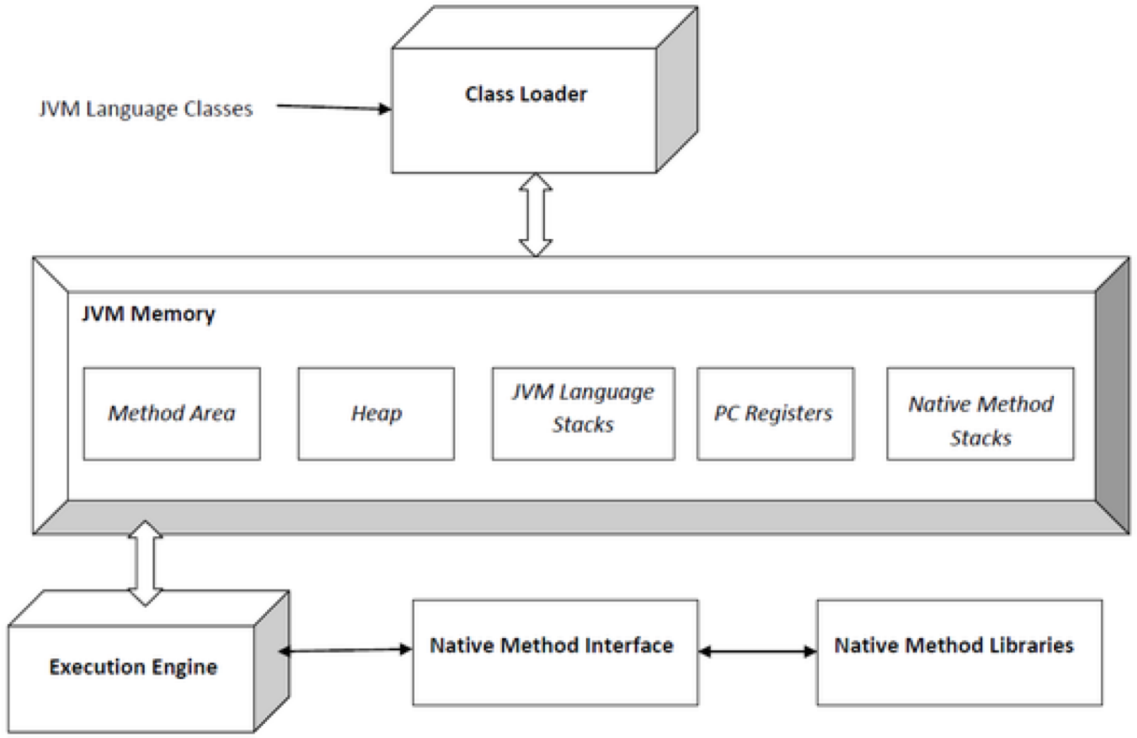
Introduction

- Specification document (Java SE 16 edition, 2021-02-12)
  - https://docs.oracle.com/javase/specs/jvms/se16/jvms16.pdf
- An abstract computing machine
  - Has stack-based instruction set (aka *bytecodes*).
  - Manipulates various memory areas at run time.
  - Does not assume any particular implementation technology, host hardware, or host operating system.
- Orthogonal to the Java® programming language
  - Program unit is a `class` file.
    - Contains bytecodes and a symbol table.
    - Must satisfy strong syntactic and structural constraints for security.
  - Any language that can be expressed in terms of a valid `class` file can be hosted on the JVM, e.g., Groovy, Kotlin, Scala, Clojure.
  - Java on Android used to be compiled to JVM bytecode, which was then translated to Dalvik bytecode (`.dex` files). Since Android 5.0 “Lollipop”, Dalvik VM has been replaced by Android Runtime (ART).

Architecture of the Java Virtual Machine



Run-Time Data Areas

Name	Scope	Lifetime	Contents
The `pc` Register	Per-thread	Thread	Address of JVM instruction currently being executed.
Java Virtual Machine Stacks	Per-thread	Thread	Stack frames, containing local variables, partial results, operand stacks.
Heap	Global, shared across threads	Virtual machine	Class instances and arrays. Garbage-collected.
Method Area	Global, shared across threads	Virtual machine	Per-class structures such as run-time constant pool, field and method data, code for methods and constructors.
Run-Time Constant Pool	Per-class or per-interface	Class or interface	Several kinds of constants, ranging from numeric literals to field references.
Native Method Stacks (optional)	Typically per-thread	Thread	Conventional stacks to support `native` methods.

Java Virtual Memory Instruction Opcodes

- Bytecode, i.e., 8-bit opcodes.
- Additional arguments may be specified.
- Instruction categories
  - Constants (00-20): nop, iconst_0, bipush, ldc, …
  - Loads (21-53): iload, fload_1, aload_3, aaload, …
  - Stores (54-86)
  - Stack (87-95): pop, dup, swap, …
  - Math (96-132): ladd, dsub, ishl, iand, …
  - Conversions (133-147): i2l, f2i, f2d, i2b, …
  - Comparisons (148-166): lcmp, iflt, if_icmpge, if_acmpne, …
  - Control (167-177): goto, jsr, ret, areturn, …
  - References (178-195): getstatic, putfield, invokevirtual, new, …
  - Extended (196-201): wide, ifnull, goto_w, …
  - Reserved (202-255): breakpoint, …

Compilation Example

```
void spin() {
  int i;
  for (i = 0; i < 100; i++);
}
```

Compilation Example

```
void spin() {
  int i;
  for (i = 0; i < 100; i++);
}
```

```
03 3C A7 00 06 84 01 01
1B 10 64 A1 FF FA B1
```

Compilation Example

```
void spin() {
  int i;
  for (i = 0; i < 100; i++);
}

03 3C A7 00 06 84 01 01
1B 10 64 A1 FF FA B1

0  iconst_0    // Push int constant 0
1  istore_1    // Store into local variable 1
2  goto 8      // First time through don't increment
5  iinc 1 1    // Increment local variable 1 by 1
8  iload_1     // Push local variable 1
9  bipush 100  // Push int constant 100
11 if_icmplt 5 // Compare and loop if less than
14 return      // Return void when done
```