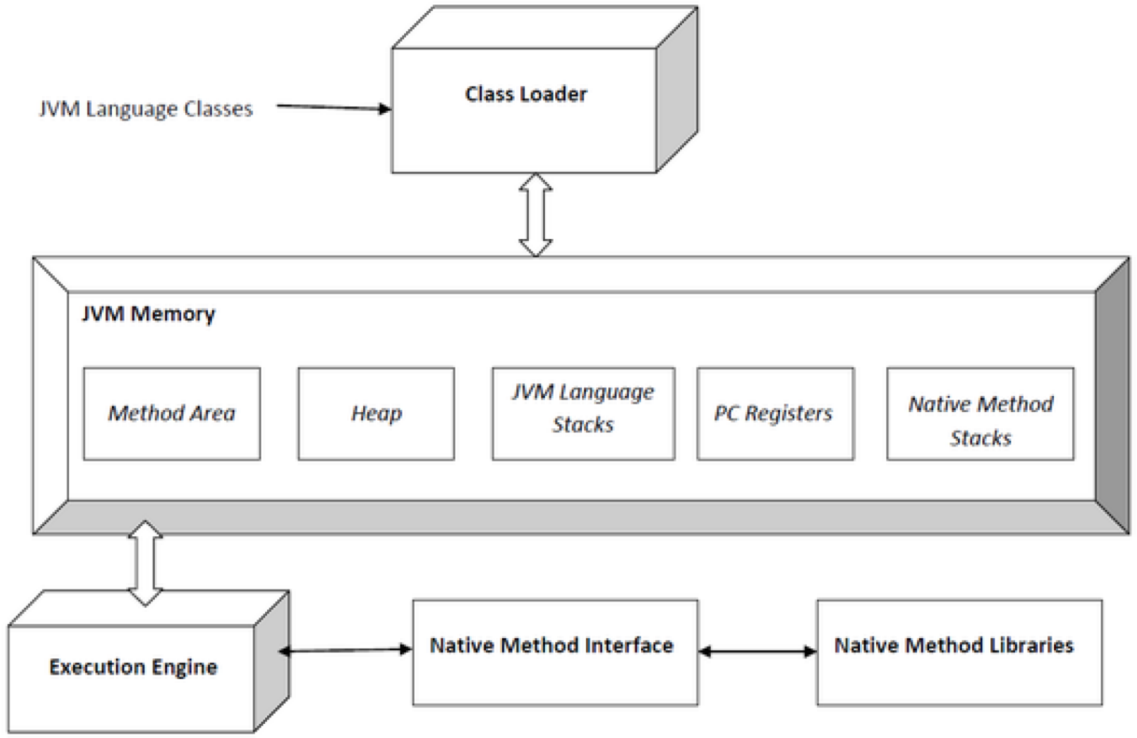
Architecture of the Java Virtual Machine



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, …

Java® Virtual Machine Support for Objects

- [Ref: JVM SE16 (§2.11.5, §2.11.8, §6.5)]
- Both class instances and arrays are objects.
  - JVM has different sets of instructions for creating and manipulating these two kinds of objects.
  - Ignoring arrays in this presentation.
- Instructions
  - Create a new class instance: `new`.
    - No instruction for object reclamation: handled by GC.
  - Access fields of classes (i.e., `static` fields or class variables): `getstatic`, `putstatic`.
  - Access fields of class instances (i.e., non-`static` fields or instance variables): `getfield`, `putfield`.
  - Check properties of class instances (or arrays): `instanceof`, `checkcast`.
  - Method invocation: `invokevirtual`, `invokeinterface`, `invokespecial`, `invokestatic`.
    - Also `invokedynamic` (ignoring).

Object Allocation and Initialization

- Combination of `new` (to allocate the object) and `invokespecial` (to execute constructor method).
- Details
  - [`new ib1 ib2`]
    - The bytes `ib1` and `ib2` used to construct an index into the run-time constant pool of current class; the item at that index must be a symbolic reference to a class or an interface type.
    - Type resolution performed; must result in a class type.
      - If the class is currently uninitialized, its `<clinit>` method is invoked.
    - Memory for a new instance of that class allocated from heap.
    - Instance variables of the new object are initialized to their default initial values.
    - A reference to the instance is pushed on the operand stack.
  - [`invokespecial ib1 ib2`]
    - Skipping details, but essentially follows the above pattern to locate and resolve the method name and to look up the actual procedure to be executed *by searching through the class, its superclasses, and its superinterfaces*.

Object Allocation and Initialization

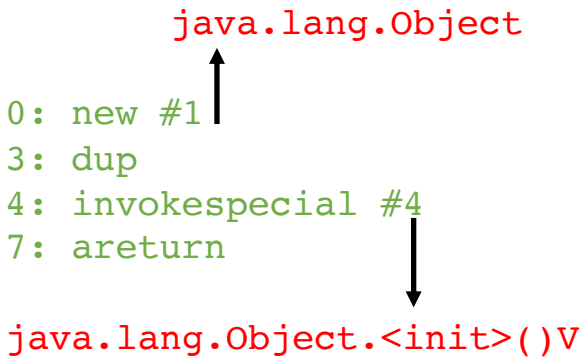
- Combination of `new` (to allocate the object) and `invokespecial` (to execute constructor method).
- Details
  - `[new ib1 ib2]`
    - The bytes `ib1` and `ib2` used to construct an index into the run-time constant pool of current class; the item at that index must be a symbolic reference to a class or an interface type.
    - Type resolution performed; must result in a class type.
      - If the class is currently uninitialized, its `<clinit>` method is invoked.
    - Memory for a new instance of that class allocated from heap.
    - Instance variables of the new object are initialized to their default initial values.
    - A reference to the instance is pushed on the operand stack.
  - `[invokespecial ib1 ib2]`
    - Skipping details, but essentially follows the above pattern to locate and resolve the method name and to look up the actual procedure to be executed *by searching through the class, its superclasses, and its superinterfaces*.

```
Object create() {
    return new Object();
}
```

Object Allocation and Initialization

- Combination of `new` (to allocate the object) and `invokespecial` (to execute constructor method).
- Details
  - `[new ib1 ib2]`
    - The bytes `ib1` and `ib2` used to construct an index into the run-time constant pool of current class; the item at that index must be a symbolic reference to a class or an interface type.
    - Type resolution performed; must result in a class type.
      - If the class is currently uninitialized, its `<clinit>` method is invoked.
    - Memory for a new instance of that class allocated from heap.
    - Instance variables of the new object are initialized to their default initial values.
    - A reference to the instance is pushed on the operand stack.
  - `[invokespecial ib1 ib2]`
    - Skipping details, but essentially follows the above pattern to locate and resolve the method name and to look up the actual procedure to be executed *by searching through the class, its superclasses, and its superinterfaces*.

```
Object create() {
    return new Object();
}
```



Bytecodes for Method Invocation

- Method invocation: `invokevirtual`, `invokeinterface`, `invokespecial`, `invokestatic`.
  - `invokevirtual`: [182, ib1, ib2] Invoke an instance method, dispatch based on class.
    - Object reference and parameters are taken from the operand stack.
  - `invokespecial`: [183, ib1, ib2] Invoke an instance method; direct invocation of instance initialization methods and methods of the current class and its supertypes.
    - Key difference is that procedure to be executed is searched in class hierarchy.
  - `invokestatic` : [184, ib1, ib2] Invoke a class (`static`) method.
    - No object reference as the first argument.
  - `invokeinterface`: [185, ib1, ib2, count, 0] Invoke an interface (instance) method.
    - The count and 0 are for historical reasons and backwards compatibility.