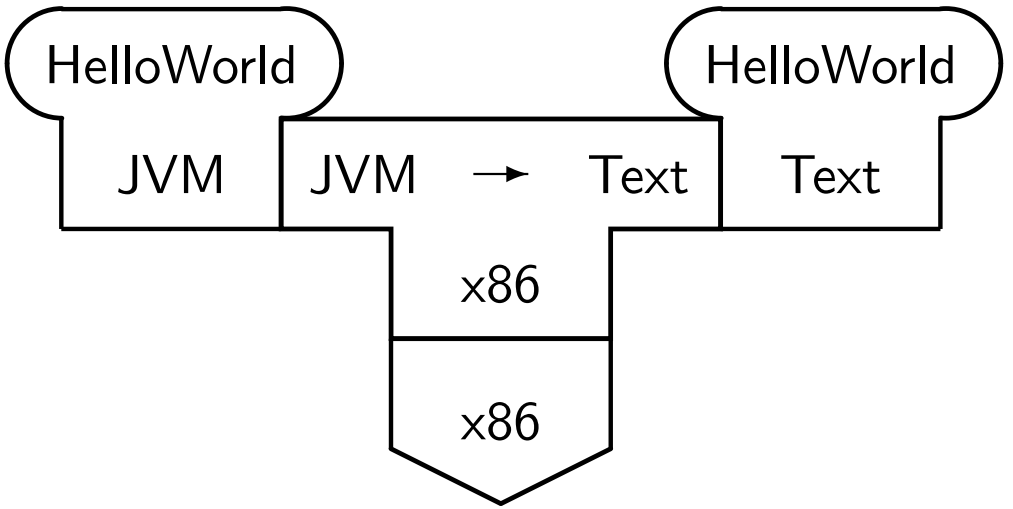
```
% javap -c HelloWorld
```

- This instruction disassembles the `HelloWorld.class` file into a textual representation of the JVM code (similar to Jasmin code)

Output of Disassembler

```
Compiled from "HelloWorld.j"
public class HelloWorld extends java.lang.Object
public HelloWorld();
  Code:
   0:   aload_0
   1:   invokespecial   #1; //Method java/lang/Object."<init>":()V
   4:   return
public static void main(java.lang.String[]);
  Code:
   0:   bipush  2
   2:   astore  0
   4:   bipush  3
   6:   astore  1
   8:   aload   0
   10:  aload   1
   12:  astore  0
   14:  astore  1
   16:  return
```

T-Diagram for Disassembly of a `.class` File



The “Text” resulting from the disassembly is similar to the Jasmin format and described in the Java Virtual Machine Specification

HelloWorld in Jasmin

```
.class public HelloWorld
    .super java/lang/Object

    ; standard initializer
    ; (calls java.lang.Object's initializer)

    .method public <init>()V
       aload_0
       invokespecial java/lang/Object/<init>()V
       return
    .end method
```

HelloWorld in Jasmin

```
; main() - prints out Hello World
;
.method public static main([Ljava/lang/String;)V
  .limit stack 2   ; up to two items can be pushed

  ; push System.out onto the stack
  getstatic
    java/lang/System/out Ljava/io/PrintStream;
  ...
```

HelloWorld in Jasmin

```
  ...
  ; push a string onto the stack
  ldc "Hello World!"

  ; call the PrintStream.println() method.
  invokevirtual
  java/io/PrintStream/println(Ljava/lang/String;)V

  ; done
  return
.end method
```

Puzzler Of The Week: Minute By Minute

```
public class Clock {
  public static void main(String[] args) {
    int minutes = 0;
    for (int ms = 0; ms < 60*60*1000; ms++)
      if (ms % 60*1000 == 0)
        minutes++;
    System.out.println(minutes);
} }
```

Question

What does this program print?

Puzzler Of The Week: Minute By Minute

```
public class Clock {
  public static void main(String[] args) {
    int minutes = 0;
    for (int ms = 0; ms < 60*60*1000; ms++)
      if (ms % 60*1000 == 0)
        minutes++;
    System.out.println(minutes);
} }
```

Question

What does this program print?

Hint

It's not 60!

Next Few Weeks

- Monday 25/1: Labs (playing with Java and JVM)
- Wednesday 27/1 lecture: Lists, Stacks, Queues I
- Thursday 28/1: tutorials: Solutions for Assignment 1
- Friady 29/1: Lists, Stacks, Queues II
- Assignment 2: Out on Wednesday 27/1, due Wednesday 2/2 6pm
- ...
- 12/2: Midterm 1 (Java; Algorithm Analysis; Lists/Stacks/Queues; Trees)