CMPE 152: Compiler Design

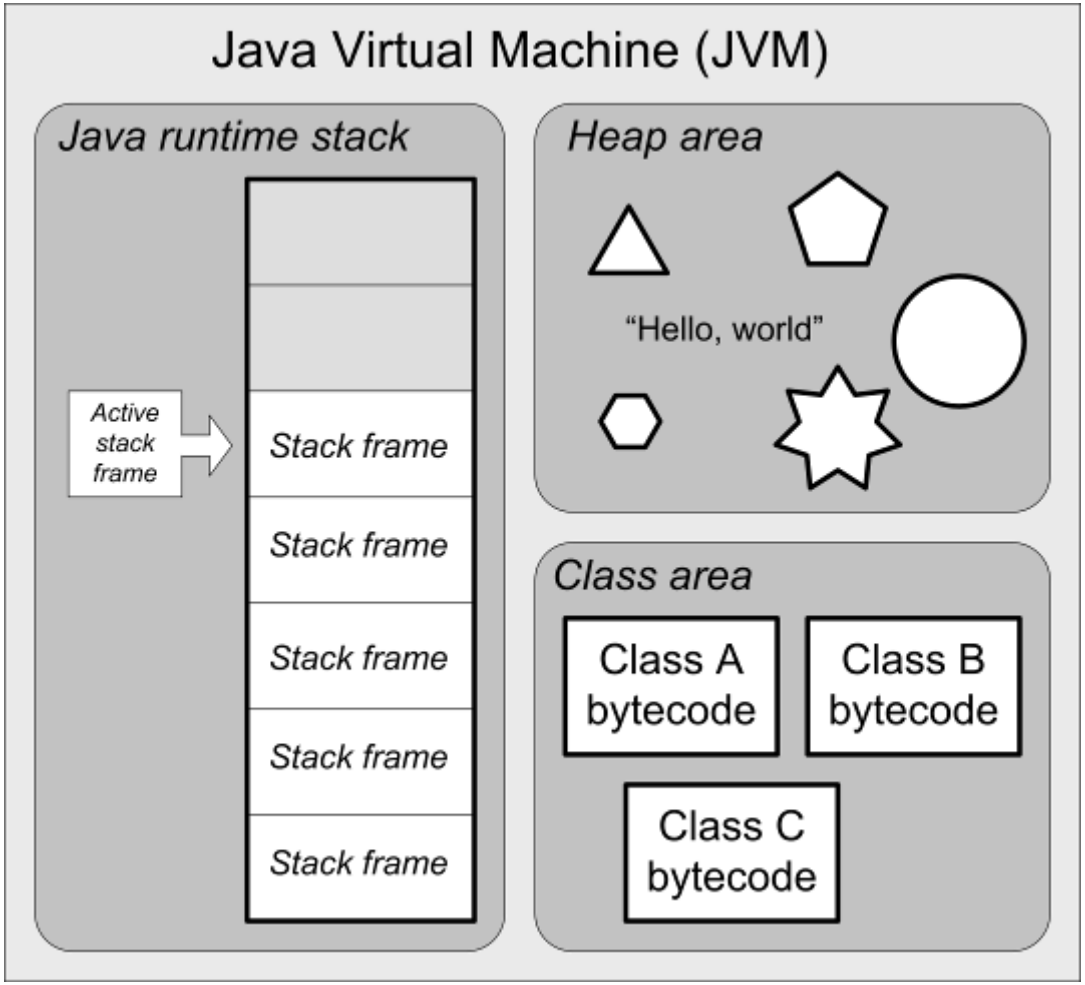
October 31 Lab

Department of Computer Engineering
San Jose State University

Fall 2017
Instructor: Ron Mak

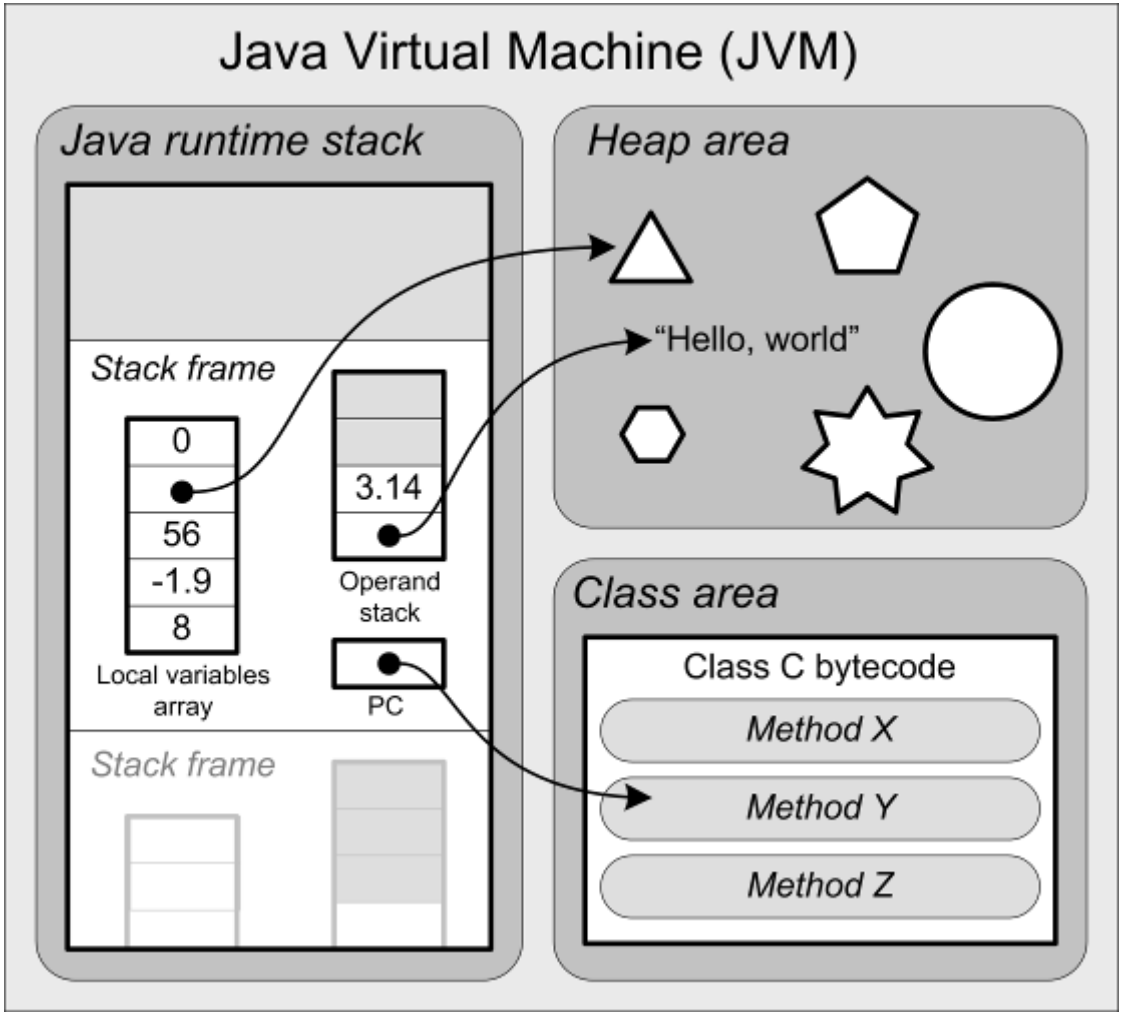
www.cs.sjsu.edu/~mak

Java Virtual Machine (JVM) Architecture



- Java stack
  - runtime stack
- Heap area
  - dynamically allocated objects
  - automatic garbage collection
- Class area
  - code for methods
  - constants pool
- Native method stacks
  - support native methods, e.g., written in C
  - (not shown)

Java Virtual Machine Architecture, *cont'd*



- The runtime stack contains stack frames.
  - Stack frame = activation record.
- Each stack frame contains:
  - local variables array
  - operand stack
  - program counter (PC)

What is missing in the JVM that we had in our Pascal interpreter?

The JVM's Java Runtime Stack

- Each method invocation pushes a stack frame.
- Equivalent to the activation record of our Pascal interpreter.
- The stack frame currently on top of the runtime stack is the active stack frame.
- A stack frame is popped off when the method returns, possibly leaving behind a return value on top of the stack.

Stack Frame Contents

- Operand stack
  - For doing computations.
- Local variables array
  - Equivalent to the memory map in our Pascal interpreter's activation record.
- Program counter (PC)
  - Keeps track of the currently executing instruction.

JVM Instructions

- Load and store values
- Arithmetic operations
- Type conversions
- Object creation and management
- Runtime stack management (push/pop values)
- Branching
- Method call and return
- Throwing exceptions
- Concurrency

Jasmin Assembler

- Download from:
  - http://jasmin.sourceforge.net/
- Site also includes:
  - User Guide
  - Instruction set
  - Sample programs

Example Jasmin Program

hello.j

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

.method public static main([Ljava/lang/String;)V
.limit stack  2
.limit locals 1

  getstatic     java/lang/System/out Ljava/io/PrintStream;
  ldc           "Hello, world!"
  invokevirtual java/io/PrintStream/println(Ljava/lang/String;)V
  return

.end method
```

- Assemble:
  - `java –jar jasmin.jar hello.j`
- Execute:
  - `java HelloWorld`

Jamin Assembly Instructions

- An Jasmin instruction consists of a mnemonic optionally followed by arguments.
  - Example:

```
aload 5     ; Push a reference to local variable #5
```

- Some instructions require operands on the operand stack.
  - Example:

```
iadd     ; Pop the two integer values on top of the
         ; stack, add them, and push the result
```

Jasmin Assembly Instructions, *cont'd*

- The JVM (and Jasmin) supports five basic data types:
  - int
  - long
  - float
  - double
  - reference
- Examples:

```
isub  ; integer subtraction
fmul  ; float multiplication
```

- Long and double values each requires two consecutive entries in the local variables array and two elements on the operand stack.

Letter	Type
a	reference
b	byte or boolean
c	char
d	double
f	float
i	int
l	long
s	short

Byte, boolean, char, and short are treated as ints on the operand stack and in the local variables array.

Loading Constants onto the Operand Stack

- Use the instructions **`ldc`** and **`ldc2_w`** (load constant and load double-word constant) to push constant values onto the **operand stack**.
  - Examples:

```
ldc      2
ldc      "Hello, world”
ldc      1.0
ldc2_w   1234567890L
ldc2_w   2.7182818284D
aconst_null  ; push null
```

Shortcuts for Loading Constants

- Special shortcuts for loading certain small constants x:

```
iconst_m1  ; Push int -1
iconst_x   ; Push int x, x = 0, 1, 2, 3, 4, or 5
lconst_x   ; Push long x, x = 0 or 1
fconst_x   ; Push float x, x = 0, 1, or 2
dconst_x   ; Push double x, x = 0 or 1

bipush x   ; Push byte x, -128 <= x <= 127
sipush x   ; Push short x, -32,768 <= x <= 32,767
```

- Shortcut instructions take up less memory and can execute faster.

Local Variables

- Local variables do not have names in Jasmin.
  - Fields of a class do have names, which we'll see later.
- Refer to a local variable by its slot number in the local variables array.
  - Example:

```
iload 5  ; Push the int value in local slot #5
```

Local Variables, *cont'd*

- Since each long and double value requires two consecutive slots, refer to it using the lower slot number.
  - Example:

```
lstore 3   ; Pop the long value
           ; from the top two stack elements
           ; and store it into local slots #3 and 4
```

Local Variables, *cont'd*

- Do not confuse constant values with slot numbers!
  - It depends on the instruction.
  - Examples:

```
bipush 14   ; push the constant value 14
iload  14   ; push the value in local slot #14
```

Local Variables, *cont'd*

- Local variables starting with slot #0 are automatically initialized to any method arguments.

```
public static double meth(int k, long m,
                          float x, String[][] s)
```

  - `k` ➔ local slot #0
    `m` ➔ local slot #1
    `x` ➔ local slot #3
    `s` ➔ local slot #4

What happened to slot #2?

  - Jasmin method signature:

```
.method public static meth(IJF[[Ljava/lang/String;)D
```

Load and Store Instructions

- In general:

```
iload n  ; push the int value in local slot #n
lload n  ; push the long value in local slot #n
fload n  ; push the float value in local slot #n
dload n  ; push the double value in local slot #n
aload n  ; push the reference in local slot #n
```

- Shortcut examples
  (for certain small values of n):

```
iload_0  ; push the int value in local slot #0
lload_2  ; push the long value in local slot #2
fload_1  ; push the float value in local slot #1
dload_3  ; push the double value in local slot #3
aload_2  ; push the reference in local slot #2
```

- Store instructions are similar.