CSc 453 — Compilers and Systems Software

17 : The Java VM

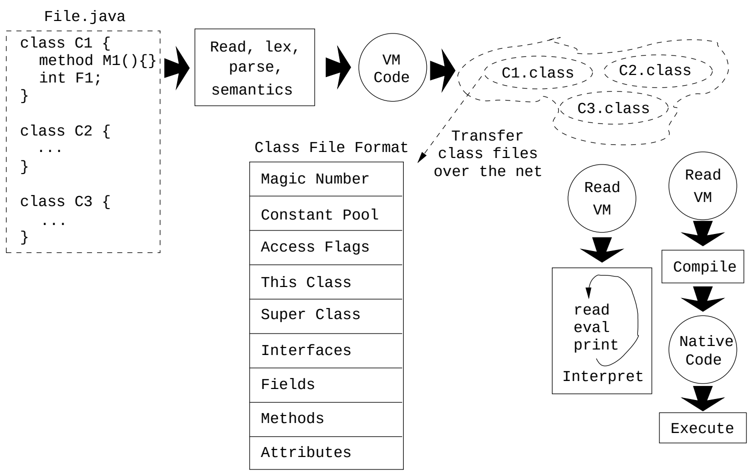
Christian Collberg
Department of Computer Science
University of Arizona
collberg@gmail.com

October 25, 2009

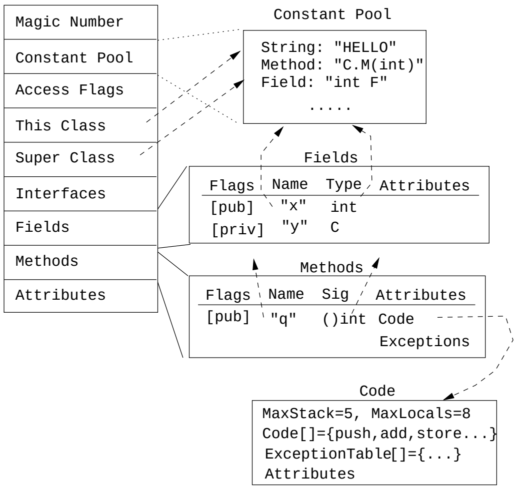
1 The Java Virtual Machine

- The Java VM has gone the "many complex instructions/large VM engine" way.
- Each Java source file may contain several Java classes. The Java compiler compiles each of these classes to a single Java *class file*.
- The Java class file stores all necessary data regarding the class. There is a symbol table (called the *Constant Pool*) which stores strings, large literal integers and floats, names and of all fields and methods.
- Each method is compiled to Java bytecode, a stack VM format.
- The class file is (almost) isomorphic to the source.

2



3



4 Java Byte Codes

- The Java bytecodes can manipulate data in these formats: integers (32-bits), longs (64-bits), floats (32-bits), doubles (64-bits), shorts (16-bits), bytes (8-bits), object references (32/64-bit pointers), and arrays.
- The bytecodes are 1 byte wide.
- Each method can have up to 256 local variables and formal parameters. The bytecode reference these by number.
- Actually, we can have up to 65536 local vars. There is a special `wide` instruction that modifies load and store instructions to reference the high-numbered locals. Hack.

5 Java Byte Codes...

- The Java stack is 32-bits wide. Longs and doubles hence take two stack entries.
- The bytecodes reference data from the class' constant pool. These references are 8 or 16 bits long. To push a reference to a literal string with constant pool # 4567, use `'ldc2 4567'`. If the # is 123, use `'ldc2 123'`.

6 Java Byte Codes...

$\texttt{int}_8$	An 8-bit integer value.
$\texttt{int}_{16}$	A 16-bit integer value.
$\texttt{int}_{32}$	A 32-bit integer value.
$\texttt{CP}_8$	An 8-bit constant pool index.
$\texttt{CP}_{16}$	A 16-bit constant pool index.
`FIdx`	An 8-bit local variable index.
$\texttt{FIdx}_{16}$	A 16-bit local variable index.
$\texttt{CP}[i]$	The i:th constant pool entry.
$\texttt{Var}[i]$	The i:th variable/formal parameter in the current method.

7

Opcode	Mnemonic	Args	Stack	Description
0	`nop`		$[] \Rightarrow []$	
1	`aconst_null`		$[] \Rightarrow$ [null]	Push null object
2	`iconst_m1`		$[] \Rightarrow$ [-1]	Push -1
$3 \ldots 8$	`iconst_`n		$[] \Rightarrow [n]$	Push integer constant $n, 0 \leq n \leq 5$
$9 \ldots 10$	`lconst_`n		$[] \Rightarrow [n]$	Push long constant $n, 0 \leq n \leq 1$
$11 \ldots 13$	`fconst_`n		$[] \Rightarrow [n]$	Push float constant $n, 0 \leq n \leq 2$
$14 \ldots 15$	`dconst_`n		$[] \Rightarrow [n]$	Push double constant $n, 0 \leq n \leq 1$

8

Opcode	Mnemonic	Args	Stack	Description
16	`bipush`	n:$\texttt{int}_8$	$[] \Rightarrow [n]$	Push 1-byte signed integer
17	`sipush`	n:$\texttt{int}_{16}$	$[] \Rightarrow [n]$	Push 2-byte signed integer
18	`ldc1`	n:$\texttt{CP}_8$	$[] \Rightarrow [\mathrm{CP}[n]]$	Push item from constant pool
19	`ldc2`	n:$\texttt{CP}_{16}$	$[] \Rightarrow [\mathrm{CP}[n]]$	Push item from constant pool
20	`ldc2w`	n:$\texttt{CP}_{16}$	$[] \Rightarrow [\mathrm{CP}[n]]$	Push long/double from constant pool

9

Opcode	Mnemonic	Args	Stack
$21 \ldots 25$	X`load` $X \in$\{i,l,f,d,a\}, Load int, long, float, double, object from local var.	n:`FIdx`	$[] \Rightarrow [\mathrm{Var}[n]]$
$26 \ldots 29$	`iload_`n Load local integer var $n, 0 \leq n \leq 3$		$[] \Rightarrow [\mathrm{Var}[n]]$
$30 \ldots 33$	`lload_`n Load local long var $n, 0 \leq n \leq 4$		$[] \Rightarrow [\mathrm{Var}[n]]$
$34 \ldots 37$	`fload_`n Load local float var $n, 0 \leq n \leq 4$		$[] \Rightarrow [\mathrm{Var}[n]]$
$38 \ldots 41$	`dload_`n Load local double var $n, 0 \leq n \leq 4$		$[] \Rightarrow [\mathrm{Var}[n]]$

10

Opcode	Mnemonic	Args	Stack
$42 \ldots 45$	`aload_`n		$[] \Rightarrow [\mathrm{Var}[n]]$
	Load local object var $n, 0 \leq n \leq 4$		
$46 \ldots 53$	X`load`		$[A, I] \Rightarrow [V]$
	X $\in$ {ia,la,fa,da,aa,ba, ca,sa}. Push the value V (an int, long, etc.) stored at index I of array A.		
$54 \ldots 58$	X`store`	n:`FIdx`	$[\mathrm{Var}[n]] \Rightarrow []$
	X $\in$ {i,l,f,d,a}, Store int, long, float, double, object to local var.		
$59 \ldots 62$	`istore_`n		$[\mathrm{Var}[n]] \Rightarrow []$
	Store to local integer var $n, 0 \leq n \leq 3$		
$63 \ldots 66$	`lstore_`n		$[\mathrm{Var}[n]] \Rightarrow []$
	Store to local long var $n, 0 \leq n \leq 4$		

11

Opcode	Mnemonic	Args	Stack
$67 \ldots 70$	`fstore_`n		$[\mathrm{Var}[n]] \Rightarrow []$
	Store to local float var $n, 0 \leq n \leq 4$		
$71 \ldots 74$	`dstore_`n		$[\mathrm{Var}[n]] \Rightarrow []$
	Store to local double var $n, 0 \leq n \leq 4$		
$75 \ldots 78$	`astore_`n		$[\mathrm{Var}[n]] \Rightarrow []$
	Store to local object var $n, 0 \leq n \leq 4$		
$79 \ldots 86$	X`store`		$[A, I, V] \Rightarrow []$
	X $\in$ {ia,la,fa,da,aa,ba,ca,sa}. Store the value V (an int, long, etc.) at index I of array A.		
87	`pop`		$[A] \Rightarrow []$
	Pop top of stack.		

12

Opcode	Mnemonic	Stack	Description
88	`pop2`	$[A, B] \Rightarrow []$	Pop 2 elements.
89	`dup`	$[V] \Rightarrow [V, V]$	
	Duplicate top of stack.		
90	`dup_x1`	$[B, V] \Rightarrow [V, B, V]$	Duplicate.
91	`dup_x2`	$[B, C, V] \Rightarrow [V, B, C, V]$	Duplicate.
92	`dup2`	$[V, W] \Rightarrow [V, W, V, W]$	Duplicate.
93	`dup2_x1`	$[A, V, W] \Rightarrow [V, W, A, V, W]$	Duplicate.
94	`dup2_x2`	$[A, B, V, W] \Rightarrow [V, W, A, B, V, W]$	Duplicate.
95	`swap`	$[A, B] \Rightarrow [B, A]$	
	Swap top stack elements.		

13

Opcode	Mnemonic	Stack	Description
96...99	X`add`	$[A,B] \Rightarrow [R]$	$X \in$ {i,l,d,f}. $R = A + B$
100...103	X`sub`	$[A,B] \Rightarrow [R]$	$X \in$ {i,l,d,f}. $R = A - B$
104...107	X`mul`	$[A,B] \Rightarrow [R]$	$X \in$ {i,l,d,f}. $R = A * B$
108...111	X`div`	$[A,B] \Rightarrow [R]$	$X \in$ {i,l,d,f}. $R = A/B$
112...115	X`mod`	$[A,B] \Rightarrow [R]$	$X \in$ {i,l,d,f}. $R = A\%B$
116...119	X`neg`	$[A] \Rightarrow [R]$	$X \in$ {i,l,d,f}. $R = -A$
120...121	X`shl`	$[A,B] \Rightarrow [R]$	$X \in$ {i,l}. $R = A << B$
122...123	X`shl`	$[A,B] \Rightarrow [R]$	$X \in$ {i,l}. $R = A >> B$
124...125	X`ushr`	$[A,B] \Rightarrow [R]$	$X \in$ {i,l}. $R = A >>> B$
126...127	X`and`	$[A,B] \Rightarrow [R]$	$X \in$ {i,l}. $R = A\&\&B$
128...129	X`or`	$[A,B] \Rightarrow [R]$	$X \in$ {i,l}. $R = A\|\|B$
130...131	X`xor`	$[A,B] \Rightarrow [R]$	$X \in$ {i,l}. $R = A$`xor`B

14

Opcode	Mnemonic	Args	Stack
133...144	$X2Y$`cnv`		$[F] \Rightarrow [T]$
	Convert F from type X to T of type Y. $X \in$ {i,l,f,d}, $Y \in$ {i,l,f,d}.		
145...147	`i2`X		$[F] \Rightarrow [T]$
	$X \in$ {b,c,s}. Convert integer F to byte, char, or short.		
148,149,151	X`cmp`		$[A,B] \Rightarrow [V]$
	$X \in$ {l,f,d}. $A > B \Rightarrow V = 1$, $A < B \Rightarrow V = -1$, $A = B \Rightarrow V = 0$. $A = \texttt{NaN} \vee B = \texttt{NaN} \Rightarrow V = -1$		
150,152	X`cmp`		$[A,B] \Rightarrow [V]$
	$X \in$ {f,d}. $A > B \Rightarrow V = 1$, $A < B \Rightarrow V = -1$, $A = B \Rightarrow V = 0$. $A = \texttt{NaN} \vee B = \texttt{NaN} \Rightarrow V = 1$		
153...154	`if`$\diamond$	L:`int`$_{16}$	$[A] \Rightarrow []$
	$\diamond$={eq,ne,lt,ge,gt,le}. If $A \diamond 0$ goto $L +$ `pc`.		

15

Opcode	Mnemonic	Args	Stack
159...164	`if_icomp`$\diamond$	L:`int`$_{16}$	$[A,B] \Rightarrow []$
	$\diamond$={eq,ne,lt,ge,gt,le}. If $A \diamond B$ goto $L +$ `pc`.		
165...166	`if_acmp`$\diamond$	L:`int`$_{16}$	$[A,B] \Rightarrow []$
	$\diamond$={eq,ne}. A, B are object refs. If $A \diamond B$ goto $L +$ `pc`.		
167	`goto`	I:`int`$_{16}$	$[] \Rightarrow []$
	Goto instruction I.		
168	`jsr`	I:`int`$_{16}$	$[] \Rightarrow []$
	Jump subroutine to instruction $I +$ `pc`.		
172...177	X`return`		$[V] \Rightarrow []$
	$X \in$ {i,f,l,d,a,v}. Return V.		
169	`ret`	L:`FIdx`	$[] \Rightarrow []$
	Return from subroutine. Address in local var L.		

16

Opcode	Mnemonic	Args	Stack
170	`tableswitch`	D:$\mathtt{int}_{32}$,l,h:$\mathtt{int}_{32}$,o^{h-l+1}	$[K] \Rightarrow []$
	Jump through the K:th offset. Else goto D.		
171	`lookupswitch`	D:$\mathtt{int}_{32}$,n:$\mathtt{int}_{32}$,$(m,o)^n$	$[K] \Rightarrow []$
	If, for one of the (m,o) pairs, $K = m$, then goto o. Else goto D.		
178	`getstatic`	F:$\mathtt{CP}_{16}$	$[] \Rightarrow [V]$
	Push value V of static field F.		
180	`getfield`	F:$\mathtt{CP}_{16}$	$[R] \Rightarrow [V]$
	Push value V of field F in object R.		
179	`putstatic`	F:$\mathtt{CP}_{16}$	$[] \Rightarrow [V]$
	Store value V into static field F.		
181	`putfield`	F:$\mathtt{CP}_{16}$	$[R,V] \Rightarrow []$
	Store value V into field F of object R.		

17

Opcode	Mnemonic	Args	Stack
182	`invokevirtual`	P:$\mathtt{CP}_{16}$	$[R, A_1, A_2, \ldots] \Rightarrow []$
	Call virtual menthod P, with arguments $A_1 \cdots A_n$, through object reference R.		
183	`invokespecial`	P:$\mathtt{CP}_{16}$	$[R, A_1, A_2, \ldots] \Rightarrow []$
	Call private/init/superclass menthod P, with arguments $A_1 \cdots A_n$, through object reference R.		
184	`invokestatic`	P:$\mathtt{CP}_{16}$	$[A_1, A_2, \ldots] \Rightarrow []$
	Call static menthod P with arguments $A_1 \cdots A_n$.		
185	`invokeinterface`	P:$\mathtt{CP}_{16}$,n:$\mathtt{int}_{16}$	$[R, A_1, A_2, \ldots] \Rightarrow []$
	Call interface menthod P, with n arguments $A_1 \cdots A_n$, through object reference R.		
187	`new`	T:$\mathtt{CP}_{16}$	$[] \Rightarrow [R]$
	Create a new object R of type T.		

18

Opcode	Mnemonic	Args	Stack
188	`newarray`	T:$\mathtt{int}_8$	$[C] \Rightarrow [R]$
	Allocate new array R, element type T, C elements long.		
191	`athrow`		$[R] \Rightarrow [?]$
	Throw exception.		
193	`instanceof`	C:$\mathtt{CP}_{16}$	$[R] \Rightarrow [V]$
	Push 1 if object R is an instance of class C. Else push 0.		
194	`monitorenter`		$[R] \Rightarrow []$
	Get lock for object R.		
195	`monitorexit`		$[R] \Rightarrow []$
	Release lock for object R.		
196	`wide`	C:$\mathtt{int}_8$,I:$\mathtt{FIdx}_{16}$	$[] \Rightarrow []$
	Perform opcode C on variable `Var[`I`]`. C is one of the load/-store instructions.		

19

Opcode	Mnemonic	Args	Stack
197	`multianewarray`	T:CP$_{16}$,D:CP$_8$	$[d_1, d_2, \ldots] \Rightarrow [R]$
	Create new D-dimensional multidimensional array R. $d_1, d_2, \ldots$ are the dimension sizes.		
198	`ifnull`	L:int$_{16}$	$[V] \Rightarrow []$
	If $V =$ `null` goto L.		
199	`ifnonnull`	L:int$_{16}$	$[V] \Rightarrow []$
	If $V \neq$ `null` goto L.		
200	`goto_w`	I:int$_{32}$	$[] \Rightarrow []$
	Goto instruction I.		
201	`jsr_w`	I:int$_{32}$	$[] \Rightarrow []$
	Jump subroutine to instruction I.		

20 JVM Example I

```
void spin() {
    int i; for (i = 0; i < 100; i++); // Empty loop body
```

$\Downarrow$

```
0   iconst_0    // Push int constant 0
1   istore_1    // Store into local 1 (i=0)
2   goto 8      // First time through don't increment
5   iinc 1 1    // Increment local 1 by 1(i++)
8   iload_1     // Push local 1 (i)
9   bipush 100  // Push int constant (100)
11  if_icmplt 5 // Compare, loop // if < (i < 100)
14  return      // Return void when done
```

21

```
double i;for (i = 0.0; i < 100.0; i++);  // Empty loop body
```

$\Downarrow$

```
0   dconst_0  // Push double constant 0.0
1   dstore_1  // Store into locals 1 and 2 (i = 0.0)
2   goto 9    // First time no incr
5   dload_1   // Push double
6   dconst_1  // Push double 1.o onto stack
7   dadd      // Add;
8   dstore_1  // Store result in locals 1 and 2
9   dload_1   // Push local
10  ldc2_w #4 // Double 100.000000
13  dcmpg
14  iflt 5    // Compare, loop if < (i < 100.000000)
17  return    // Return void when done
```

22 JVM Example III

```
    double doubleLocals(double d1, double d2) {
```

```
        return d1 + d2;
    }
```

$\Downarrow$

```
0  dload_1 // First argument in locals 1 and 2
1  dload_3 // Second argument in locals 3 and 4
2  dadd    // Each also uses two words on stack
3  dreturn
```

23 JVM Example IV

```
    int align2grain(int i, int grain) {
        return ((i + grain-1) & ~(grain-1));}
```

$\Downarrow$

```
0   iload_1
1   iload_2
2   iadd
3   iconst_1
4   isub
5   iload_2
6   iconst_1
7   isub
8   iconst_m1
9   ixor
10  iand
11  ireturn
```

24 JVM Example V

```
void useManyNumeric() {
    int i = 100; int j = 1000000;
    long l1 = 1; long l2 = 0xffffffff; double d = 2.2; }
```

$\Downarrow$

```
0  bipush 100 // Push a small int
2  istore_1
3  ldc #1     // Integer 1000000; a larger int value uses ldc
5  istore_2
6  lconst_1   // A tiny long value
7  lstore_3
8  ldc2_w #6  // A long 0xffffffff. A long constant value.
11 lstore 5
13 ldc2_w #8  // Double 2.200000
16 dstore 7
```

25 JVM Example VI

```
    void whileInt() {
        int i = 0;
        while (i < 100) i++;
    }
```

$\Downarrow$

```
0   iconst_0
1   istore_1
2   goto 8
5   iinc 1 1
8   iload_1
9   bipush 100
11  if_icmplt 5
14  return
```

26 JVM Example VII

```
    int lessThan100(double d) {
        if (d < 100.0) return 1; else  return -1; }
```

$\Downarrow$

```
0   dload_1
1   ldc2_w #4 // Double 100.000000
4   dcmpg     // Push 1 if d is NaN or d < 100.000000;
              // push 0 if d == 100.000000
5   ifge 10   // Branch on 0 or 1
8   iconst_1
9   ireturn
10  iconst_m1
11  ireturn
```

27 JVM Example VIII

```
int add12and13() {return addTwo(12, 13);}
```

$\Downarrow$

```
0  aload_0          // Push this local 0 (this) onto stack
1  bipush 12        // Push int constant 12 onto stack
3  bipush 13        // Push int constant 13 onto stack
5  invokevirtual #4 // Method Example.addtwo(II)I
8  ireturn          // Return int on top of stack; it is
                    // the int result of addTwo()
```

28 JVM Example IX

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

$$\Downarrow$$

```
0  new #1           // Class java.lang.Object
3  dup
4  invokespecial #4 // Method java.lang.Object.<init>()V
7  areturn
```

29 JVM Example X

```
void createBuffer() {
   int buf[]; int bsz = 100; int val=12;
   buf = new int[bsz]; buf[10]=val;  value = buf[11]; }
```

$$\Downarrow$$

30

$$\Downarrow$$

```
0   bipush 100   // Push bsz
2   istore_2     // Store bsz in local 2
3   bipush 12    // Push val
5   istore_3     // Store val in local 3
6   iload_2      // Push bsz...
7   newarray int // and create new int array
9   astore_1     // Store new array in buf
10  aload_1      // Push buf
11  bipush 10    // Push constant 10
13  iload_3      // Push val
14  iastore      // Store val at buf[10]
15  aload_1      // Push buf
16  bipush 11    // Push constant 11
18  iaload       // Push value at buf[11]
19  istore_3     // ...and store it in value
20  return
```

31 JVM Example XI

```
int chooseNear(int i) {
switch(i){case 0:return 0; case 2:return 2; default:return -1;}}
```

$$\Downarrow$$

```
0   iload_1     // Load local 1 (argument i)
1   tableswitch 0 to 2:
    0: 28       // If i is 0, continue at 28
    1: 32       // If i is 1, continue at 34
    2: 30       // If i is 2, continue at 32
    default:34  // Otherwise, continue at 34
28  iconst_0    // i was 0; push int 0...
29  ireturn     // ...and return it
```

```
30  iconst_2    // i was 2; push int 2...
31  ireturn     // ...and return it
32  iconst_m1   // otherwise push int -1...
33  ireturn     // ...and return it
```