Java Virtual Machine

VM Functional Units			
Frame	Local variables	int[]	local variables; addressed by indexing; parameters passed as local variables on method invocation
	Operand Stack	int[]	LIFO operand stack; used to pass parameters to methods and to receive method results
	Constant Pool	cp_info[]	a per-class or per-interface run-time representation of the constant_pool table in a class file
	Code	byte[]	instructions; accessed by byte offsets
	pc	wide enough to hold a native pointer	per-thread; contains the address of the Java Virtual Machine instruction currently being executed
	JVM stack	frame[]	stores frames; analogous to the stack of a conventional language such as C
	Heap	byte[]	the run-time data area from which memory for all class instances and arrays is allocated

Instruction Formats (big endian)	
Opcode	1 byte
Explicit operands	stored in instruction memory, 1 byte per *parameter* (wide iinc *parameters* use 2 bytes, also marked with the 2-prefix)
Implicit operands	stored on the operand stack, in this representation the stack grows from left to right, and the -> shows the result stack

Method Invocation	
Class method caller	pushes the args on the operand stack and calls `invokestatic`; args are popped, result is pushed
Class method callee	params are stored in the local variables starting from index 0, pushes result to the operand stack and calls `return`
Instance method caller	pushes the objectref and the args to the operand stack and calls `invokevirtual`; objectref and args are popped, result is pushed
Instance method callee	params are stored in the local variables starting from index 1, `this` is stored at index 0, pushes result to the operand stack and calls `return`

Class File Structure		
u4	magic	magic number 0xCAFEBABE
u4	u2 minor_version, u2 major_version	version number (Java 14: major 58, minor 0 or 65535)
u2	constant_pool_count	number of entries in the constant_pool table plus one
cp_info	constant_pool[constant_pool_count - 1]	representing various string constants, class and interface names, field names, and other constants
u2	access_flags	denotes access permissions to and properties of this class or interface
u2	this_class	valid index into the constant_pool table representing the class or interface defined by this file
u2	super_class	valid index into the constant_pool table representing the direct superclass of this file
u2	interfaces_count	the number of direct superinterfaces of this class or interface type
u2	interfaces[interfaces_count]	each value must be a valid index into the constant_pool table
u2	fields_count	the number of field_info structures in the fields table
field_info	fields[fields_count]	each value must be a field_info structure giving a complete description of a field
u2	methods_count	the number of method_info structures in the methods table
method_info	methods[methods_count]	each value must be a method_info structure giving a complete description of a method
u2	attributes_count	the number of attributes in the attributes table of this class

Java Virtual Machine

attribute_info		attributes[attributes_count]	each value of the attributes table must be an attribute_info structure	
	Opcode	Mnemonic	Format	Stack	Operation
Constants	0x02	iconst_*i*	iconst_m1	… --> …, -1	Push int constant -1
	0x03	iconst_*i*	iconst_0	… --> …, 0	Push int constant 0
	0x04	iconst_*i*	iconst_1	… --> …, 1	Push int constant 1
	0x05	iconst_*i*	iconst_2	… --> …, 2	Push int constant 2
	0x06	iconst_*i*	iconst_3	… --> …, 3	Push int constant 3
	0x07	iconst_*i*	iconst_4	… --> …, 4	Push int constant 4
	0x08	iconst_*i*	iconst_5	… --> …, 5	Push int constant 5
	0x10	bipush	bipush *byte*	… -> …, *value*	Push byte
	0x11	sipush	sipush *byte1 byte2*	… -> …, *value*	Push short
	0x12	ldc	ldc *index*	… -> …, *value*	Push item from run-time constant pool
	0x13	ldc_w	ldx *indexbyte1 indexbyte2*	… -> …, *value*	Push item from run-time constant pool (wide index)
Loads	0x15	iload	iload *index*	… -> …, *value*	Load int from local variable
	0x19	aload	aload *index*	… -> …, *value*	Load reference from local variable
	0x1a	iload_*n*	iload_0	… -> …, *value*	Load int from local variable
	0x1b	iload_*n*	iload_1	… -> …, *value*	Load int from local variable
	0x1c	iload_*n*	iload_2	… -> …, *value*	Load int from local variable
	0x1d	iload_*n*	iload_3	… -> …, *value*	Load int from local variable
	0x2a	aload_*n*	aload_0	… -> …, *value*	Load reference from local variable
	0x2b	aload_*n*	aload_1	… -> …, *value*	Load reference from local variable
	0x2c	aload_*n*	aload_2	… -> …, *value*	Load reference from local variable
	0x2d	aload_*n*	aload_3	… -> …, *value*	Load reference from local variable
	0x2e	iaload	iaload	…, *arrayref, index* -> …, *value*	Load int from array
	0x32	aaload	aaload	…, *arrayref, index* -> …, *value*	Load reference from array
	0x33	baload	baload	…, *arrayref, index* -> …, *value*	Load byte or boolean from array
Stores	0x36	istore	istore *index*	…, *value* -> …	Store int into local variable
	0x3a	astore	astore *index*	…, *value* -> …	Store reference into local variable
	0x3b	istore_*n*	istore_0	…, *value* -> …	Store int into local variable
	0x3c	istore_*n*	istore_1	…, *value* -> …	Store int into local variable
	0x3d	istore_*n*	istore_2	…, *value* -> …	Store int into local variable
	0x3e	istore_*n*	istore_3	…, *value* -> …	Store int into local variable
	0x4b	astore_*n*	astore_0	…, *value* -> …	Store reference into local variable

Stores	0c4c	astore_*n*	astore_1	..., *value* -> ...	Store reference into local variable
	0x4d	astore_*n*	astore_2	..., *value* -> ...	Store reference into local variable
	0x4e	astore_*n*	astore_3	..., *value* -> ...	Store reference into local variable
	0x4f	iastore	iastore	..., *arrayref, index, value* -> ...	Store into int array
	0x53	aastore	aastore	..., *arrayref, index, value* -> ...	Store into reference array
	0x54	bastore	bastore	..., *arrayref, index, value* -> ...	Store into byte or boolean array
Stack	0x57	pop	pop	..., *value* -> ...	Pop the top operand stack value
	0x59	dup	dup	..., *value* -> ..., *value, value*	Duplicate the top operand stack value
	0x5f	swap	swap	..., *value1, value2* -> ..., *value2, value1*	Swap the top two operand stack values
Math	0x60	iadd	iadd	..., *value1, value2* -> ..., *result*	Add int
	0x64	isub	isub	..., *value1, value2* -> ..., *result*	Subtract int
	0x68	imul	imul	..., *value1, value2* -> ..., *result*	Multiply int
	0x6c	idiv	idiv	..., *value1, value2* -> ..., *result*	Divide int
	0x70	irem	irem	..., *value1, value2* -> ..., *result*	Remainder int
	0x74	ineg	ineg	..., *value* -> ..., *result*	Negate int
	0x78	ishl	ishl	..., *value1, value2* -> ..., *result*	Shift left int
	0x7a	ishr	ishr	..., *value1, value2* -> ..., *result*	Arithmetic shift right int
	0x84	iinc	iinc *index const*	[no change]	Increment local variable by constant
Comparison	0x99	if*cond*	ifeq *branchbyte1 branchbyte2*	..., *value* -> ...	Branch if int comparison with zero succeeds
	0x9a	if*cond*	ifne *branchbyte1 branchbyte2*	..., *value* -> ...	Branch if int comparison with zero succeeds
	0x9b	if*cond*	iflt *branchbyte1 branchbyte2*	..., *value* -> ...	Branch if int comparison with zero succeeds
	0x9c	if*cond*	ifge *branchbyte1 branchbyte2*	..., *value* -> ...	Branch if int comparison with zero succeeds
	0x9d	if*cond*	ifgt *branchbyte1 branchbyte2*	..., *value* -> ...	Branch if int comparison with zero succeeds
	0x9e	if*cond*	ifle *branchbyte1 branchbyte2*	..., *value* -> ...	Branch if int comparison with zero succeeds
	0x9f	if_icmp*cond*	if_icmpeq *branchbyte1 branchbyte2*	..., *value1, value2* -> ...	Branch if int comparison succeeds
	0xa0	if_icmp*cond*	if_icmpne *branchbyte1 branchbyte2*	..., *value1, value2* -> ...	Branch if int comparison succeeds
	0xa1	if_icmp*cond*	if_icmplt *branchbyte1 branchbyte2*	..., *value1, value2* -> ...	Branch if int comparison succeeds
	0xa2	if_icmp*cond*	id_icmpge *branchbyte1 branchbyte2*	..., *value1, value2* -> ...	Branch if int comparison succeeds
	0xa3	if_icmp*cond*	if_icmpgt *branchbyte1 branchbyte2*	..., *value1, value2* -> ...	Branch if int comparison succeeds
	0xa4	if_icmp*cond*	if_icmple *branchbyte1 branchbyte2*	..., *value1, value2* -> ...	Branch if int comparison succeeds
	0xa5	if_acmp*cond*	if_acmpeq *branchbyte1 branchbyte2*	..., *value1, value2* -> ...	Branch if reference comparison succeeds
	0xa6	if_acmp*cond*	if_acmpne *branchbyte1 branchbyte2*	..., *value1, value2* -> ...	Branch if reference comparison succeeds

Java Virtual Machine

Control	0xa7	goto	goto *branchbyte1 branchbyte2*	[no change]	Branch always
	0xac	ireturn	ireturn	…, value -> [empty]	Return int from method
	0xb0	areturn	areturn	…, objectref -> [empty]	Return reference from method
	0xb1	return	return	… -> [empty]	Return void from method
References	0xb2	getstatic	getstatic *indexbyte1 indexbyte2*	… -> …, *value*	Get static field from class
	0xb3	putstatic	putstatic *indexbyte1 indexbyte2*	…, *value* -> ...	Set static field in class
	0xb4	getfield	getfield *indexbyte1 indexbyte2*	…, *objectref* -> …, *value*	Fetch field from object
	0xb5	putfield	putfield *indexbyte1 indexbyte2*	…, *objectref, value* -> ...	Set field in object
	0xb6	invokevirtual	invokevirtual *indexbyte1 indexbyte2*	…, *objectref,* [*arg1* , [*arg2* , …]] -> [*retVal*], …	Invoke instance method; dispatch based on class
	0xb7	invokespecial	invokespecial *indexbyte1 indexbyte2*	…, *objectref,* [*arg1* , [*arg2* , …]] -> [*retVal*], …	Invoke instance method; instance initialization methods
	0xb8	invokestatic	invokestatic *indexbyte1 indexbyte2*	…, [*arg1* , [*arg2* , …]] -> …	Invoke a class (static) method
	0xbb	new	new *indexbyte1 indexbyte2*	… -> …, *objectref*	Create new object
	0xbc	newarray	newarray *aType*	…, *count* -> …, *arrayref*	Create new array
	0xbd	anewarray	anewarray *indexbyte1 indexbyte2*	…, *count* -> …, *arrayref*	Create new array of reference
	0xc5	multianewarray	multianewarray *2indexbytes dimensions*	…, *count1,* [*count2, …*] -> …, *arrayref*	Creates a new multidimensional array
	0xbe	arraylength	arraylength	…, *arrayref* -> …, *length*	Get length of array
Wide	0xc4	wide	wide *opcode indexbyte1 indexbyte2* wide iinc *2indexbytes 2constbytes*	[Same as modified instruction]	Extend local variable index by additional bytes (opcode: iload / istore)