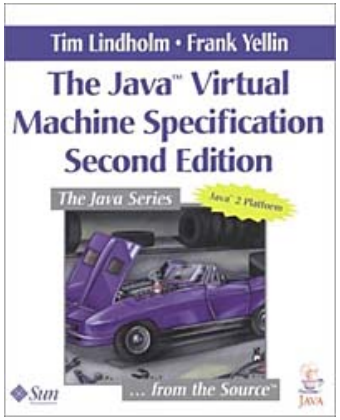
Programming Language Systems



Instructors: Taiichi Yuasa and Masahiro Yasugi

Course Description (overview, purpose):

The course provides an introduction to run-time mechanisms such as memory allocation, garbage collection, virtual machines, code generation, and exception handling that are required to implement programming language processors.

Course Schedule:

The following topics are covered, each in two to four lectures.

1. JVM execution model: The execution model of the virtual machine JVM is introduced together with actual Java systems.
2. JVM instruction set: Selected instructions of JVM are explained with their relation to Java code and to the execution model.
3. Translation to JVM: Compilation techniques for translating Java programs to JVM code sequences are presented.
4. Garbage collection: Various methods are introduced for garbage collection which is inevitable for program execution.
5. Optimization: Various techniques are introduced for optimization at compile time, at load time, and at run time.

Prerequisites and Course Requirements:

Students are expected to have some prior knowledge of programming languages, compilers, and computer architecture.

Grading Methods and Evaluation Criteria :

Students are required to submit reports on some subjects that will be given during lectures. These reports are used to judge how much each student has mastered the structure and implementation of programming language systems.

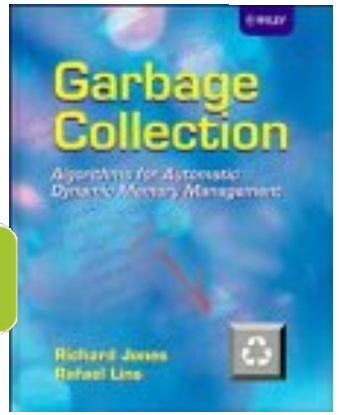
Textbooks: N/A

References :

- Jon Meyer and Troy Downing: Java Virtual Machine (O'Reilly).
- Richard Jones and Rafael Lins: Garbage Collection (Wiley).

Miscellaneous :

Course materials can be downloaded from the course web page that will be announced during the first lecture.



Programming Languages, Adv.

Instructors: Masahiro Yasugi and Taiichi Yuasa

Course Description (overview, purpose):

The course covers programming language topics such as syntax and semantics, functionality and its implementation, the object-oriented paradigm, type systems, and language features for concurrency and exception

Course Schedule:

1. Programming languages and virtual machines (2 or 3 lectures): The first part of this course explains programming languages and virtual machines, interpreters, and translators.
2. The object-oriented paradigm and a brief introduction to Java (2 or 3 lectures): The second part of this course explains the object-oriented paradigm and a brief introduction to Java.
3. The object-oriented paradigm and type systems (4 or 5 lectures): The third part of this course explains class inheritance, overloading, late binding, multiple inheritance, delegation, abstract classes, interfaces, and subtyping.
4. Various language features (4 or 5 lectures): The fourth part of this course explains exception handling, concurrency (multi-threading), generics, and interaction with garbage collectors (finalization and weak references).

Prerequisites and Course Requirements:

Students are expected to have some prior knowledge of programming languages, including the C programming language.

Grading Methods and Evaluation Criteria:

The evaluation is based on submitted reports and an examination. The goal of the course is to acquire the ability to understand and think the design of language specifications.

Textbooks: N/A

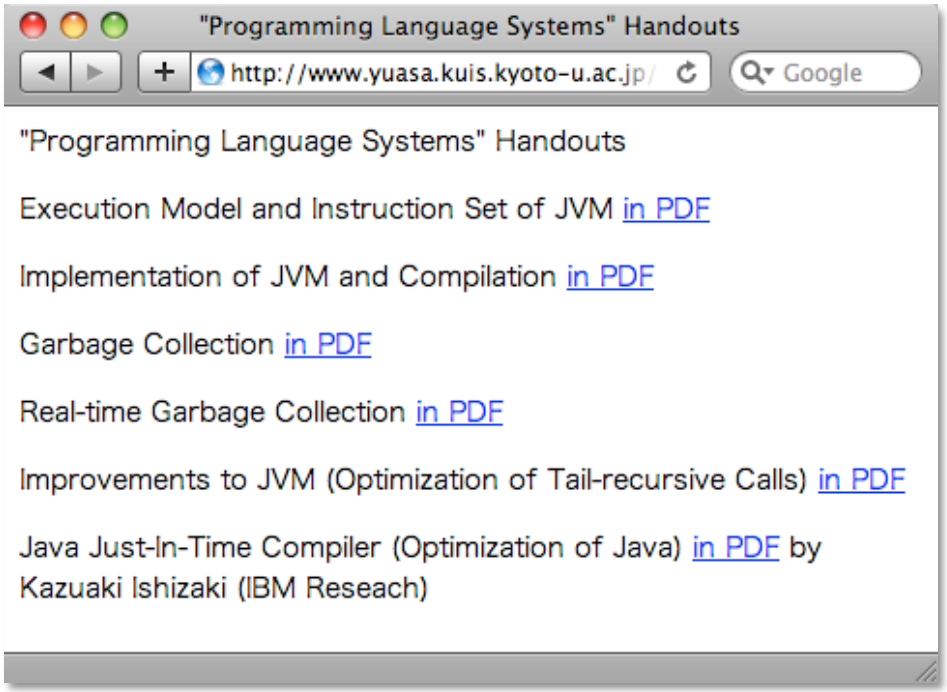
References : N/A

Web page for download

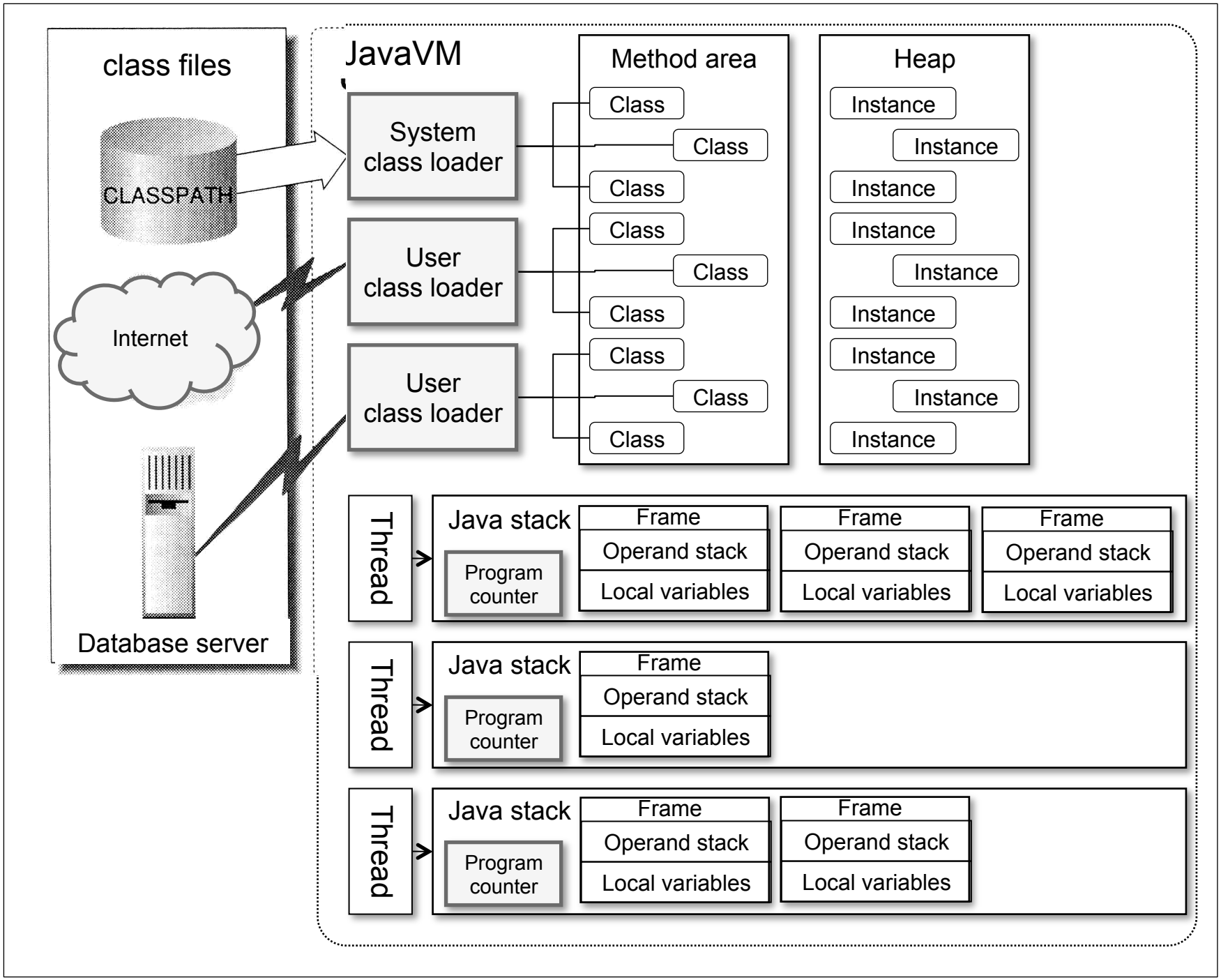
URL: http://ryujin.kuis.kyoto-u.ac.jp/~yuasa/index.html

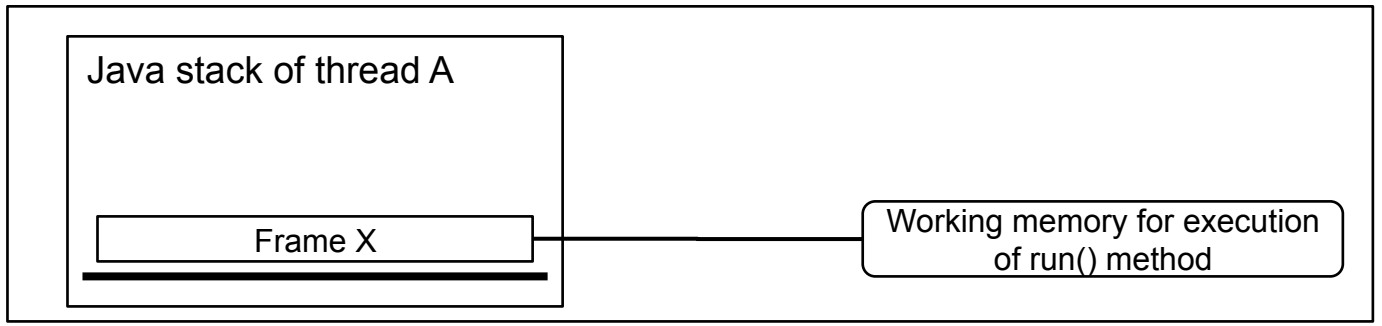
"Programming Language Systems" Handouts (password required)

User name: language-system
Password: system-language

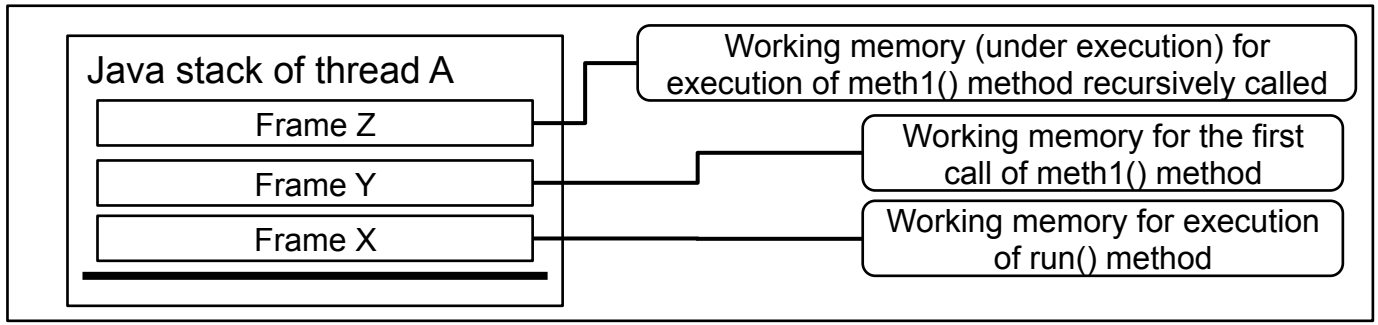


Java Virtual Machine

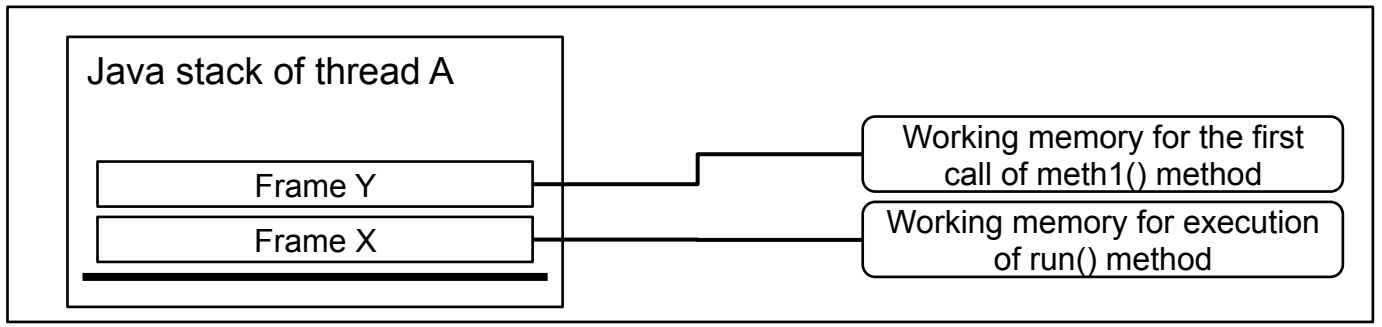

class files
CLASSPATH
Internet
Database server
JavaVM
System class loader
User class loader
User class loader
Method area
Class
Class
Class
Class
Class
Class
Class
Class
Heap
Instance
Instance
Instance
Instance
Instance
Instance
Instance
Instance
Thread
Java stack
Program counter
Frame
Operand stack
Local variables
Frame
Operand stack
Local variables
Frame
Operand stack
Local variables
Thread
Java stack
Program counter
Frame
Operand stack
Local variables
Thread
Java stack
Program counter
Frame
Operand stack
Local variables
Frame
Operand stack
Local variables



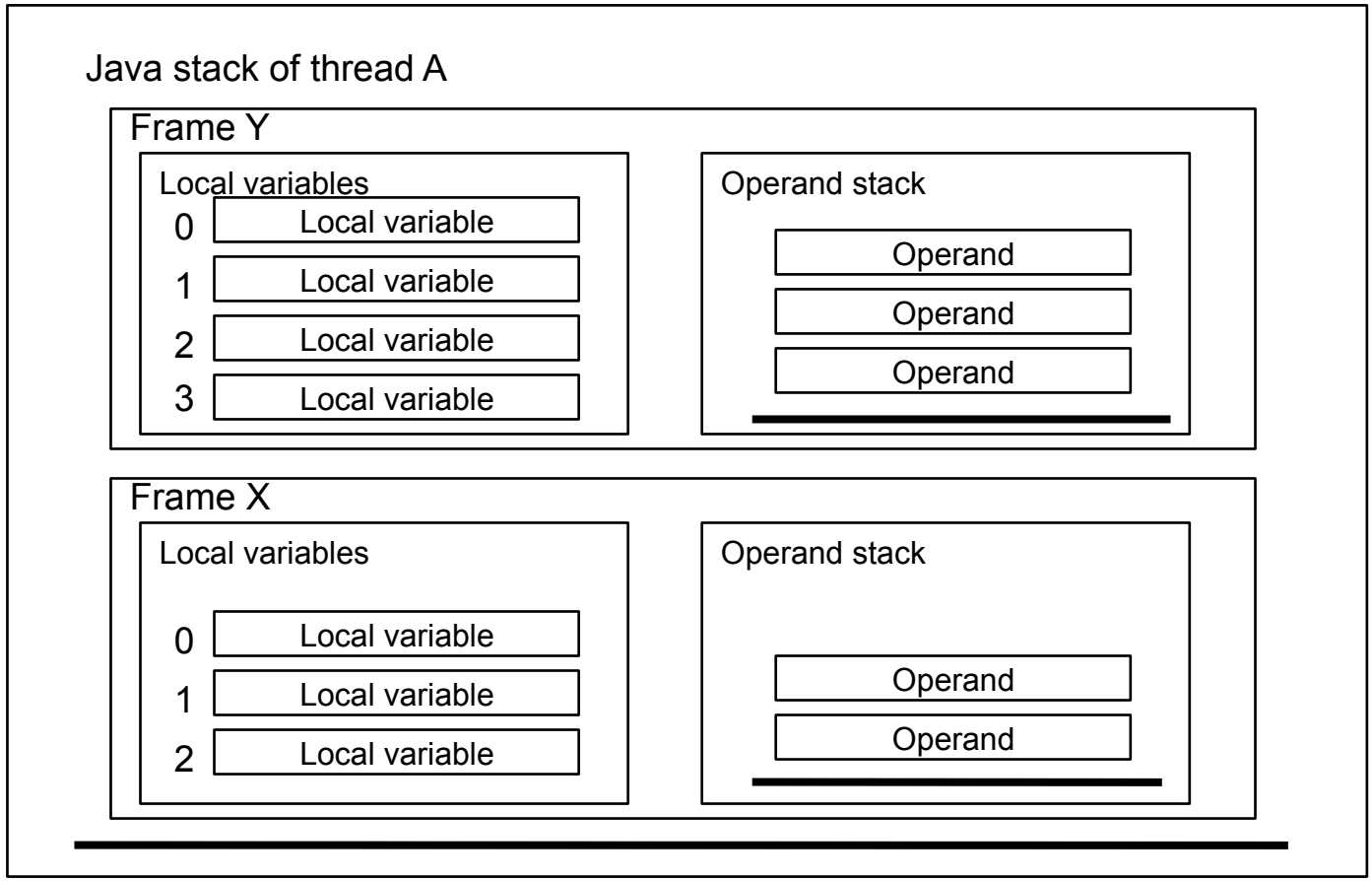
Java stack of thread A when run() is executed



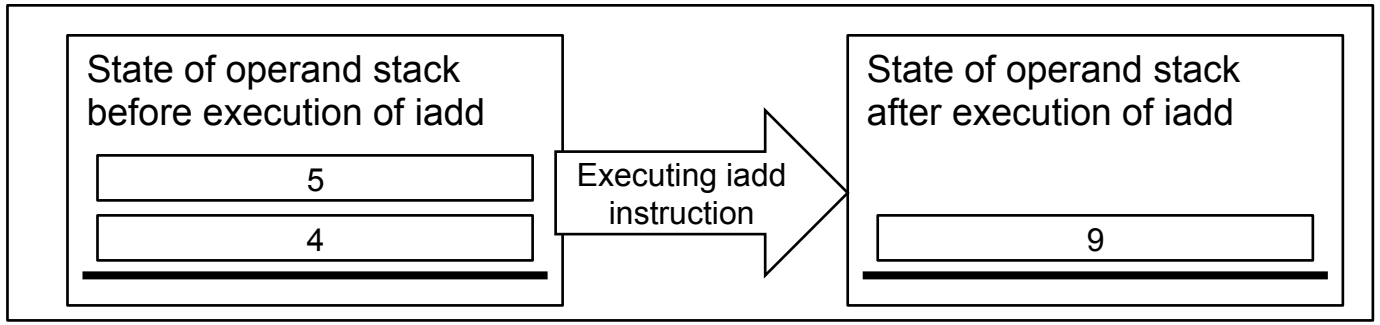
Java stack of thread A when meth1() is recursively executed



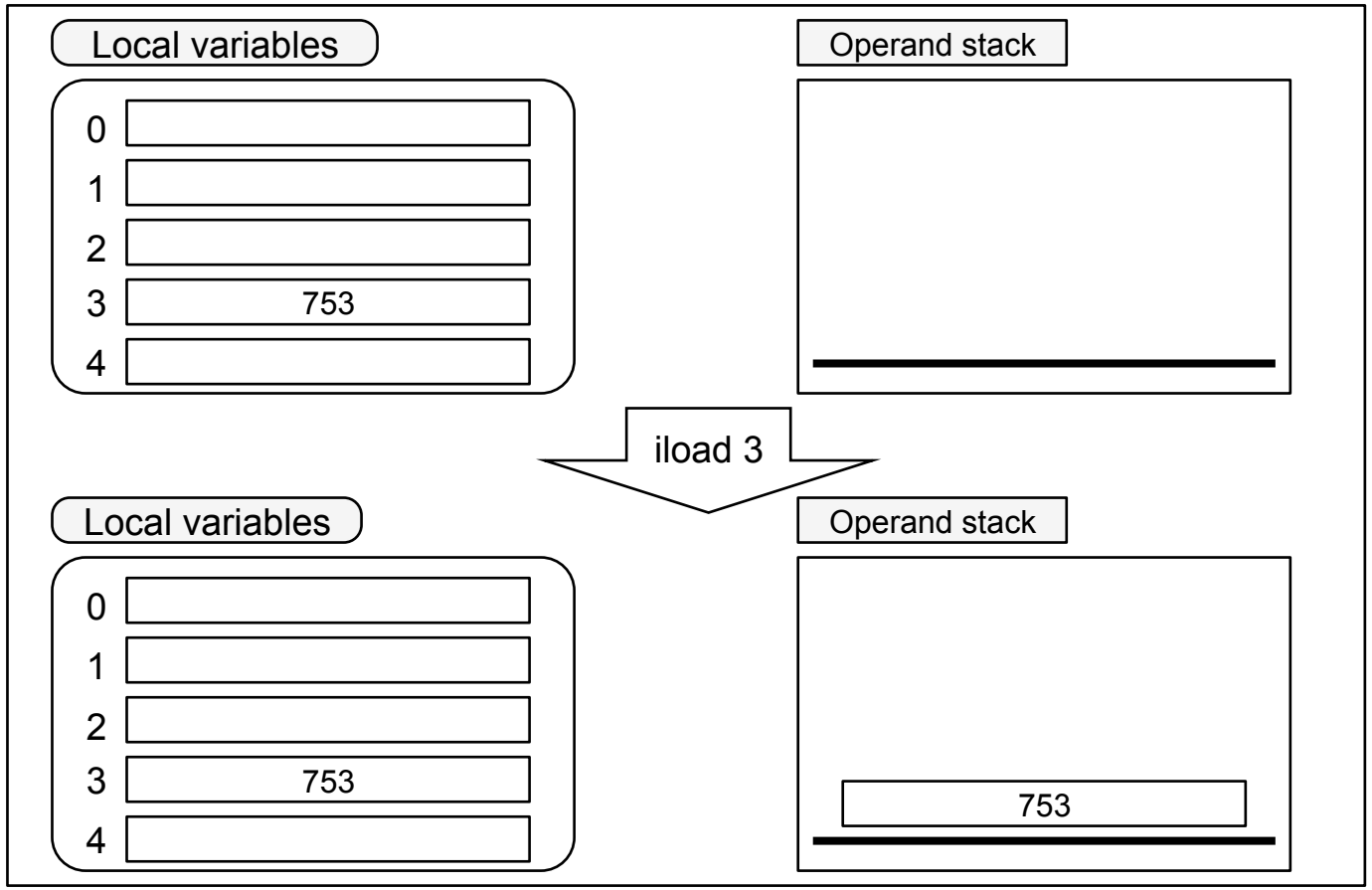
Java stack of thread A immediately after completing the execution of meth1() recursively called



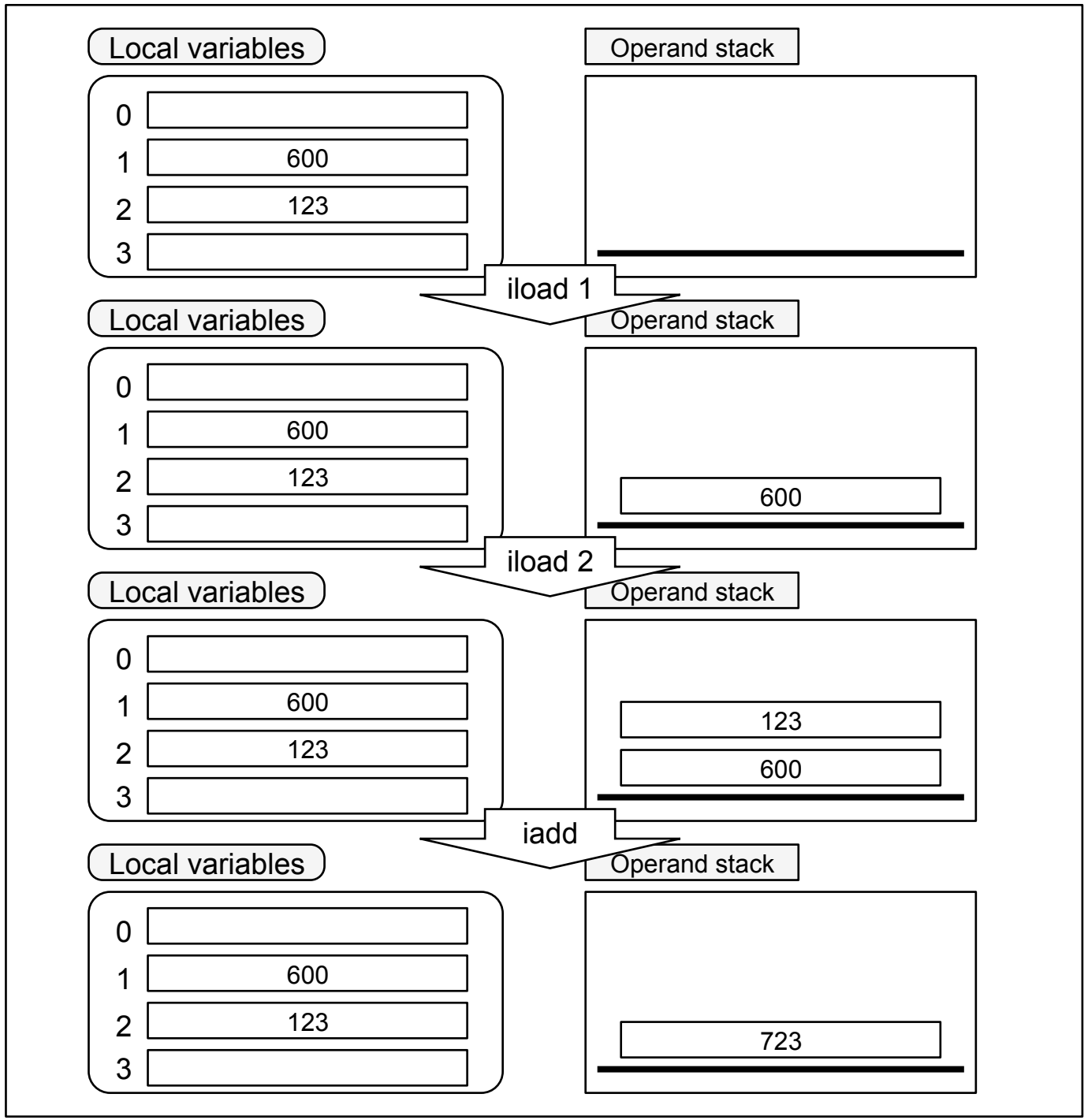
Relationship between Java stack and operand stack

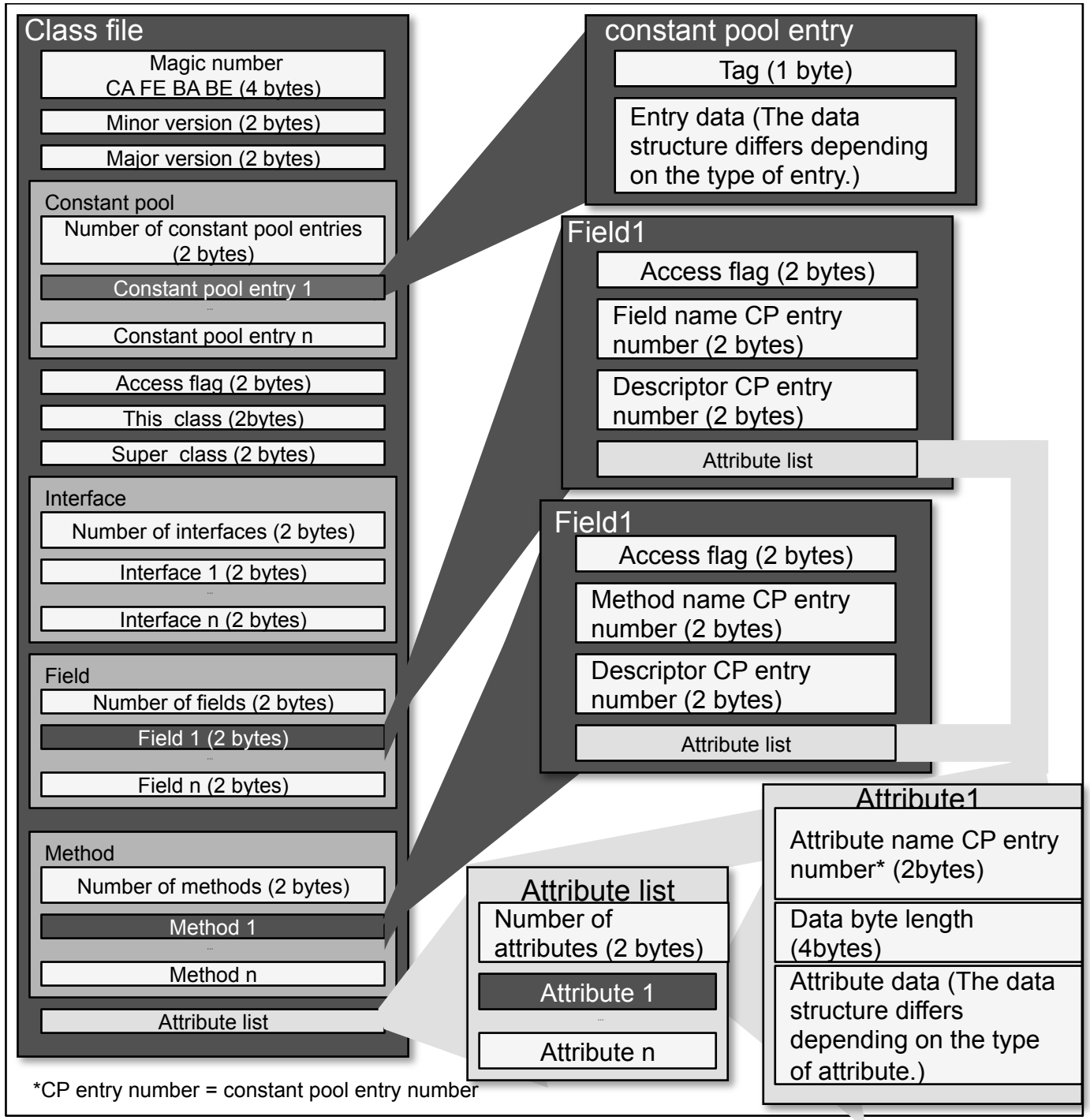


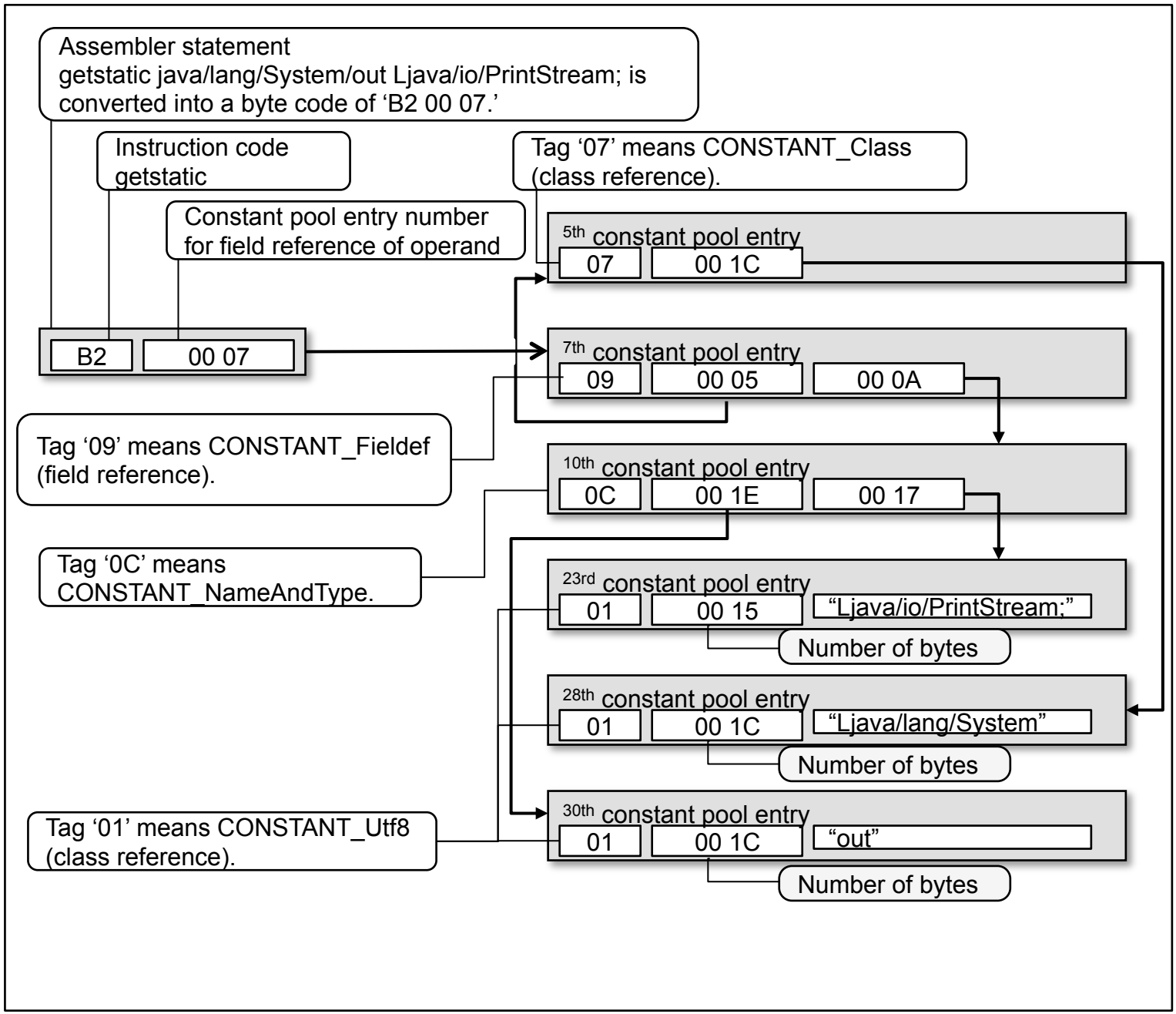
States of operand stack before and after execution of iadd



Executing iload 3


Local variables
Operand stack
0
1
600
2
123
3
iload 1
Local variables
Operand stack
0
1
600
2
123
3
600
iload 2
Local variables
Operand stack
0
1
600
2
123
3
123
600
iadd
Local variables
Operand stack
0
1
600
2
123
3
723


Class file
Magic number
CA FE BA BE (4 bytes)
Minor version (2 bytes)
Major version (2 bytes)
Constant pool
Number of constant pool entries (2 bytes)
Constant pool entry 1
Constant pool entry n
Access flag (2 bytes)
This class (2bytes)
Super class (2 bytes)
Interface
Number of interfaces (2 bytes)
Interface 1 (2 bytes)
Interface n (2 bytes)
Field
Number of fields (2 bytes)
Field 1 (2 bytes)
Field n (2 bytes)
Method
Number of methods (2 bytes)
Method 1
Method n
Attribute list
constant pool entry
Tag (1 byte)
Entry data (The data structure differs depending on the type of entry.)
Field1
Access flag (2 bytes)
Field name CP entry number (2 bytes)
Descriptor CP entry number (2 bytes)
Attribute list
Field1
Access flag (2 bytes)
Method name CP entry number (2 bytes)
Descriptor CP entry number (2 bytes)
Attribute list
Attribute list
Number of attributes (2 bytes)
Attribute 1
Attribute n
Attribute1
Attribute name CP entry number* (2bytes)
Data byte length (4bytes)
Attribute data (The data structure differs depending on the type of attribute.)
*CP entry number = constant pool entry number



Constant pool referenced by getstatic

Instruction Set

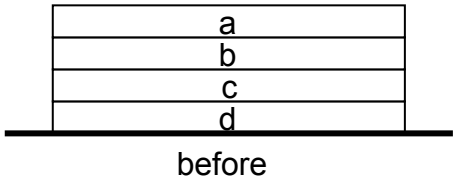
Load constant

bipush	Push byte
sipush	Push short
ldc	Push item from runtime constant pool (single-byte index)
ldc_w	Push item from runtime constant pool (wide index)
ldc2_w	Push long or double from runtime constant pool (wide index)

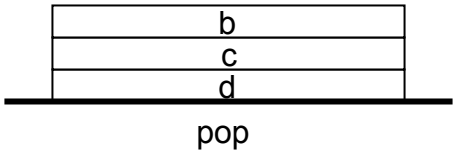
aconst_null	Push null
iconst_<i>	Push int constant (<i> : m1,0,1,2,3,4,5)
lconst_<l>	Push long constant (<l> : 0,1)
fconst_<f>	Push float constant (<f> : 0,1,2)
dconst_<d>	Push double constant (<d> : 0,1)

Pop

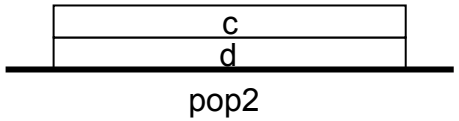
pop	Pop the top op-stack value
pop2	Pop the top two op-stack values
dup	Duplicate the top op-stack value
dup2	Duplicate the top two op-stack values
dup_x1	Duplicate the top op-stack value and insert two values down
dup_x2	Duplicate the top op-stack value and insert three values down
dup2_x1	Duplicate the top two op-stack values and insert three values down
dup2_x2	Duplicate the top two op-stack values and insert four values down
swap	Swap the top two op-stack values



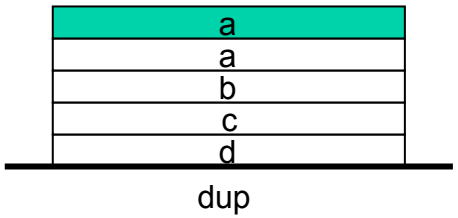
before



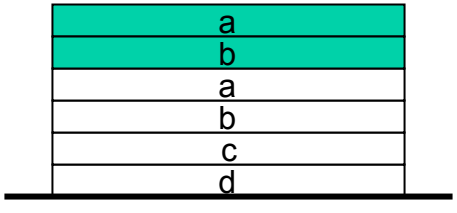
pop



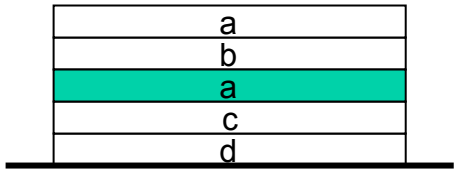
pop2



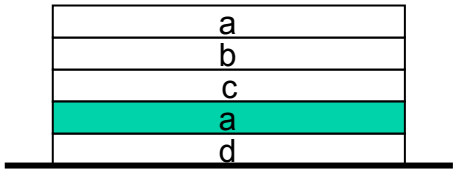
dup



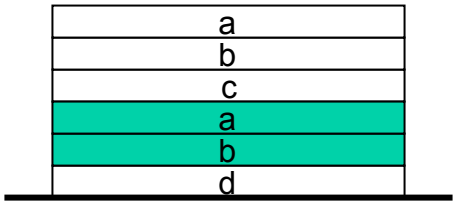
dup2



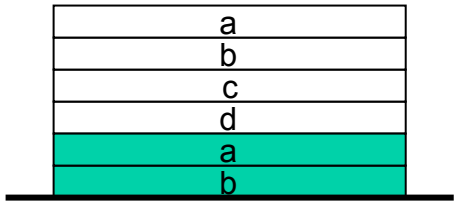
dup_x1



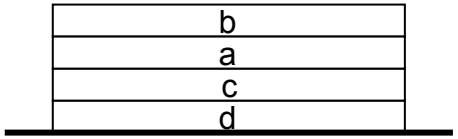
dup_x2



dup2_x1



dup2_x2

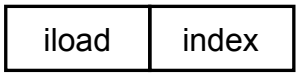


swap

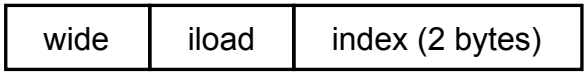
Load from local variable

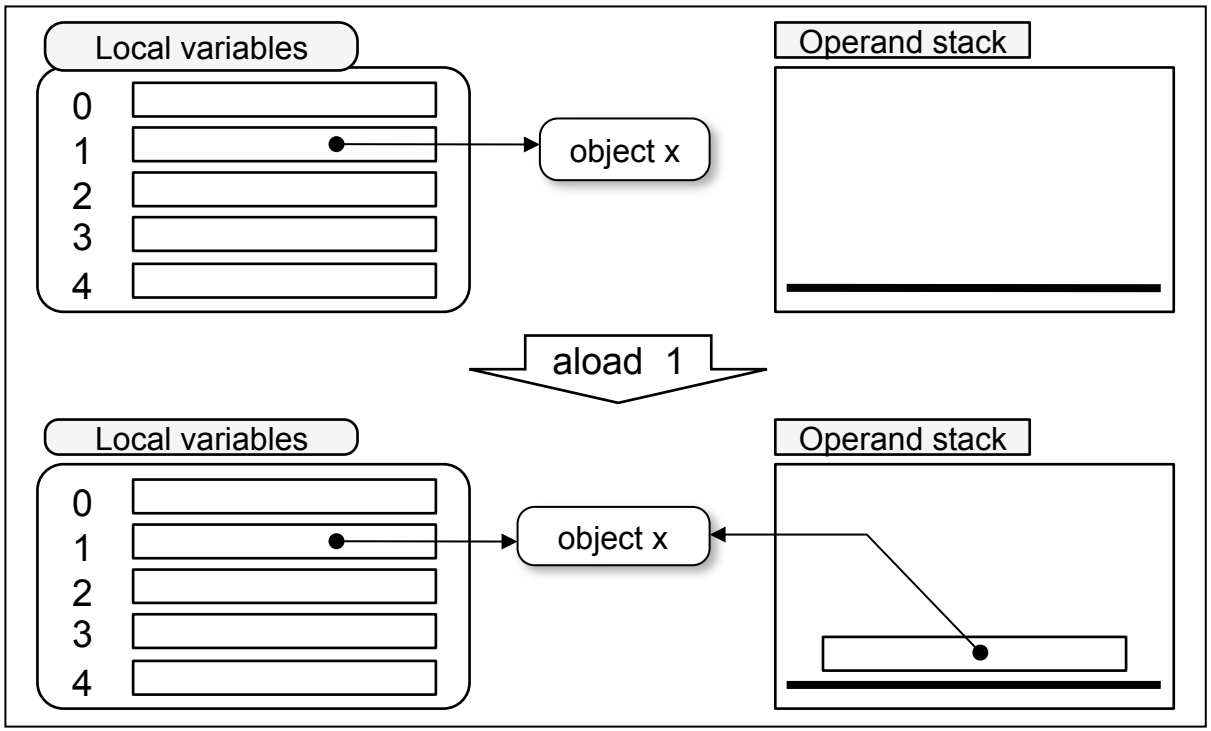
iload	Load int from local variable
iload_<n>	Load int from local variable (<n> = 0,1,2,3)
lload	Load long from local variable
lload_<n>	Load long from local variable (<n> = 0,1,2,3)
fload	Load float from local variable
fload_<n>	Load float from local variable (<n> = 0,1,2,3)
dload	Load double from local variable
dload_<n>	Load double from local variable (<n> = 0,1,2,3)
aload	Load reference from local variable
aload_<n>	Load reference from local variable (<n> = 0,1,2,3)

Variables 0～255



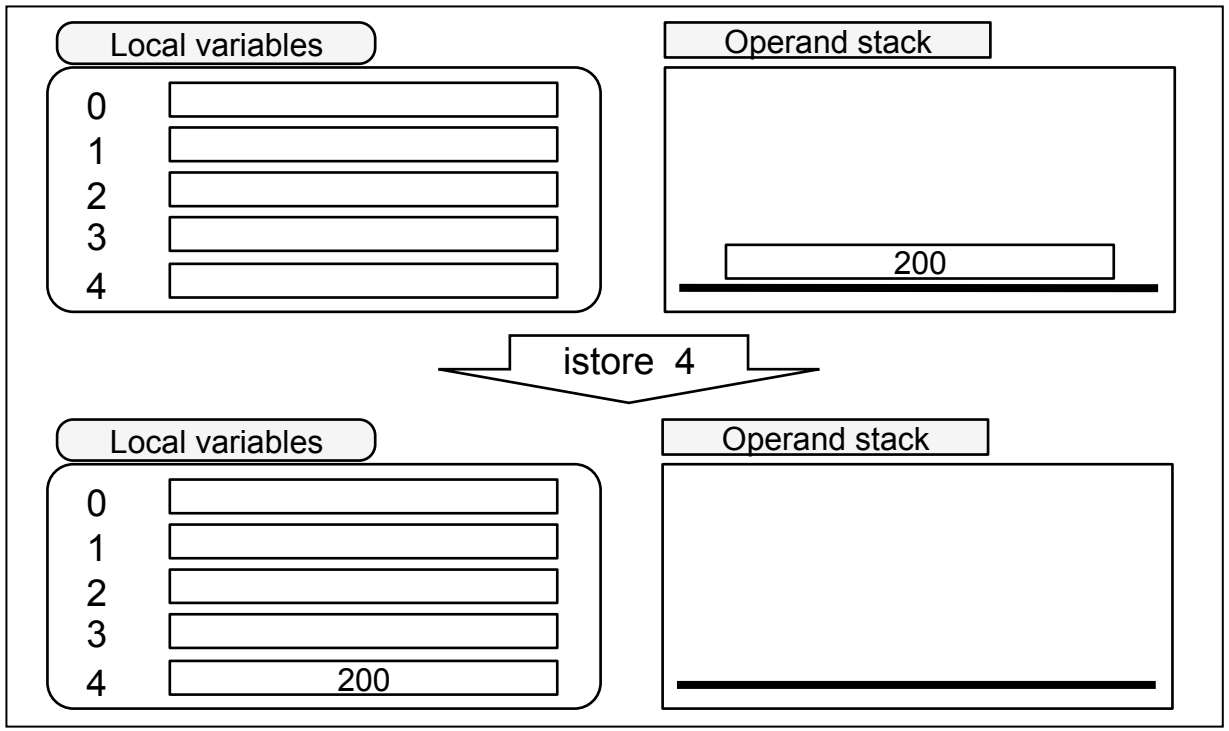
Variables 256～





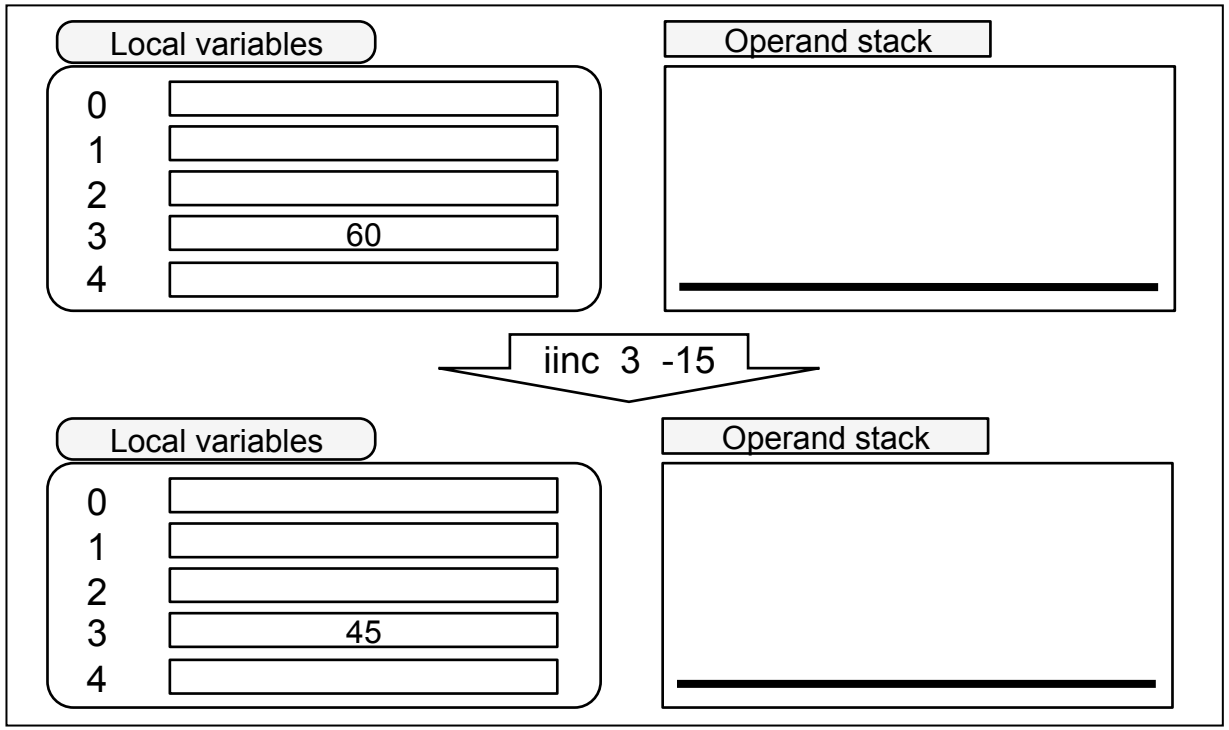
Store into local variable

istore	Store int into local variable
istore_<n>	Store int into local variable (<n> = 0,1,2,3)
lstore	Store long into local variable
lstore_<n>	Store long into local variable (<n> = 0,1,2,3)
fstore	Store float into local variable
fstore_<n>	Store float into local variable (<n> = 0,1,2,3)
dstore	Store double into local variable
dstore_<n>	Store double into local variable (<n> = 0,1,2,3)
astore	Store reference into local variable
astore_<n>	Store reference into local variable (<n> = 0,1,2,3)



Increment

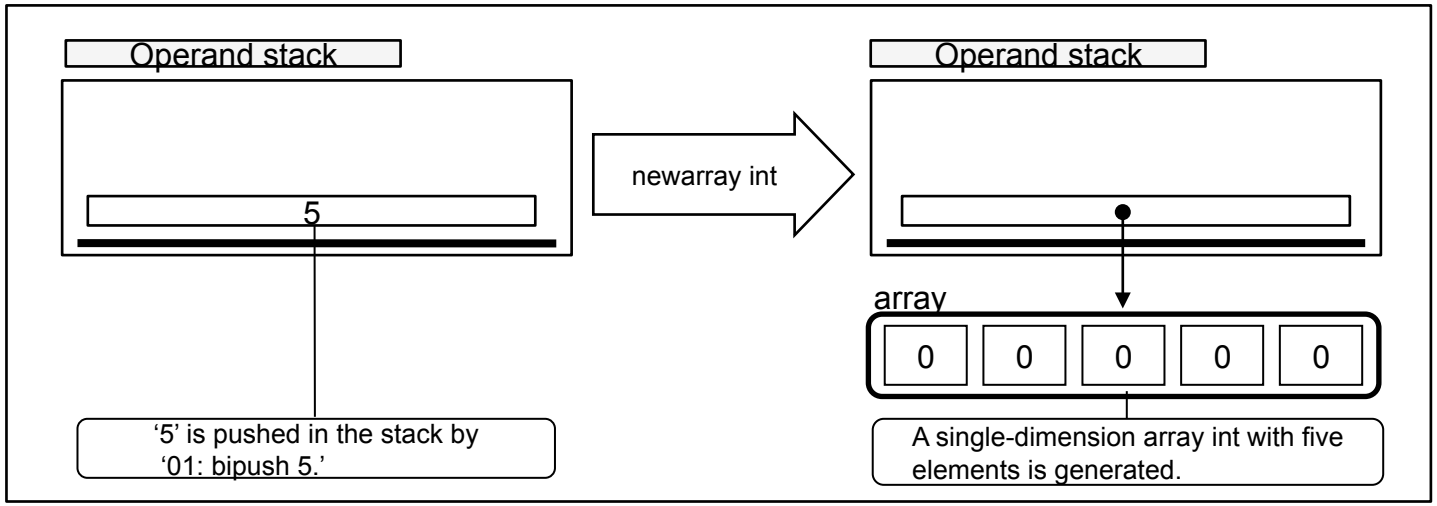
iinc Increment local variable by constant

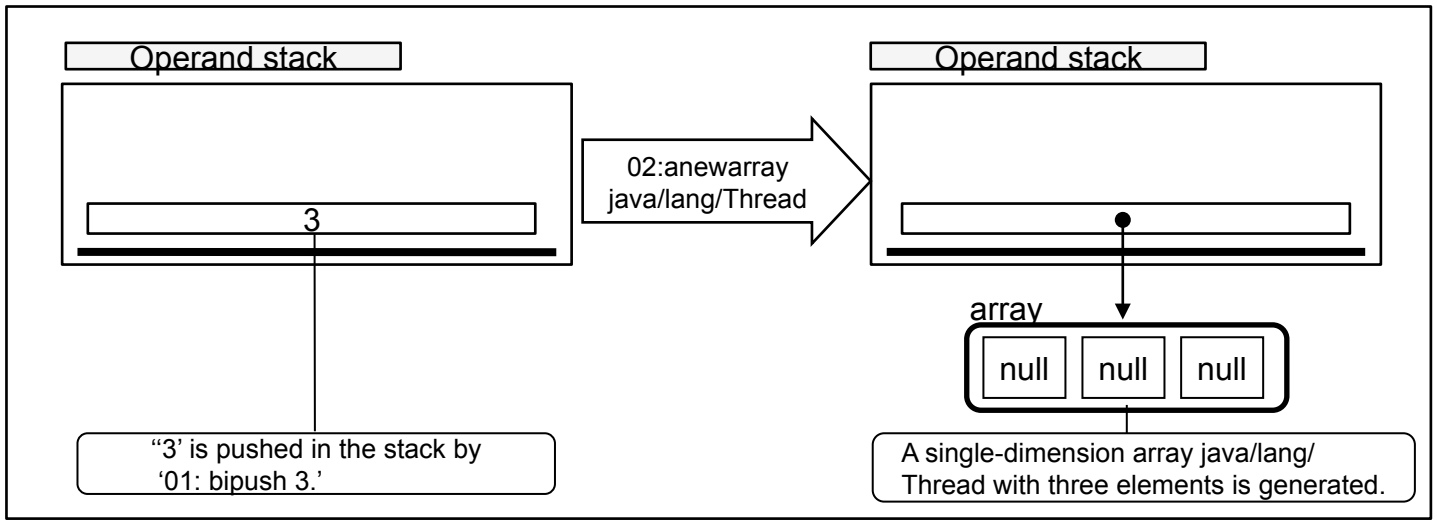


Create array

newarray Create new array of basic type
anewarray Create new array of reference
multianewarray Create new multidimensional array

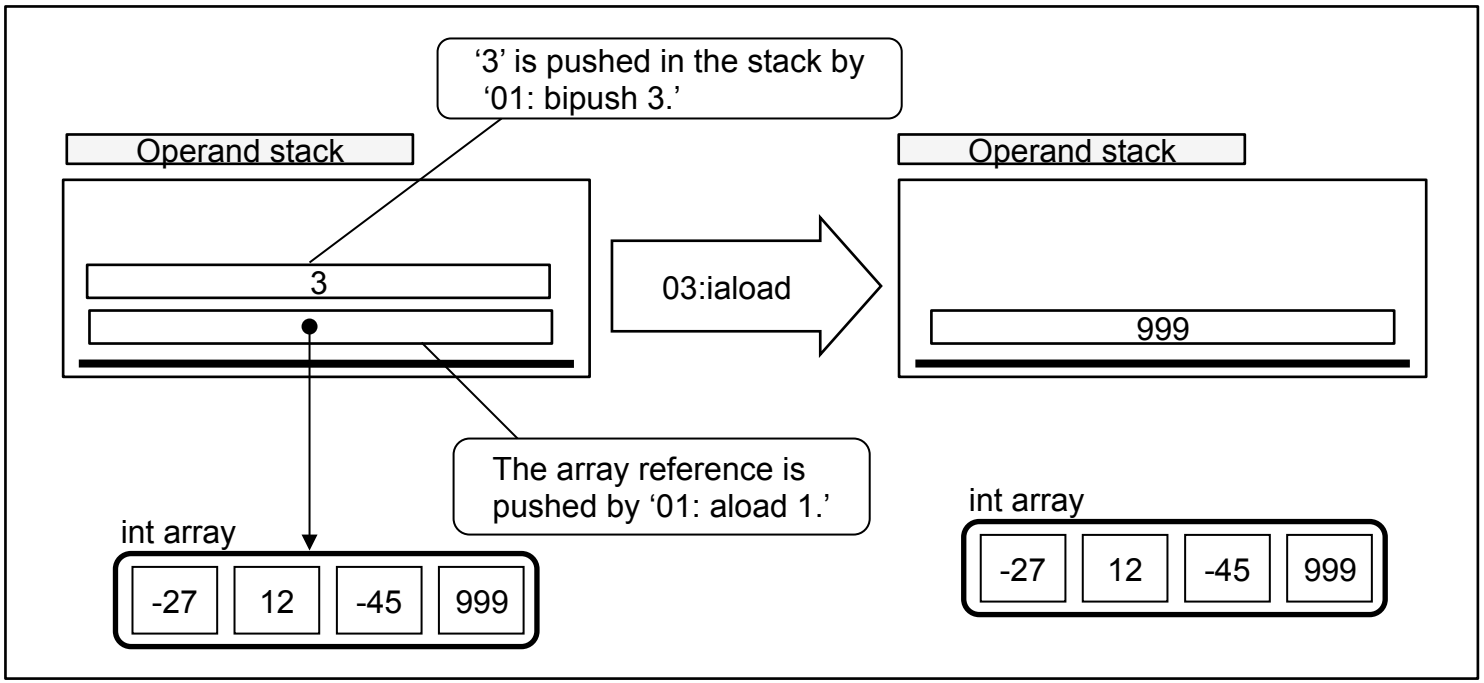
arraylength Get length of array





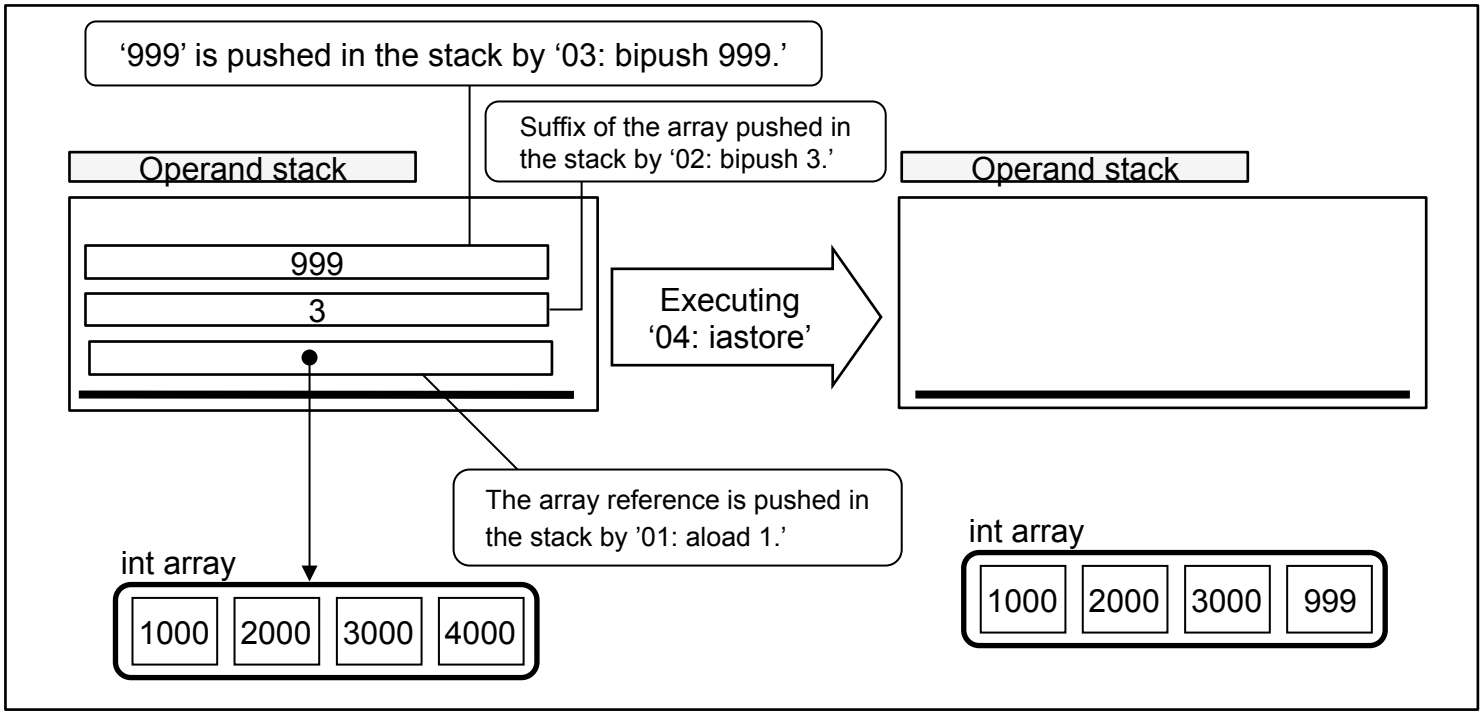
Load from Array

baload Load byte or boolean from array
caload Load char from array
saload Load short from array
iaload Load int from array
laload Load long from array
faload Load float from array
daload Load double from array
aaload Load reference from array



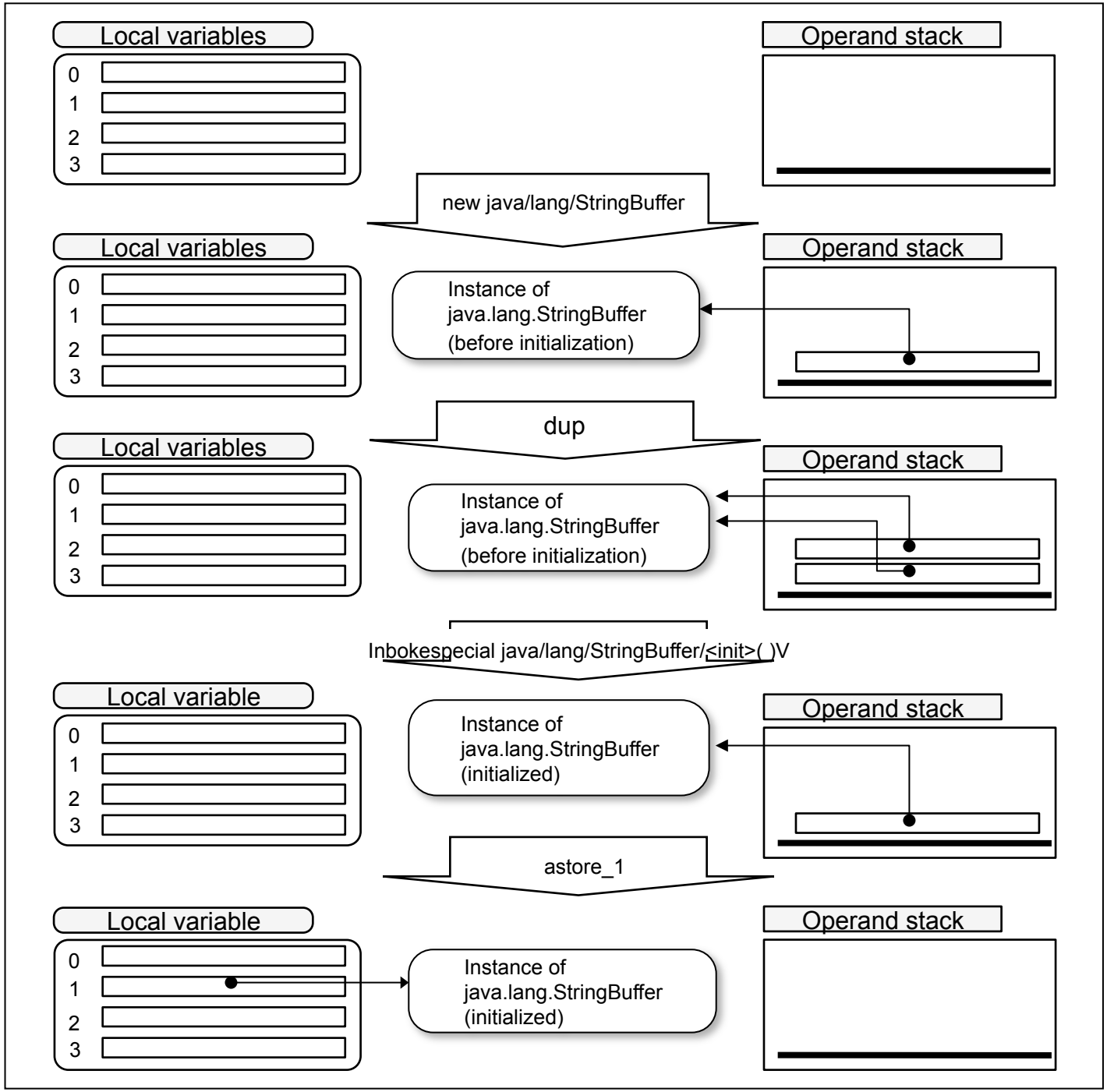
Store into array

bastore	Store into byte or boolean array
castore	Store into char array
sastore	Store into short array
iastore	Store into int array
lastore	Store into long array
fastore	Store into float array
dastore	Store into double array
aastore	Store into reference array



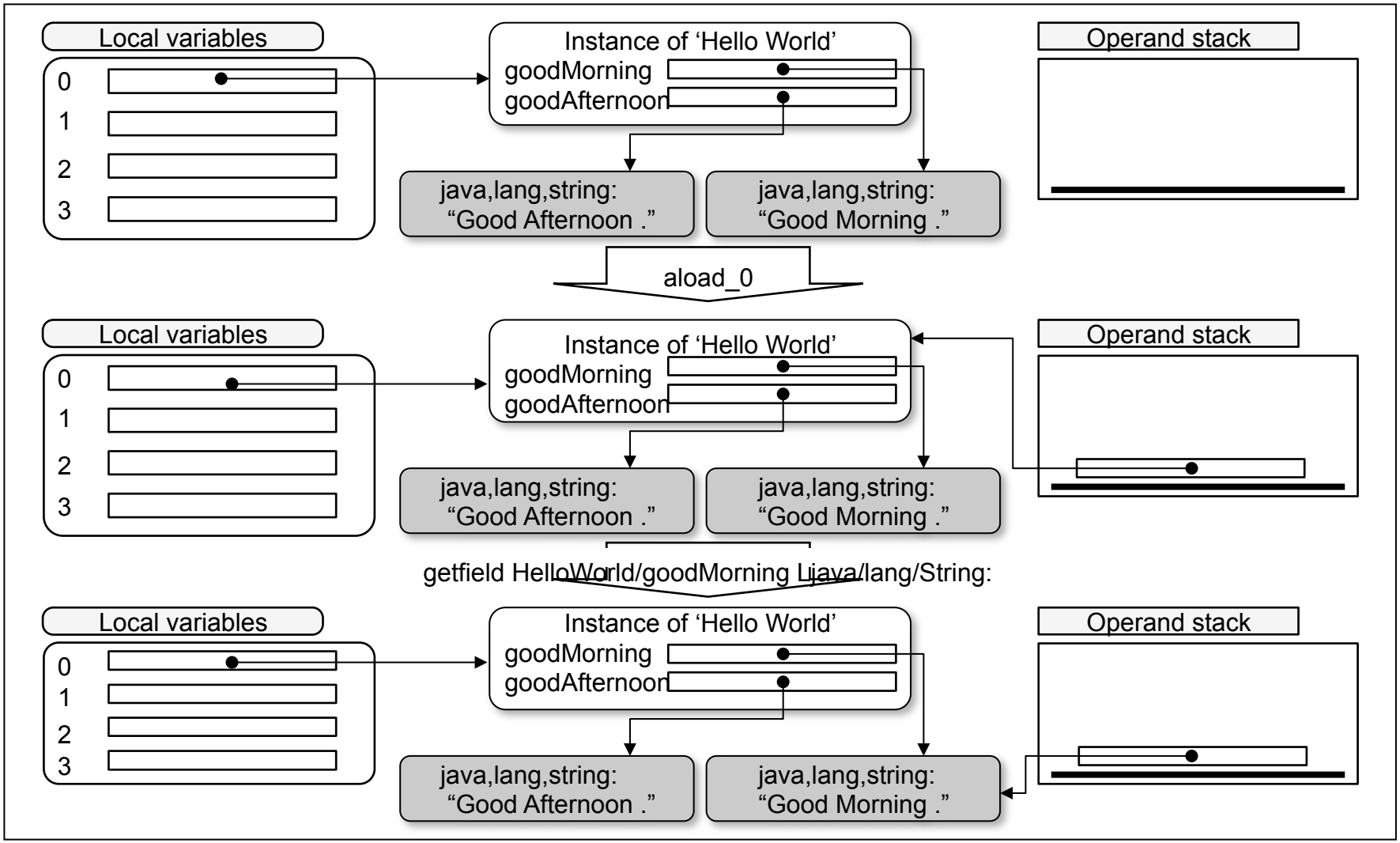
Create new object

new Create new object



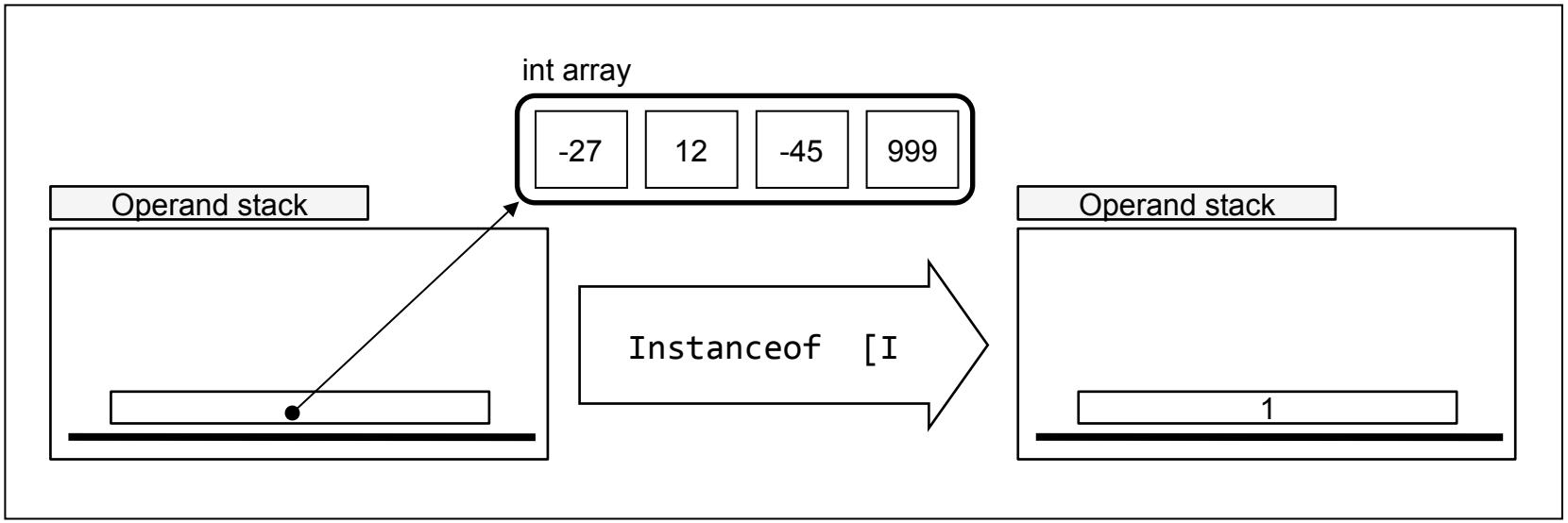
Access field

putfield Set field in object
putstatic Set static field in class
getfield Fetch field from object
getstatic Get static field from class



Type check

checkcast	Check whether object is of given type
instanceof	Determine if object is of given type

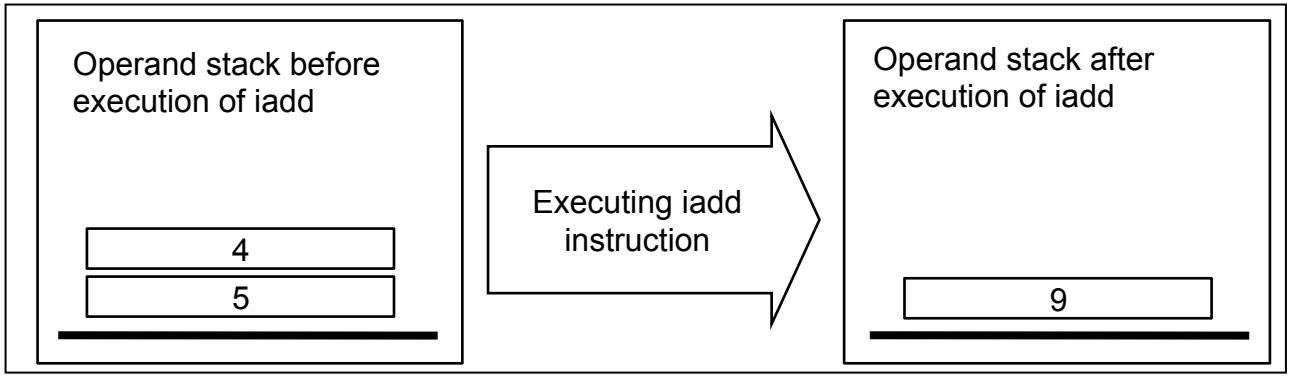


Arithmetic & logical operations

iadd	Add int
ladd	Add long
fadd	Add float
dadd	Add double
isub	Subtract int
lsub	Subtract long
fsub	Subtract float
dsub	Subtract double
imul	Multiply int
lmul	Multiply long
fmul	Multiply float
dmul	Multiply double
idiv	Divide int
ldiv	Divide long
fdiv	Divide float
ddiv	Divide double
irem	Remainder int
lrem	Remainder long
frem	Remainder float
Drem	Remainder double

ineg	Negate int
lneg	Negate long
fneg	Negate float
dneg	Negate double

ishl	Shift left int
ishr	Arithmetic shift right int
iushr	Logical shift right int
iand	Boolean AND int
ior	Boolean OR int
ixor	Boolean XOR int
lshl	Shift left long
lshr	Arithmetic shift right long
lushr	Logical shift right long
land	Boolean AND long
lor	Boolean OR long
lxor	Boolean XOR long



Type conversion

i2d	Convert int to double
i2f	Convert int to float
i2l	Convert int to long
l2d	Convert long to double
l2f	Convert long to float
f2d	Convert float to double

f2i	Convert float to int
f2l	Convert float to long
d2f	Convert double to float
d2i	Convert double to int
d2l	Convert double to long

i2b	Convert int to byte
i2c	Convert int to char
i2s	Convert int to short
l2i	Convert long to int

Conditional jump & comparison

if<cond>	Branch if int comparison with zero succeeds (<cond> : eq, ne, lt, ge, gt, le)
if_icmp<cond>	Branch if int comparison succeeds (<cond> : eq, ne, lt, ge, gt, le)
if_acmp<cond>	Branch if reference comparison succeeds (<cond> : eq, ne)
ifnonnull	Branch if reference not null
ifnull	Branch if reference is null

lcmp	Compare long, values 1, 0, -1
fcmp<op>	Compare float (<op> : g, l)
dcmp<op>	Compare double (<op> : g, l)

```
class Flow1 {
    int test (int i) {
        if (i < 10)
            return i;
        else
            return 10;
    }
}
```

```
Method int test(int)
        0 iload_1
        1 bipush 10
        3 if_icmpge 8
        6 iload_1
        7 ireturn
        8 bipush 10
       10 ireturn
```

0	1	2	3	4	5	6	7	8	9	10
iload	bipush	10	if	0	8	iload	ireturn	bipush	10	ireturn

Jump

goto	Branch always
goto_w	Branch always (wide index)
jsr	Jump subroutine
jsr_w	Jump subroutine (wide index)
ret	Return from subroutine

```
int c=10;
while (c > 0) c--;
```

```
        bipush 10
        istore_1
Loop:
        iload_1
        ifle Done
        iinc 1 -1
        goto Loop
Done:
```

Switch-case

lookupswitch Access jump table by key match and jump

tableswitch Access jump table by index and jump

```
switch (i) {
case 1:
        return 1;
case 10:
        return 2;
case 100:
        return 3;
default:
        return 0;
}
```

```
        iload_1;
        lookupswitch 3
          1    : R1
          10   : R2
          100 : R3
          default: R4
R1:   iconst_1
        ireturn
R2:   iconst_2
        ireturn
R3:   iconst_3
        ireturn
R4:   iconst_0
        ireturn
```

```
switch (i) {
case 1:
        return 1;
case 2:
        return 2;
case 3:
        return 3;
default:
        return 0;
}
```

```
        iload_1;
        tableswitch 1 to 3
              R1
              R2
              R3
          default: R4
R1:   iconst_1
        ireturn
R2:   iconst_2
        ireturn
R3:   iconst_3
        ireturn
R4:   iconst_0
        ireturn
```

Method invocation

invokeinterface	Invoke interface method
invokespecial	Invoke instance method; special handling for superclass, private, and instance initialization method invocations
invokestatic	Invoke a class (static) method
invokevirtual	Invoke instance method; dispatch based on class

Return from method

ireturn	Return int from method
lreturn	Return long from method
freturn	Return float from method
dreturn	Return double from method
areturn	Return reference from method
return	Return void from method

```
class Flow1 {
    int test (int i) {
        if (i < 10)
            return i;
        else
            return 10;
    }
}
```

```
Method int test(int)
        0 iload_1
        1 bipush 10
        3 if_icmpge 8
        6 iload_1
        7 ireturn
        8 bipush 10
       10 ireturn
```

Miscellaneous

nop	Do nothing
athrow	Throw exception or error
monitorenter	Enter monitor for object
monitorexit	Exit monitor for object
wide	Declares next instruction uses double-byte local variable index