Compiling Techniques

Lecture 10: Introduction to Java ByteCode

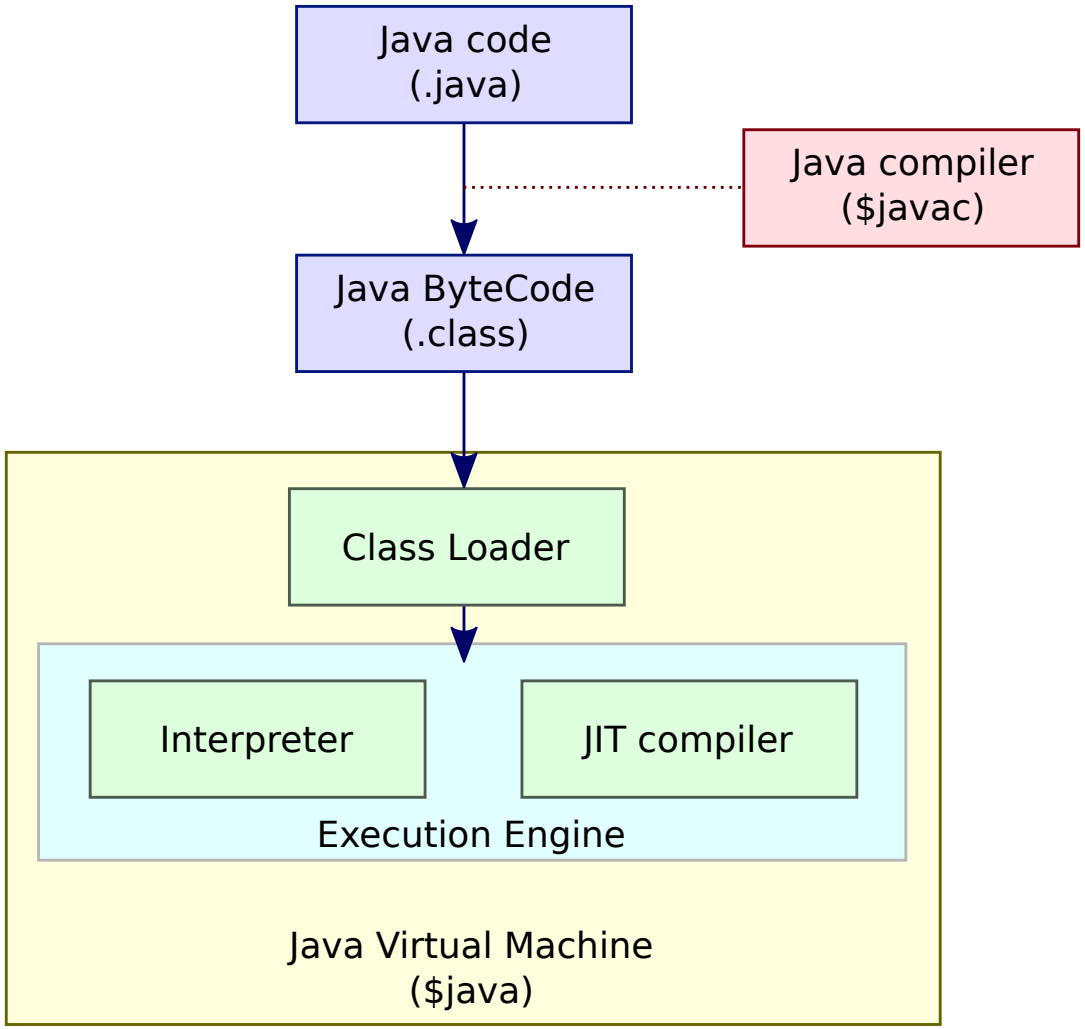
Christophe Dubach

10 November 2015

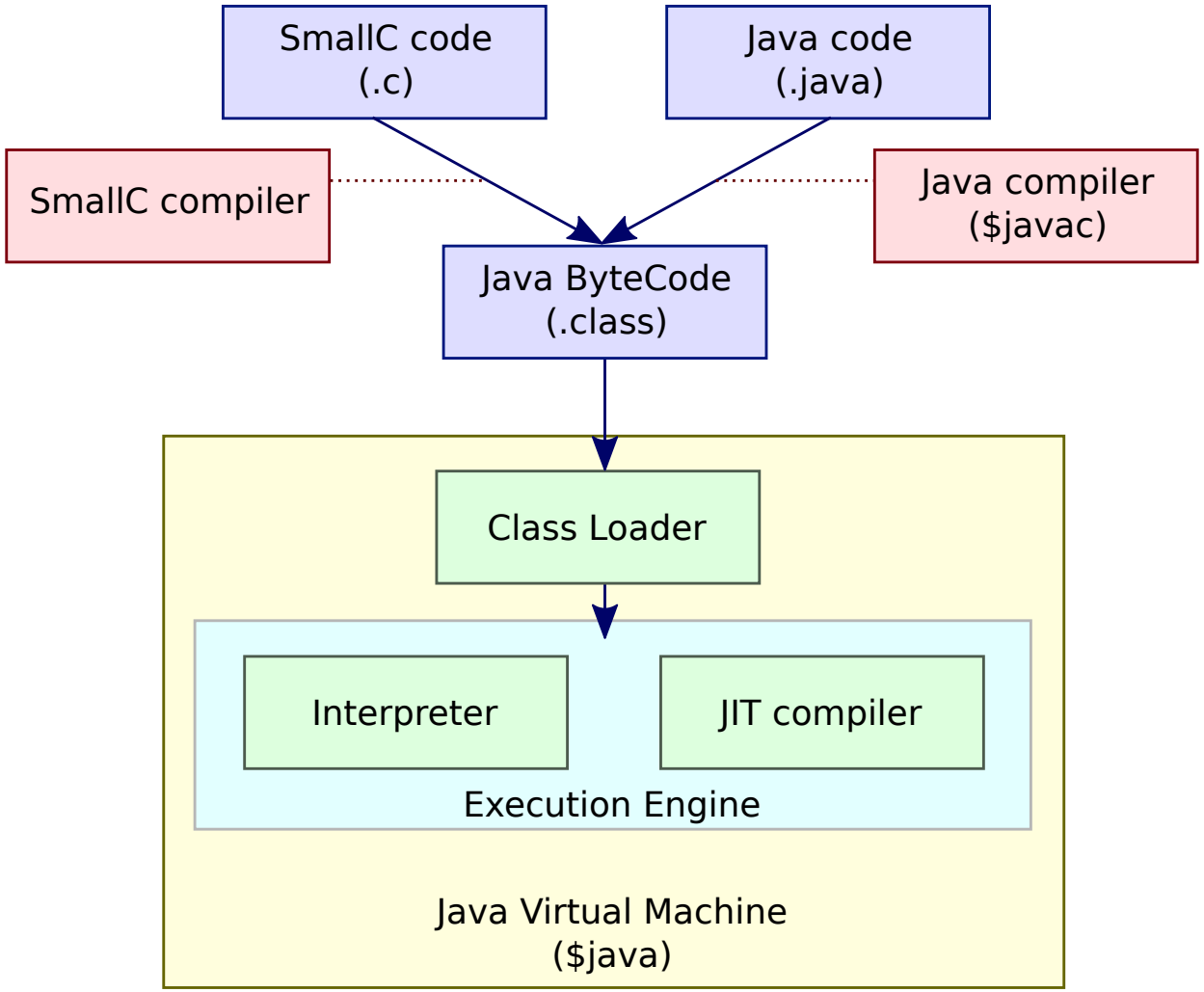
Coursework: Block and Procedure

Table of contents

From Java source code to the JVM



From SmallC source code to the JVM



- Java (or SmallC) compiler: compile the source code into Java ByteCode
- Classloader: loads the bytecode from class files into the Runtime
- Execution Engine: executes the ByteCode
  - Interpreter: interprets the ByteCode
  - JIT compiler: compiles the ByteCode into native instruction on the fly (JIT = Just-In-Time)

Java Virtual Machine (JVM)

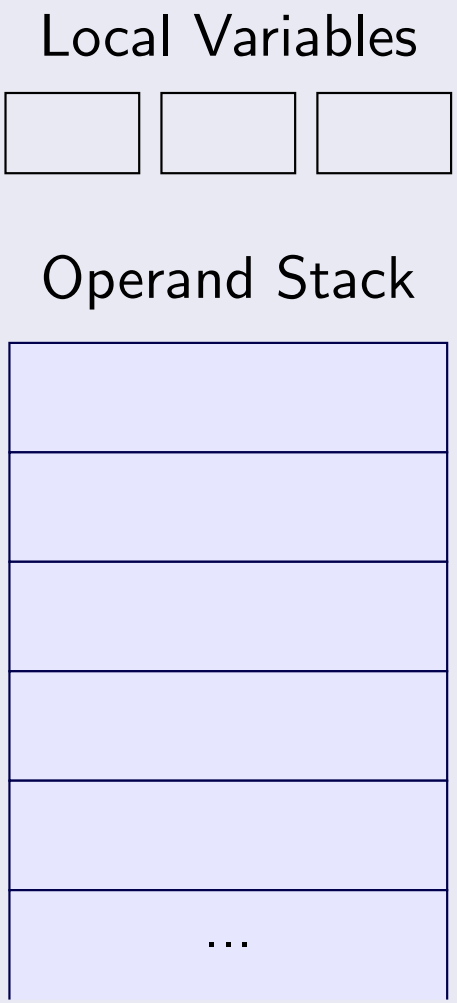
- The Java Virtual Machine is an abstract computing machine.
- It has an instruction set and manipulates various memory areas at run time
- The Java Virtual Machine knows nothing of the Java programming language, only of a particular binary format, the class file format.
- A class file contains Java Virtual Machine instructions (or Java ByteCode) as well as other information.

Frames

In the JVM, a frame:

- stores data and partial results;
- performs dynamic linking;
- returns values for methods;
- and dispatches exceptions.

Frame



- Local Variables: array of variables declared locally in a method (includes parameters)
- Operand Stack: LIFO (Last-In Last-Out) stack of operands

Java Stack != Operand Stack

The operand stack is used to perform computation. The Java stack is used to keep track of function calls.

Frame for foo

Local Variables

a	b

Operand Stack

13
7

New frame for bar

Local Variables

x=7	y=13

Operand Stack

Example

```
void bar(int x, int y) {...}
void foo(int a) {
  int b;
  bar(7,13);
  // push 7
  // push 13
  // call bar
  ...
}
```

What is Java ByteCode?

Java ByteCode is the virtual instruction set of the Java virtual machine.

- One-byte instruction
- 256 possible opcodes (200+ in use)
- Stack-based computation

Suggested reading:

- The Java Virtual Machine Specification: `http://docs.oracle.com/javase/specs/jvms/se8/html/index.html`
- Instructions listing: `https://en.wikipedia.org/wiki/Java_bytecode_instruction_listings`
- Online tutorial `http://blog.jamesdbloom.com/JavaCodeToByteCode_PartOne.html`

JVM Types

Java byte code instructions operates on 9 main different types:

- the 8 primitives types: byte, char, short, int, long, float, double
  - boolean, byte, short and char are sometimes treated as int
- and reference
  - a reference is a pointer to an Object in the heap
- long and double values takes two slots in the operand stack and local variables (all the other types takes one)

Mnemonics

- A mnemonic is a textual form of an operation
- Each mnemonic is encoded as a byte in the class file
- This byte is called an operation code or opcode

Examples:

- iadd: add two integers
- fmul: multiply two floats
- lload_1 : load a long value from the local variable 1

Mnemonics

Prefix/Suffix	Type	Size(in byte)
b	byte	1
s	short	2
c	char	1
i	int	4
l	long	8
f	float	4
d	double	8
a	reference	4 or 8

Instructions dealing with the stack or local variables start with a letter corresponding to the type. Examples:

- iadd: adds two integers
- dadd: add two doubles

Arguments

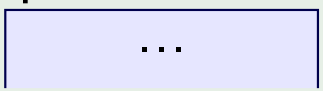
Arguments follows an instruction

Examples:

- bipush 5 : load a byte onto the stack
- ldc "Hello World!" : load a constant from the constant pool
- iconst_0 : load 0 onto the stack

Example

Operand Stack

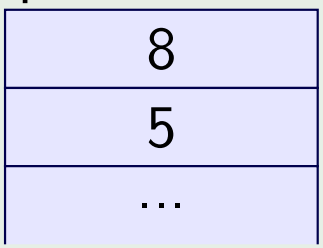


after

bipush 5

bipush 8

Operand Stack



Operands

Operands are taken from the operand stack and resulting value is produced on the stack

Examples:

- iadd

Example

Operand Stack

8
5
...

after

iadd

Operand Stack

13
...

Exercise

- Write the ByteCode for the following expression: 5*(3+4)
- Write down the status of the stack after each instruction

Local variables can be retrieved via load/store instructions.

Frame for foo

Local Variables

33	5

Operand Stack

5
33
7
40

Example

```
... foo(33) ...
void foo(int a) {
  bipush 5
  istore 1
  bipush 7
  iload 0
  iadd
}
```

Function Call and Return Value

Function call

A function call is performed with one of the invoke instructions (dynamic, interface, special, static, virtual). When a function call occurs, a new frame is created and the arguments taken from the operand stack become local variables

Return value

When a value is returned by the function, the current frame is destroyed and the return value is passed back to the callee onto its operand stack.

Example: SmallC/Java

```
int add(int x, int y) {
  return x+y;
}
void main() {
  add(7,13);
}
```

Example: Java ByteCode

```
int add(int x, int y) {
  iload_0
  iload_1
  ireturn
}
void main() {
  bipush 7
  bipush 13
  invokestatic "add(II)I"
}
```

Branching instructions

Branching instructions takes a label that points to the place where to jump to. Java ByteCode offers both unconditional and conditional branches.

Unconditional branch:

- goto

Conditional branches:

- if_icmpeq jump if two stack operands are equals
- if_icmplt jump if the first operand is less than the second one
- ...
- ifeq jump if operand is 0
- ifge jump if operand is greater than 0
- ...

Exmple

```
0: iload_1
1: iload_2
2: if_icmple 7
5: iconst_0
6: ireturn
7: iconst_1
8: ireturn
```

First the two parameters are loaded onto the operand stack using iload_1 and iload_2. if_icmple then compares the top two values on the operand stack. This operand branches to byte code 7 if the first operand is less then or equal to the second one.

Next lecture:

- Introduction to code generation