© Manuel Cargaleiro

Introduction to the Java Virtual Machine (JVM)

Masters in Informatics and Computing Engineering (MIEIC), 3rd Year

João M. P. Cardoso
Email: jmpc@fe.up.pt

U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

DEI DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

Examples of programming languages and their execution environments

Language ☞ IR ☞ Implementations

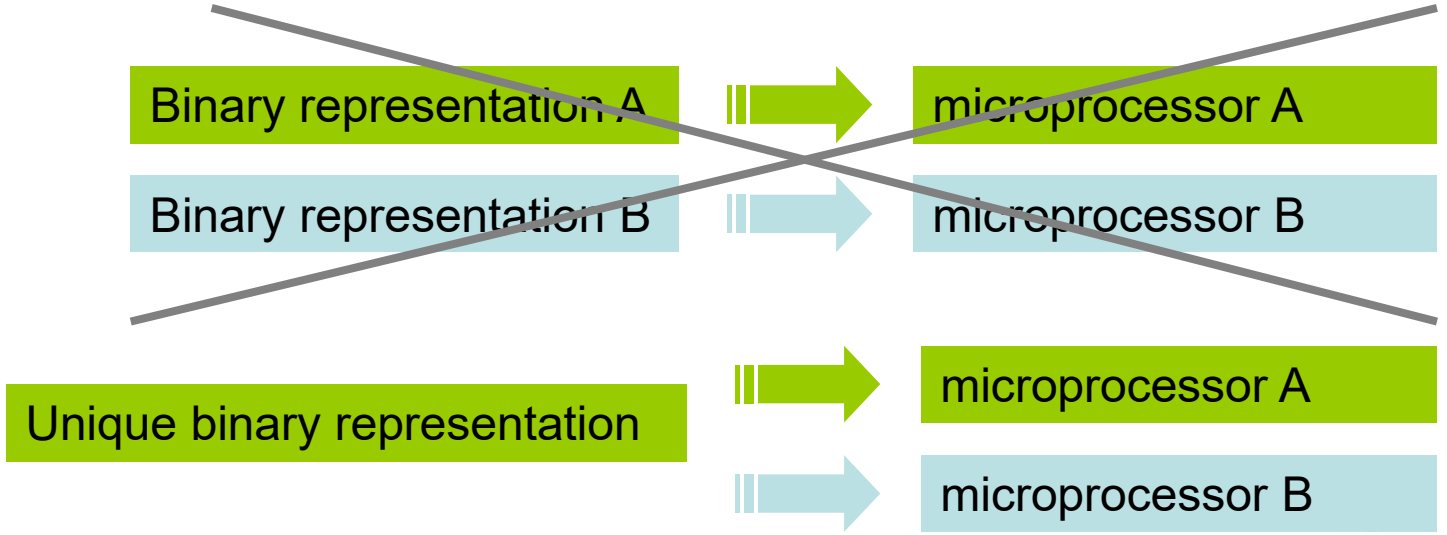
- Java ☞ JVM bytecode ☞ Interpreter, JIT
- C# ☞ MSIL ☞ JIT (but can be pre-compiled)
- Prolog ☞ WAM code ☞ compiled, interpreted
- Forth ☞ bytecode ☞ interpreted
- Smalltalk ☞ bytecode ☞ interpreted
- Pascal ☞ p-code ☞ interpreted--compiled
- C, C++ ☞ -- ☞ compiled (usually)
- Perl 6 ☞ PVM ☞ interpreted
- Parrot ☞ -- ☞ interpreted, JIT
- Python ☞ -- ☞ interpreted, JIT
- MATLAB ☞ -- ☞ interpreted, JIT

Virtual Machines

- Software layer that allows to execute in a real machine a program available in a format not specific to that machine

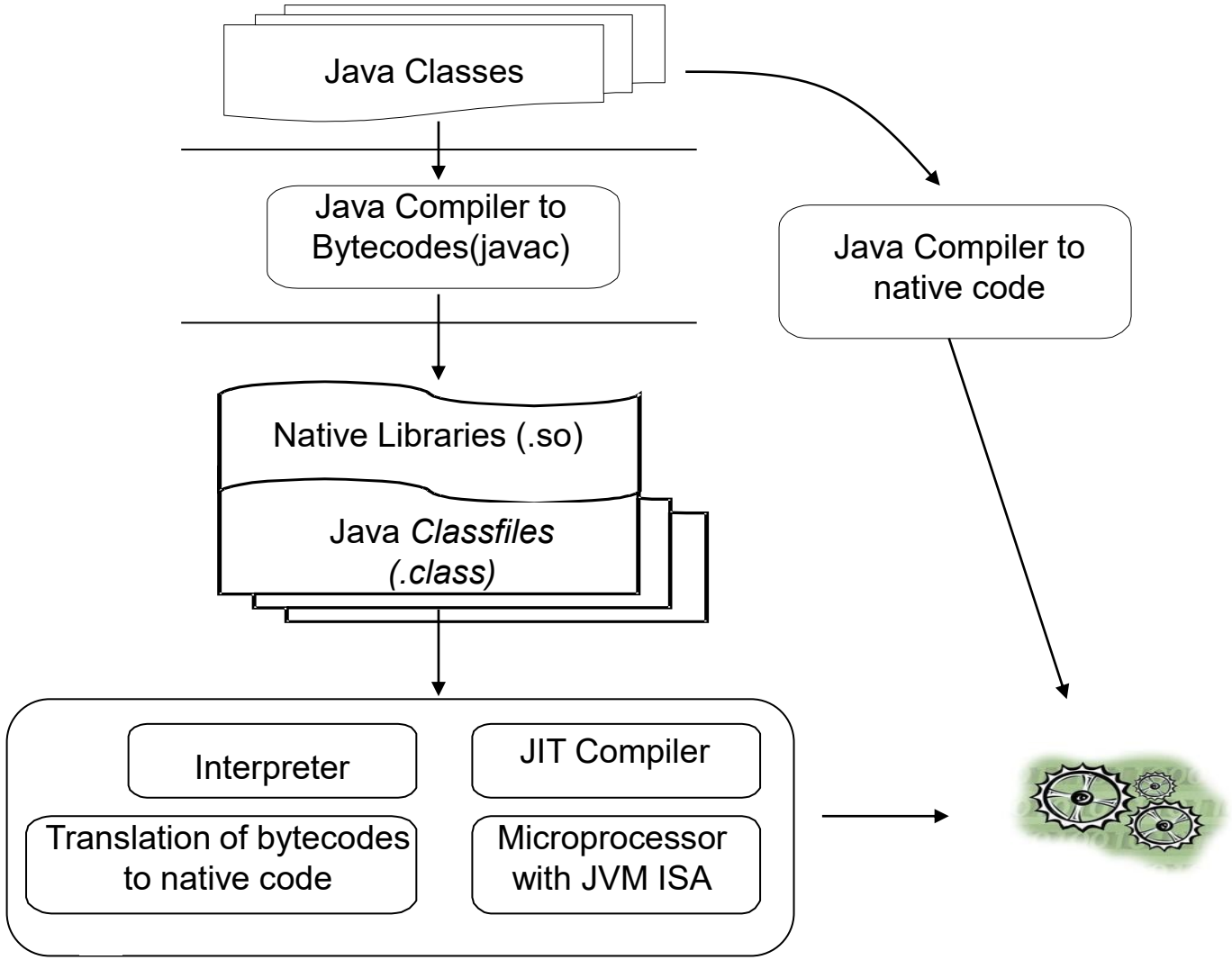
- Importance in many systems – including embedded and mobile systems
  - Guarantee the portability of applications
  - Without virtual machines, the myriad of embedded systems would make the development of applications a nightmare!

Virtual Machines



- Examples
  - JVM, Java Virtual Machine
  - CLR, Common Language Runtime

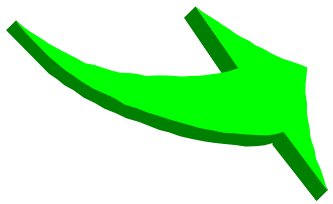
Java Technology



Java Technology

- Java
- Classfiles (include Java bytecodes)

```
class Mult {
 static int mult(int a, int b) {
  int r=0;
  for(int i=0; i<b; i++) r+=a;
  return r;
 }
}
```



```
CAFEBABE00
03002D0020
08...
```

Classfile example

```
0000 | CA FE BA BE 00 03 00 2D 00 20 08 00 1D 07 00 0E | .......-. ......
0010 | 07 00 16 07 00 1E 07 00 1C 09 00 05 00 0B 0A 00 | ................
0020 | 03 00 0A 0A 00 02 00 09 0C 00 0C 00 15 0C 00 1A | ................
0030 | 00 1F 0C 00 14 00 1B 01 00 07 70 72 69 6E 74 6C | ..........printl
0040 | 6E 01 00 0D 43 6F 6E 73 74 61 6E 74 56 61 6C 75 | n...ConstantValu
0050 | 65 01 00 13 6A 61 76 61 2F 69 6F 2F 50 72 69 6E | e...java/io/Prin
0060 | 74 53 74 72 65 61 6D 01 00 0A 45 78 63 65 70 74 | tStream...Except
0070 | 69 6F 6E 73 01 00 0F 4C 69 6E 65 4E 75 6D 62 65 | ions...LineNumbe
0080 | 72 54 61 62 6C 65 01 00 0A 53 6F 75 72 63 65 46 | rTable...SourceF
0090 | 69 6C 65 01 00 0E 4C 6F 63 61 6C 56 61 72 69 61 | ile...LocalVaria
00a0 | 62 6C 65 73 01 00 04 43 6F 64 65 01 00 03 6F 75 | bles...Code...ou
00b0 | 74 01 00 15 28 4C 6A 61 76 61 2F 6C 61 6E 67 2F | t...(Ljava/lang/
00c0 | 53 74 72 69 6E 67 3B 29 56 01 00 10 6A 61 76 61 | String;)V...java
00d0 | 2F 6C 61 6E 67 2F 4F 62 6A 65 63 74 01 00 04 6D | /lang/Object...m
00e0 | 61 69 6E 01 00 0F 48 65 6C 6C 6F 57 6F 72 6C 64 | ain...HelloWorld
00f0 | 2E 6A 61 76 61 01 00 16 28 5B 4C 6A 61 76 61 2F | .java...([Ljava/
0100 | 6C 61 6E 67 2F 53 74 72 69 6E 67 3B 29 56 01 00 | lang/String;)V..
0110 | 06 3C 69 6E 69 74 3E 01 00 15 4C 6A 61 76 61
0120 | 69 6F 2F 50 72 69 6E 74 53 74 72 65 61 6D
0130 | 00 10 6A 61 76 61 2F 6C 61 6E 67 2F 53 79
0140 | 65 6D 01 00 0C 48 65 6C 6C 6F 20 57 6F 72
0150 | 21 01 00 0A 48 65 6C 6C 6F 57 6F 72 6C 64
0160 | 03 28 29 56 00 01 00 04 00 03 00 00 00 00
0170 | 00 09 00 17 00 19 00 01 00 13 00 00 00 25 00 02 | .............%..
0180 | 00 01 00 00 00 09 B2 00 06 12 01 B6 00 08 B1 00 | ................
```

```
class HelloWorld {
    public static void main(String args[]) {
        System.out.println("Hello world!");
    }
}
```

JVM

- A JVM represents an abstract computing machine
  - A set of instructions
  - Various memory regions
- JVM Properties:
  - Stack of operands, and local variables
  - Non-orthogonal instruction set
  - All arithmetic instructions use the operand stack

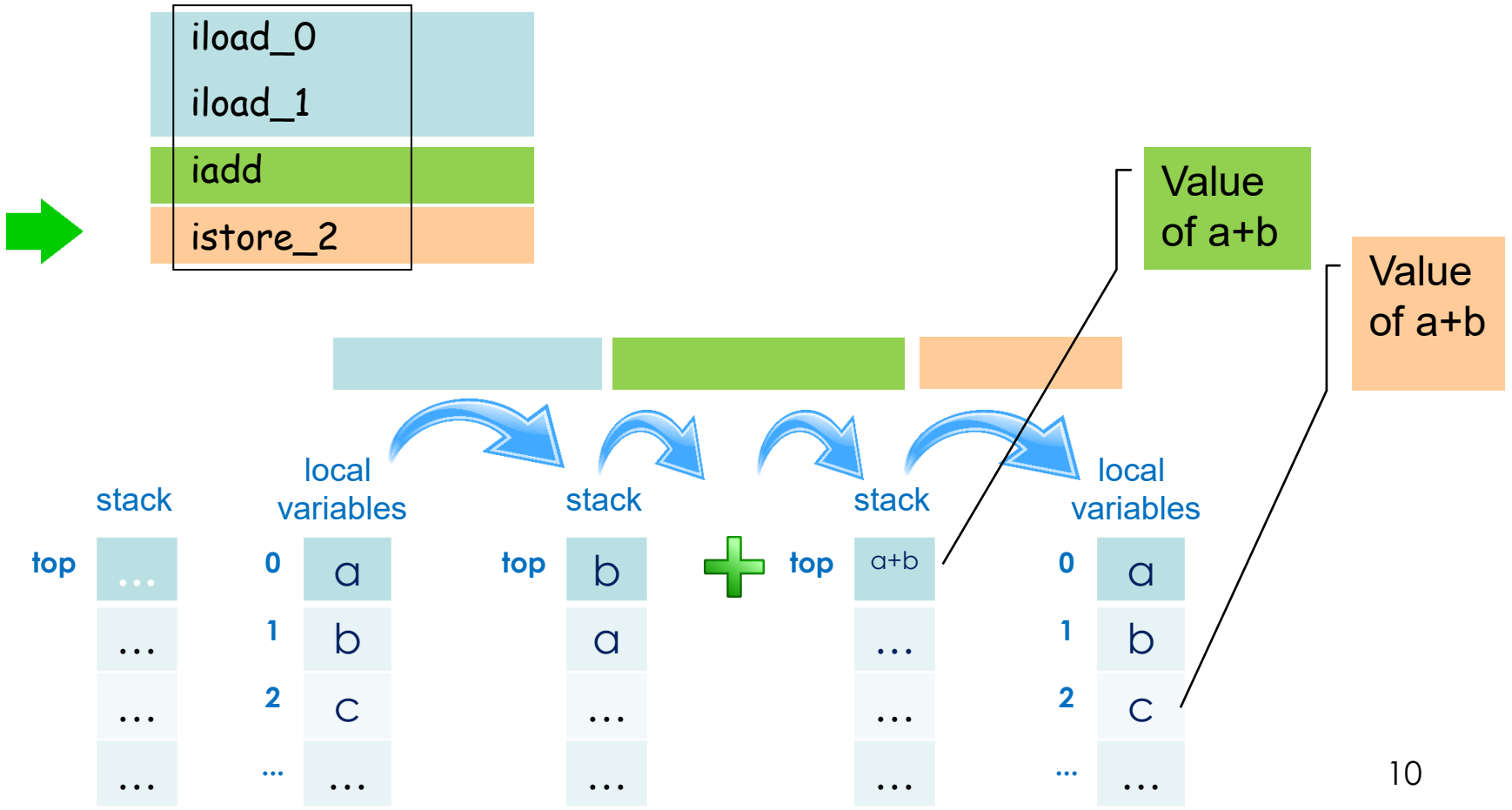
JVM

- A JVM ***allows to execute any programming language***, *as long as the programs in that language can be translated to classfiles*
- "class file"
  - JVM instructions (known as bytecodes)
  - A table of symbols
  - And additional information…

JVM

- Example:

```
void m1() {
    int a,b,c;
   ...
   c = a + b;
   ...
}
```



Bytecodes

```
public class ex {
   static int mult(int a, int b) {
      int r=0;
      for(int i=0; i<b; i++) r+=a;
      return r;
   }
}
```

- Javap –v ex.class

```
public class ex
  minor version: 0
  major version: 55
  flags: (0x0021) ACC_PUBLIC, ACC_SUPER
  this_class: #2                          // ex
  super_class: #3                         // java/lang/Object
  interfaces: 0, fields: 0, methods: 2, attributes: 1
Constant pool:
   #1 = Methodref #3.#13 // java/lang/Object."<init>":()V
   #2 = Class              #14  // ex
   #3 = Class              #15  // java/lang/Object
   #4 = Utf8               <init>
   #5 = Utf8               ()V
   #6 = Utf8               Code
   #7 = Utf8               LineNumberTable
   #8 = Utf8               mult
   #9 = Utf8               (II)I
  #10 = Utf8               StackMapTable
  #11 = Utf8               SourceFile
  #12 = Utf8               ex.java
  #13 = NameAndType  #4:#5  // "<init>":()V
  #14 = Utf8               ex
  #15 = Utf8               java/lang/Object
```

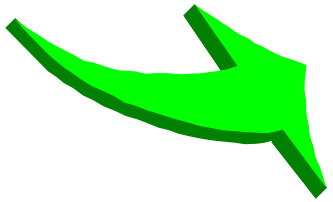
```
{
  public ex();
    descriptor: ()V
    flags: (0x0001) ACC_PUBLIC
    Code:
      stack=1, locals=1, args_size=1
         0: aload_0
         1: invokespecial #1  // Method
                    java/lang/Object."<init>":()V
         4: return
      LineNumberTable:
        line 1: 0
```

```
static int mult(int, int);
    descriptor: (II)I
    flags: (0x0008) ACC_STATIC
    Code:
      stack=2, locals=4, args_size=2
         0: iconst_0
         1: istore_2
         2: iconst_0
         3: istore_3
         4: iload_3
         5: iload_1
         6: if_icmpge     19
         9: iload_2
        10: iload_0
        11: iadd
        12: istore_2
        13: iinc          3, 1
        16: goto          4
        19: iload_2
        20: ireturn
      LineNumberTable:
        line 3: 0
        line 4: 2
        line 5: 19
      StackMapTable: number_of_entries = 2
        frame_type = 253 /* append */
          offset_delta = 4
          locals = [ int, int ]
        frame_type = 250 /* chop */
          offset_delta = 14
}
```

2 parameters => 2 local variables
LV 0: a (if it was not a static method LV 0 would be for **this**)
LV 1: b

2 local variables of the method => +2 local variables
LV 2: r
LV 3: i

```
static int mult(int a, int b) {
  int r=0;  // LV2 = 0
  for(int i=0; i<b; i++) r+=a;
  return r;
 }
```



```
static int mult(int a, int b) {
  int r=0;
  int i = 0;
loop:
  if(i>=b) goto endloop
  r+=a;
  i++
  goto loop
endloop:
  return r;
 }
```

```
static int mult(int a, int b) {
  $2=0;
  $3=0;
loop:
  if($3>=$1) goto endloop
  $2=$2+$0;
  $3++ // $3=$3+1
  goto loop
endloop:
  return $2;
 }
```

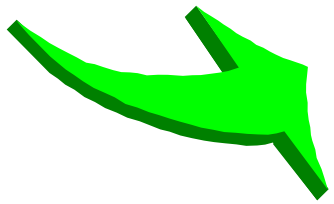
```
0:  iconst_0
1:  istore_2
2:  iconst_0
3:  istore_3
4:  iload_3
5:  iload_1
6:  if_icmpge 19
9:  iload_2
10: iload_0
11: iadd
12: istore_2
13: iinc   3, 1
16: goto   4
19: iload_2
20: ireturn
```

Local JVM variables, and the variables of the method stored:
0 ➔ a
1 ➔ b
2 ➔ r
3 ➔ i

```
// $3=$3+1
iconst_1
iload_3
iadd
istore_3
```

Bytecodes

```
static int mult(int a, int b) {
  int r=0;
  for(int i=0; i<b; i++) r+=a;
  return r;
 }
```

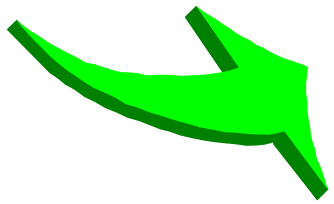


```
0:  iconst_0
1:  istore_2
2:  iconst_0
3:  istore_3
4:  iload_3
5:  iload_1
6:  if_icmpge 19
9:  iload_2
10: iload_0
11: iadd
12: istore_2
13: iinc    3, 1
16: goto    4
19: iload_2
20: ireturn
```

Local JVM variables, and the variables of the method stored:
0 ➔ a
1 ➔ b
2 ➔ r
3 ➔ i

Bytecodes

```
static int mult(int a, int b) {
  int r=0;
  for(int i=0; i<b; i++) r+=a;
  return r;
}
```



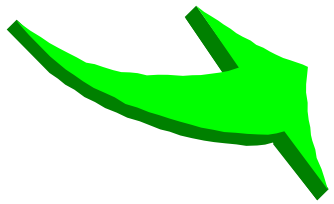
```
0:  iconst_0
1:  istore_2
2:  iconst_0
3:  istore_3
4:  iload_3
5:  iload_1
6:  if_icmpge 19
9:  iload_2
10: iload_0
11: iadd
12: istore_2
13: iinc   3, 1
16: goto   4
19: iload_2
20: ireturn
```

Local JVM variables, and the variables of the method stored:

0 ➔ a
1 ➔ b
2 ➔ r
3 ➔ i

Bytecodes

```
static int mult(int a, int b) {
  int r=0;
  for(int i=0; i<b; i++) r+=a;
  return r;
 }
```

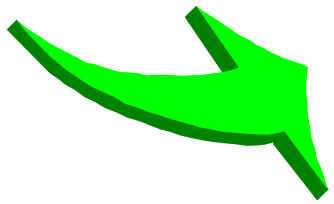


```
0:  iconst_0
1:  istore_2
2:  iconst_0
3:  istore_3
4:  iload_3
5:  iload_1
6:  if_icmpge 19
9:  iload_2
10: iload_0
11: iadd
12: istore_2
13: iinc    3, 1
16: goto    4
19: iload_2
20: ireturn
```

Local JVM variables, and the variables of the method stored:
0 ➔ a
1 ➔ b
2 ➔ r
3 ➔ i

Bytecodes

```
static int mult(int a, int b) {
  int r=0;
  for(int i=0; i<b; i++) r+=a;
  return r;
 }
```

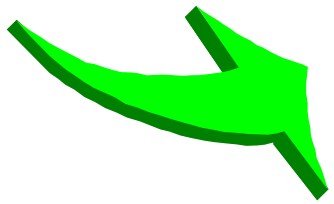


```
0:  iconst_0
1:  istore_2
2:  iconst_0
3:  istore_3
4:  iload_3
5:  iload_1
6:  if_icmpge 19
9:  iload_2
10: iload_0
11: iadd
12: istore_2
13: iinc   3, 1
16: goto   4
19: iload_2
20: ireturn
```

Local JVM variables, and the variables of the method stored:
0 ➔ a
1 ➔ b
2 ➔ r
3 ➔ i

Bytecodes

```
static int mult(int a, int b) {
  int r=0;
  for(int i=0; i<b; i++) r+=a;
  return r;
 }
```



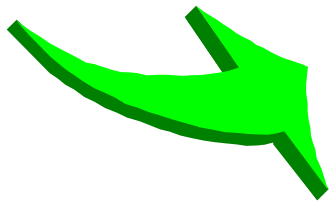
```
0:  iconst_0
1:  istore_2
2:  iconst_0
3:  istore_3
4:  iload_3
5:  iload_1
6:  if_icmpge 19
9:  iload_2
10:  iload_0
11:  iadd
12:  istore_2
13:  iinc    3, 1
16:  goto    4
19:  iload_2
20:  ireturn
```

Local JVM variables, and the variables of the method stored:

0 ➔ a
1 ➔ b
2 ➔ r
3 ➔ i

Bytecodes

```
static int mult(int a, int b) {
  int r=0;
  for(int i=0; i<b; i++) r+=a;
  return r;
 }
```



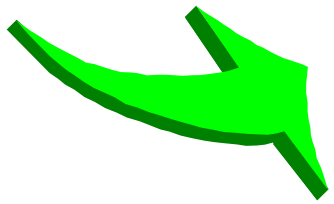
```
0:  iconst_0
1:  istore_2
2:  iconst_0
3:  istore_3
4:  iload_3
5:  iload_1
6:  if_icmpge 19
9:  iload_2
10: iload_0
11: iadd
12: istore_2
13: iinc    3, 1
16: goto    4
19: iload_2
20: ireturn
```

Local JVM variables, and the variables of the method stored:

0 ➔ a
1 ➔ b
2 ➔ r
3 ➔ i

Bytecodes

```
static int mult(int a, int b) {
  int r=0;
  for(int i=0; i<b; i++) r+=a;
  return r;
 }
```



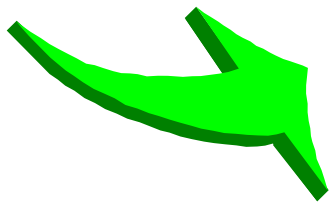
```
0:  iconst_0
1:  istore_2
2:  iconst_0
3:  istore_3
4:  iload_3
5:  iload_1
6:  if_icmpge 19
9:  iload_2
10:  iload_0
11:  iadd
12:  istore_2
13:  iinc    3, 1
16:  goto    4
19:  iload_2
20:  ireturn
```

Local JVM variables, and the variables of the method stored:

0 ➔ a
1 ➔ b
2 ➔ r
3 ➔ i

Bytecodes

```
static int mult(int a, int b) {
  int r=0;
  for(int i=0; i<b; i++) r+=a;
  return r;
}
```



```
0:  iconst_0
1:  istore_2
2:  iconst_0
3:  istore_3
4:  iload_3
5:  iload_1
6:  if_icmpge 19
9:  iload_2
10: iload_0
11: iadd
12: istore_2
13: iinc   3, 1
16: goto   4
19: iload_2
20: ireturn
```

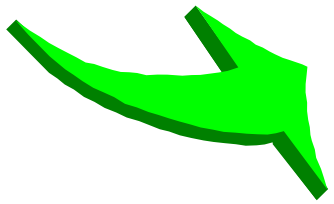
Local JVM variables, and the variables of the method stored:

0 ➔ a
1 ➔ b
2 ➔ r
3 ➔ i

Bytecodes

- Another example:

```
…
// Number of array elements N
int N=4;

int L2NORM = 0;

for(int i=0; i<N;i++) {
  short Aux = X[i] - Y[i];
  L2NORM += Aux*Aux;
}
…
```

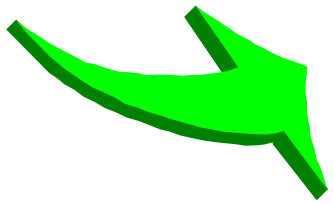


```
-------
1       iconst_0
2       istore_2
3       iconst_0
4       istore_3
5       goto 28
-------
6       aload_0
7       iload_3
8       saload
-------
9       aload_1
10       iload_3
11       saload
-------
12       isub
13       i2s
14       istore 4
15       iload_2
16       iload 4
17       iload 4
18       imul
19       iadd
20       istore_2
21       iinc 3 1
-------
22       iload_3
23       iconst_4
24       if_icmplt 7
-------
25       iload_2
26       ireturn
```

Bytecodes

- Another example:

```
…
// Number of array elements N
int N=4;

int L2NORM = 0;

for(int i=0; i<N;i++) {
  short Aux = X[i] - Y[i];
  L2NORM += Aux*Aux;
}
…
```



```
-------
1       iconst_0
2       istore_2
3       iconst_0
4       istore_3
5       goto 28
-------
6       aload_0
7       iload_3
8       saload
-------
9       aload_1
10      iload_3
11      saload
-------
12      isub
13      i2s
14      istore 4
15      iload_2
16      iload 4
17      iload 4
18      imul
19      iadd
20      istore_2
21      iinc 3 1
-------
22      iload_3
23      iconst_4
24      if_icmplt 7
-------
25      iload_2
26      ireturn
```

Bytecodes

- Type of instructions:
  - Load/store from/to a local variable (*iload_1*, *fload 4*, *istore 6*, etc.)
  - Operand stack manipulation (*pop*, *dup*, etc.)
  - Arithmetic, logical, and conversions (*iadd*, *fsub*, *ineg*, *i2s*, *d2f*, etc.)
  - Conditional and unconditional branches (*if_icmplt*, *ifne*, *goto*, *jsr*, etc.)
  - Comparisons (*lcmp*, etc.)
  - Creation and manipulation of objects and *arrays* (*new*, *newarray*, etc.)
  - Access fields of an object (*putfield*, *getfield*, etc.)
  - Invocation of methods (*invokeinterface*, *invokespecial*, etc.)
  - Constants (*iconst_1*, *ldc* "Hello", etc.)
  - *Switch* (*lookupswitch* e *tableswitch*)
  - Definition of synchronous regions (*monitorenter* e *monitorexit*)

Classfile

- Binary representation of the class from which it originated

```
ClassFile {
    signature data
    constant pool
    inheritance information (superclass and interface(s))
    field information
    method information
    attributes
}
```

```
CAFEBABE00
03002D0020
08...
```

Data in Classfiles

Constant Pool (symbol table) + *Bytecodes* of each method + …

```
#1 = Method       #3.#12; //
java/lang/Object."<init>":()V
#2 = class        #13;    //  Mult
#3 = class        #14;    //
java/lang/Object
#4 = Asciz        <init>;
#5 = Asciz        ()V;
#6 = Asciz        Code;
#7 = Asciz        LineNumberTable;
#8 = Asciz        mult;
#9 = Asciz        (II)I;
#10 = Asciz       SourceFile;
#11 = Asciz       Mult.java;
#12 = NameAndType #4:#5;//
"<init>":()V
#13 = Asciz       Mult;
#14 = Asciz       java/lang/Object;
```

```
static int mult(int, int);
0:   iconst_0
1:   istore_2
2:   iconst_0
3:   istore_3
4:   iload_3
5:   iload_1
6:   if_icmpge 19
9:   iload_2
10:  iload_0
11:  iadd
12:  istore_2
13:  iinc    3, 1
16:  goto    4
19:  iload_2
20:  ireturn
```

```
method mult(II)I
Stack=2,
Locals=4,
Args_size=2
code_length = 21

LineNumberTable:
 line 5: 0
 line 6: 2
 line 7: 19
```

```
1.
2. class Mult {
3.
4. static int mult(int a, int b) {
5.   int r=0;
6.   for(int i=0; i<b; i++) r+=a;
7.   return r;
8. }
9. }
```

Data in Classfiles

Constant Pool (symbol table) + *Bytecodes* of each method + …

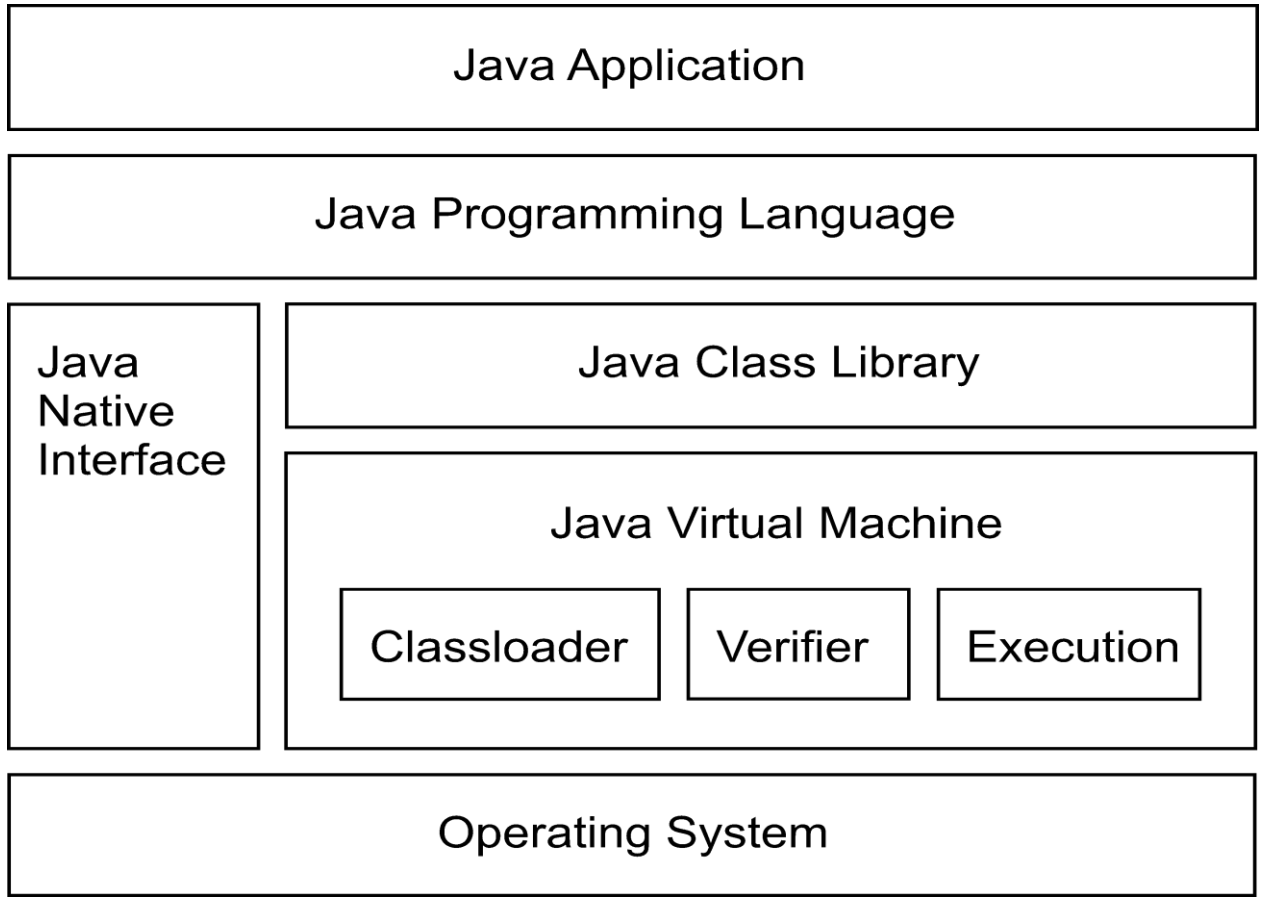
```
#1 = Method       #3.#12; //
java/lang/Object."<init>":()V
#2 = class        #13;    //  Mult
#3 = class        #14;    //
java/lang/Object
#4 = Asciz        <init>;
#5 = Asciz        ()V;
#6 = Asciz        Code;
#7 = Asciz        LineNumberTable;
#8 = Asciz        mult;
#9 = Asciz        (II)I;
#10 = Asciz       SourceFile;
#11 = Asciz       Mult.java;
#12 = NameAndType #4:#5;//
"<init>":()V
#13 = Asciz       Mult;
#14 = Asciz       java/lang/Object;
```

```
Mult()
0:   aload_0
1:   invokespecial
#1; //Method
java/lang/Object."<in
it>":()V
4:   return
```

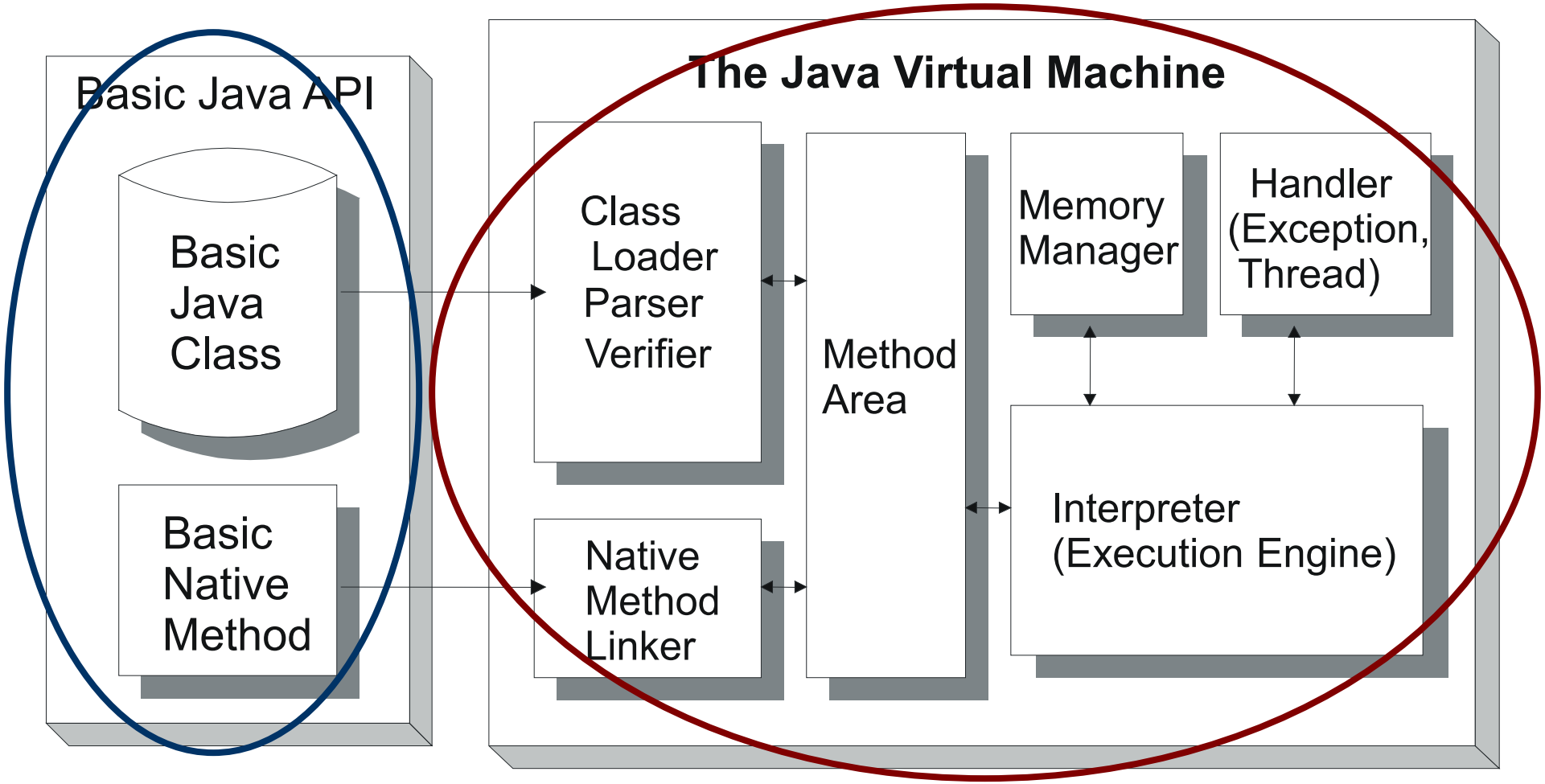
```
method Mult()
Stack=1
Locals=1
Args_size=1
code_length = 5

LineNumberTable:
  line 2: 0
```

```
1.
2. class Mult {
3.
4. static int mult(int a, int b) {
5.   int r=0;
6.   for(int i=0; i<b; i++) r+=a;
7.   return r;
8. }
9. }
```

Java Systems



JVM Structures



Additional Details

THE JAVA VIRTUAL MACHINE (JVM)

Data Types

- *Primitive types*
  - integer
    - signed, two's-complement integers
      - Byte (8-bit); short (16-bit); int (32-bit); long (64-bit)
  - char, 16-bit unsigned integers representing Unicode version 1.1.5 characters
  - IEEE 754 standard floating-point types:
    - float (32-bit); double (64-bit)

And the boolean?

Data Types

- *Reference types:*
  - class types
  - interface types
  - array types
- A reference can have as value the special reference **null**

Data storage

- Assigned space using **word** as unit
  - 32-bit in a 32 bit machine
  - 64-bit in a 64 bit machine
- A word stores a value of type
  - byte, char, short, int, float, reference, returnAddress, or a native pointer
- Uses two words to store data with types
  - long and double

Creation of Objects and Manipulation

- Creation of an instance of a class:
  - **new**
- Creation of a new array:
  - **newarray**, **anewarray**, **multianewarray**
- Access to fields of class and of class instances:
  - **getfield**, **putfield**, **getstatic**, **putstatic**

Creation of Objects and Manipulation (cont.)

- Load of an array element to the operand stack
  - **baload, caload, saload, iaload, laload, faload, daload, aaload**
- Store of a value in the stack to na elemento of an array:
  - **bastore, castore, sastore, iastore, lastore, fastore, dastore, aastore.**
- Size of an array:
  - **arraylength**.
- Verify the properties of class instances or of arrays:
  - **instanceof, checkcast**

Method invocation

- invokevirtual
  - Invokes a method of an object: resolved by *this* or by dynamic dispatching
- invokeinterface
  - Invokes a method implemented by an interface: the search of the method is done based on object (*this*) at runtime
- invokespecial
  - Invokes a method of na instance that needs special treatment: a method to nitialize an instance <init>, a private method, or a method of the superclass
- invokestatic
  - Invokes a method of a class (*static*): uses the name of the class

Return from Methods

- The return instructions are distinguished by the type:
  - ireturn (values of type: byte, char, short, ou int)
  - lreturn
  - freturn
  - dreturn
  - areturn
  - return (for methods declared as void)

JVM Instructions' Details

Summary of instruction set

- A JVM instruction consists in:
  - An opcode of 1 byte that specifies the operation
  - Zero or more operands that furnish the operands used by the operation
    - The arguments can identify local variables, constants, or other arguments via references to the *constant pool*
- Main interpreter actions

```
do {
    fetch an opcode;
    if (operands) fetch operands;
    execute the action for the opcode;
} while (there is more to do);
```

JVM and Types

- Many JVM instructions are grouped with variations according to the data types
  - iadd ladd fadd dadd
    (variations of *addition*)
- *Convention uses the first letter of the mnemonic of the instruction to represente the type*

type	code
int	i
long	l
float	f
double	d
byte	b
char	c
short	s
reference	a

Load and Store Instructions

- Load: transfer values between operand stack and local variables:
  - iload, iload_0, iload_1, ..., aload
- Store: transfer values between local variables and operand stack:
  - .... istore, fstore,..., astore,…
- Load a constant to the operand stack:
  - bipush, sipush, ldc, ldc_w, ldc2_w, aconst_null,...

JVM: *try-catch*

- Try-catch is converted to an exception table

```
void catchOne() {
    try {
        tryItOut();
    } catch (TestExc e) {
        handleExc(e);
    }
}
```

```
Method void catchOne()
0 aload_0 // Beginning of try block
1 invokevirtual #6 // Method Example.tryItOut()V
4 return // End of try block; normal return
5 astore_1 // Store thrown value in local variable 1
6 aload_0 // Push this
7 aload_1 // Push thrown value
8 invokevirtual #5 // Invoke handler method:
                   // Example.handleExc(LTestExc;)V
11 return // Return after handling TestExc
Exception table:
From To Target Type
 0   4   5   Class TestExc
```

Arithmetic Instructions

- Typically, *pop* 2 operands in the top of stack and *push* the result to the stack
  - Add: iadd, ladd, fadd, dadd
  - Subtract: isub, lsub, fsub, dsub
  - Multiply: imul, lmul, fmul, dmul
  - Divide: idiv, ldiv, fdiv, ddiv
  - Remainder: irem, lrem, frem, drem
  - Negate: ineg, lneg, fneg, dneg
  - Shift: ishl, ishr, iushr, lshl, lshr, lushr
  - Bitwise OR, AND, OR exclusivo: ior, lor; iand, land; ixor, lxor
  - Increment (can add negative numbers: decrement)of local variables: iinc

Instructions to Convery Types

- Conversions “up”:
  - int to long, float, or double (i2l, i2f, i2d)
  - long to float or double (l2f, l2d)
  - float to double (f2d)
- Conversions “down”:
  - int to byte, short, or char (i2b, i2c, i2s)
  - long to int (l2i)
  - float to int or long (f2i, f2l)
  - double to int, long, or float (d2i, d2l, and d2f)

Access to Fields

- Example: getstatic 178 (0xb2)
  - Operation: push to stack the value of a static field
  - Format:
    getstatic (opcode)
    indexbyte1 (operands)
    indexbyte2
- Indexbyte1 and indexbyte2 are 2 bytes to index (`(indexbyte1 << 8) | indexbyte2`) *constant pool* itens of the class
  - Name of class

Creation of Objects

- new, 187 (0xbb)
  - Operation: creates a new object (teh address of the object is pushed to the stack)

  new

  indexbyte1

  indexbyte2
- Indexbyte1 and indexbyte2 are 2 bytes to index the *constant pool*
  - Name of classe

Control Instructions

- Conditional branches:
  - ifeq, iflt, ifle, ifne, ifgt, ifge, ifnull, ifnonnull, if_icmpeq, if_icmpne, if_icmplt, if_icmpgt, if_icmple, if_icmpge, if_acmpeq, if_acmpne, lcmp, fcmpl, fcmpg, dcmpl, dcmpg.
- Conditional branches with multiple targets:
  - tableswitch, lookupswitch.
- Inconditional branches:
  - goto, goto_w, jsr, jsr_w, ret.

Other instructions

- About 230 instructions
- Manipulation of stack (e.g., switch of operands in the top, duplication of elements)
- Synchronization – monitorenter, ...
- Exceptions – athrow
- *I/O?*
  - *No!*
  - *Native methods of a given class*

JVM Instructions: summary

Category	No.	Example
arithmetic operation	24	iadd, lsub, frem
logical operation	12	iand, lor, ishl
numeric conversion	15	int2short, f2l, d2l
pushing constant	20	bipush, sipush, ldc, iconst_0, fconst_1
stack manipulation	9	pop, pop2, dup, dup2
flow control instructions	28	goto, ifne, ifge, if_null, jsr, ret
managing local variables	52	astore, istore, aload, iload, aload_0
manipulating arrays	17	aastore, bastore, aaload, baload
creating objects and array	4	new, newarray, anewarray, multianewarry
object manipulation	6	getfield, putfield, getstatic, putstatic
method call and return	10	invokevirtual, invokestatic, areturn
miscellaneous	5	throw, monitorenter, breakpoint, nop

JVM Verifications

- Analyzes the bytecodes and verifies if they are according to predefined rules
  - Magic number 0xCAFEBABE
  - Size of the file
  - Valid bytecodes
  - Organization according to rules
  - Use of data types according to rules
  - Valid use of local variables
  - No *overflow* of stack operands
  - Verifies consistency of the stack (stack content must have the same number and type of operands independently of the of the instruction executed before the actual instruction)
  - Arguments of methods are of valid types
  - Etc.

JVM Verifications (cont.)

- Verifies at runtime if there is consistency
  - Branch to the bytecodes of the method being executed,
  - Access to visible methods,
  - Indexing of arrays accoring to the boundary limits of the array,
  - etc.

Resources

- Official Java Language and Virtual Machine Specifications: https://docs.oracle.com/javase/specs/
  - The Java Virtual Machine Specification, Java SE 14 Edition
    - Html: https://docs.oracle.com/javase/specs/jvms/se14/html/index.html
    - PDF: https://docs.oracle.com/javase/specs/jvms/se14/jvms14.pdf
    - See CHAPTER 6: The Java Virtual Machine Instruction Set, page 397

Java Virtual Machine (JVM)

END