let's move
the java world

Mastering Java Bytecode with ASM

Learn some bytecode to yourself!

whoami

Anton Arhipov

Java Dev / Product Lead

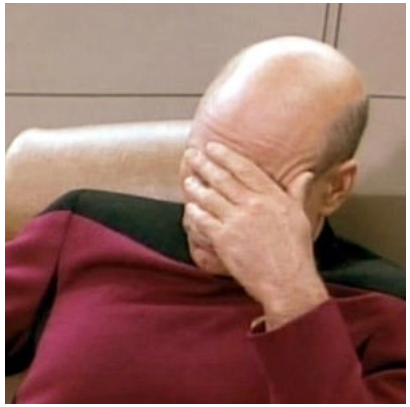
ZeroTurnaround, **JRebel**

Messing with bytecode since 2010

anton@zeroturnaround.com

@antonarhipov @javarebel

whoami



Anton Arhipov

Java Dev / Product Lead

ZeroTurnaround, **JRebel**

Messing with bytecode since 2010

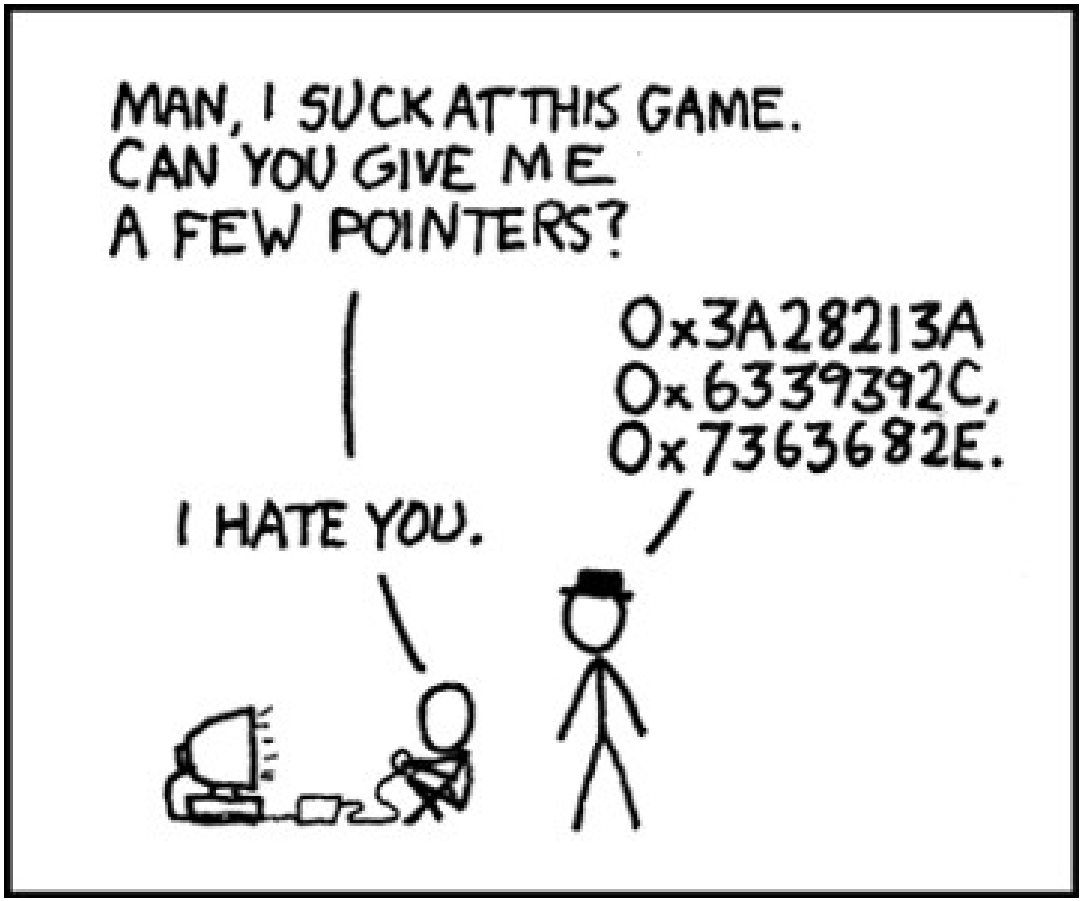
anton@zeroturnaround.com

@antonarhipov @javarebel

Why Bytecode?

- Know your platform!
- Build your own JVM language?
- Programming models (AOP, ORM)
- Awesome tools (like JRebel ^)

... just bored?



Bytecode 101

Instrumentation AP

javap

ObjectWeb ASM

Bytecode 101

Gentle introduction

Adding Two Values

$$A + B$$

Adding Two Values

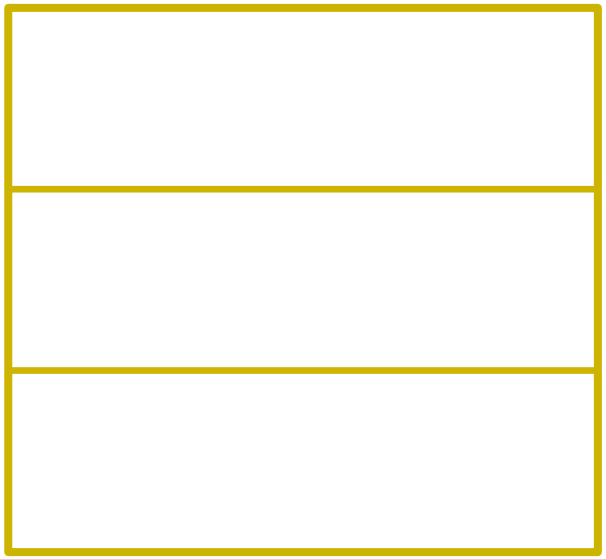
A + B

A B +

Adding Two Values

A + B

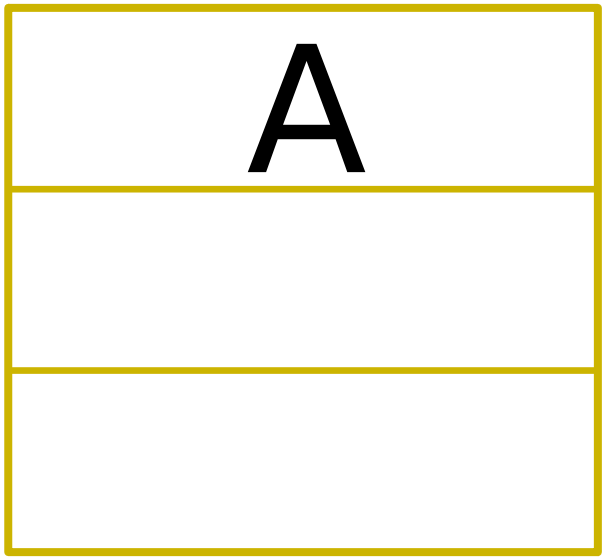
A B +



Adding Two Values

A + B **PUSH A**

A B +



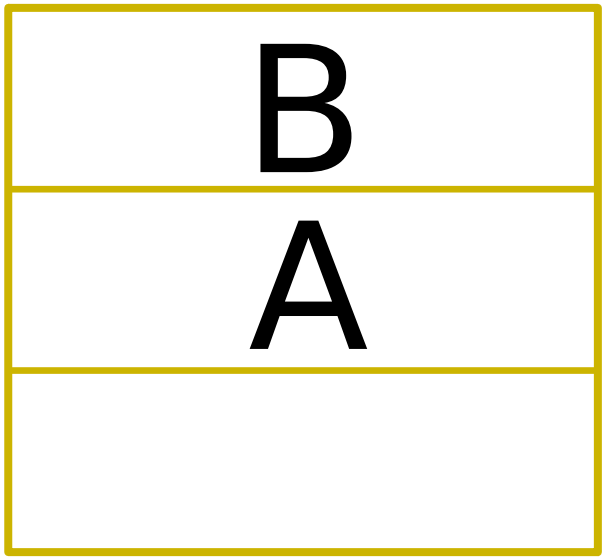
Adding Two Values

A + B

A B +

PUSH 1

PUSH 2



Adding Two Values

A + B

A B +

PUSH 1

PUSH 2

ADD

15

Adding Two Values

A + B

A B +

ICONST_1
ICONST_2
IADD

15

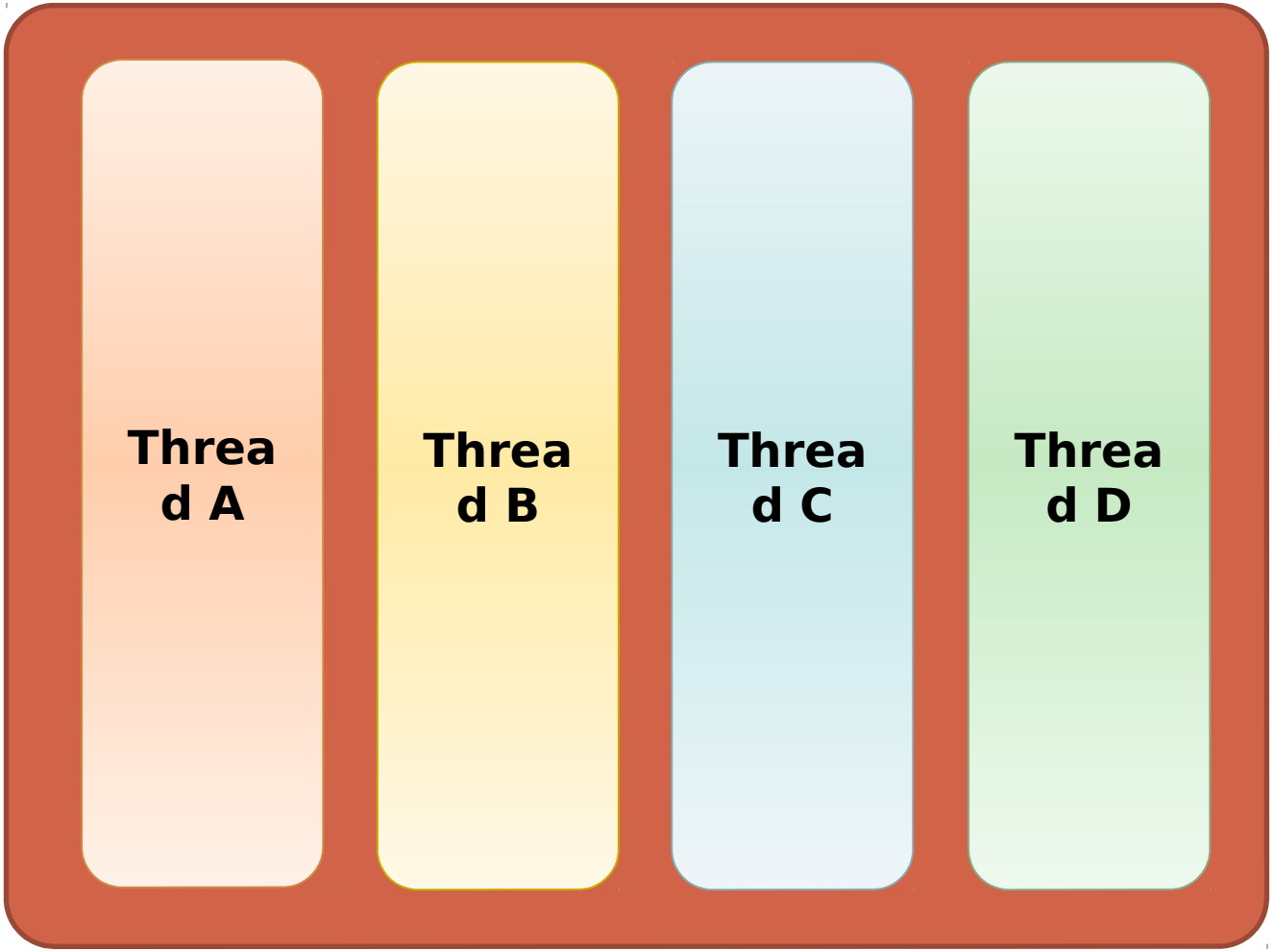
TYPE OPERATION

- <TYPE> ::= b, s, c, i, l, f, d, a
- ***constant values (ldc, iconst_1)***
- Local variables and stack interaction (**load**/**store**)
- Array operations (**aload**, **astore**)
- Math (**add**, **sub**, **mul**, **div**)
- Boolean/bitwise operations (**iand**, **ixor**)
- Comparisons & branching (**cmpl**, **ifeq, jsr, tableswitch**)
- Conversions (**l2d**, **i2l**)

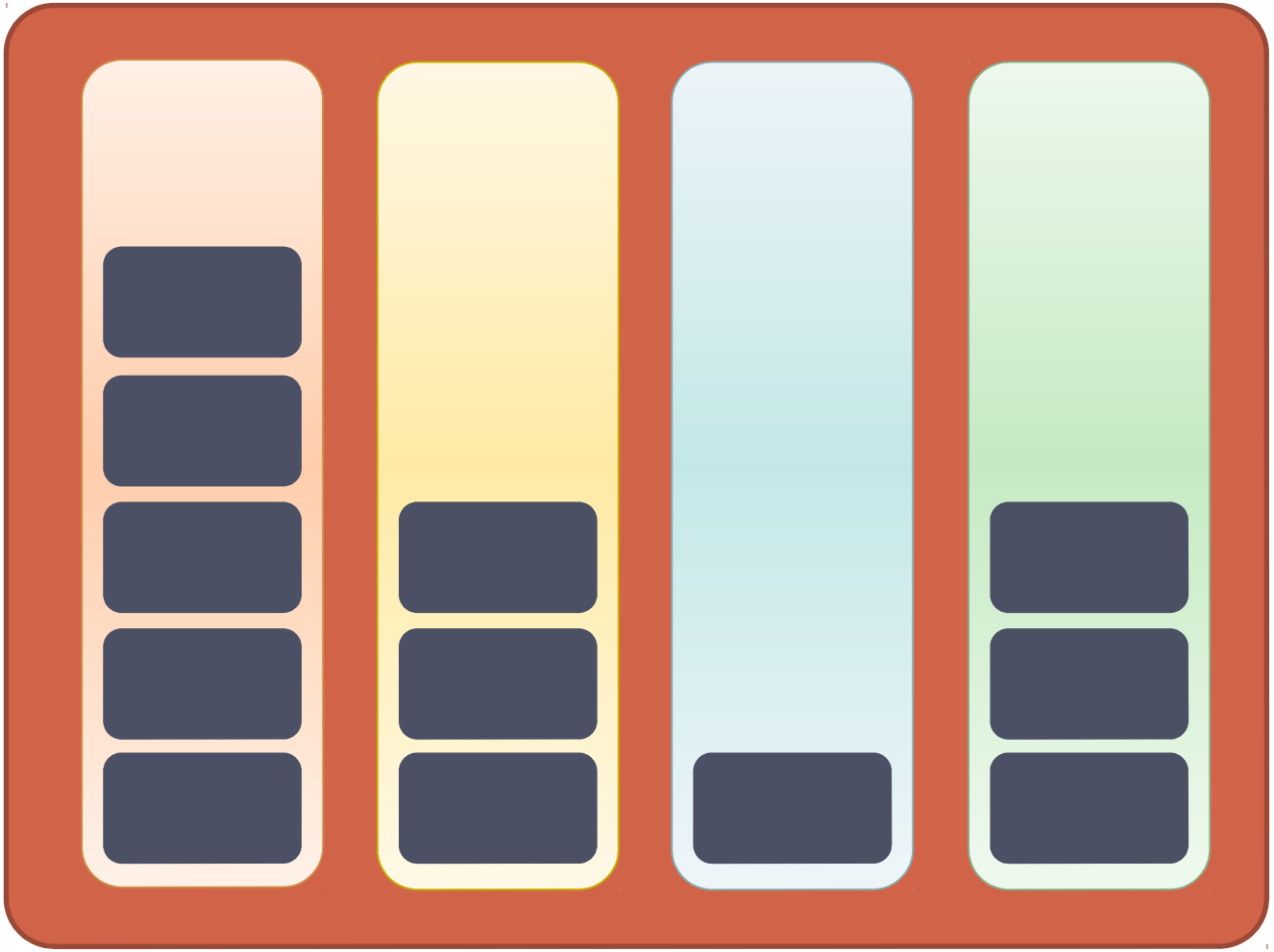
Model of Execution

Enter JVM

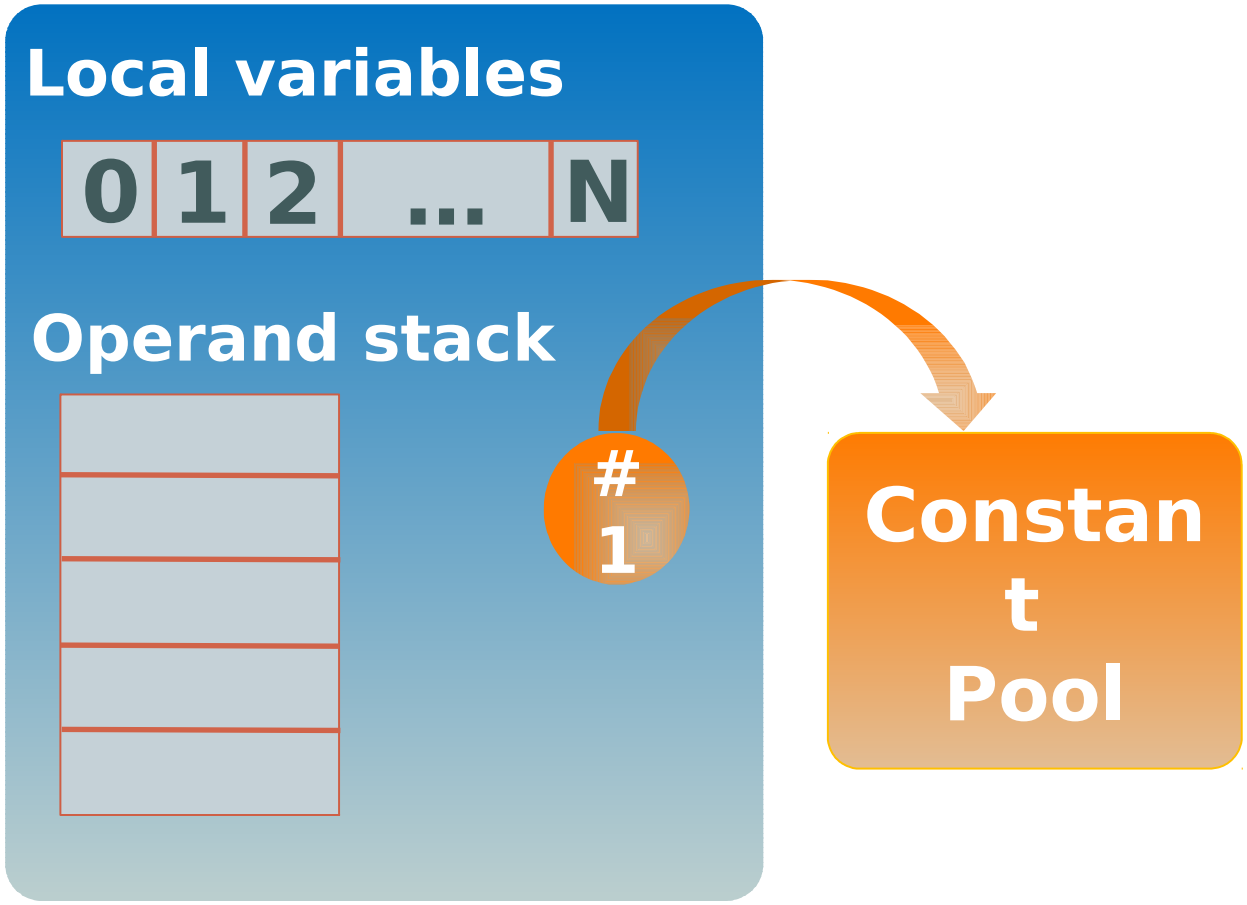
Enter Threads



Enter Frames



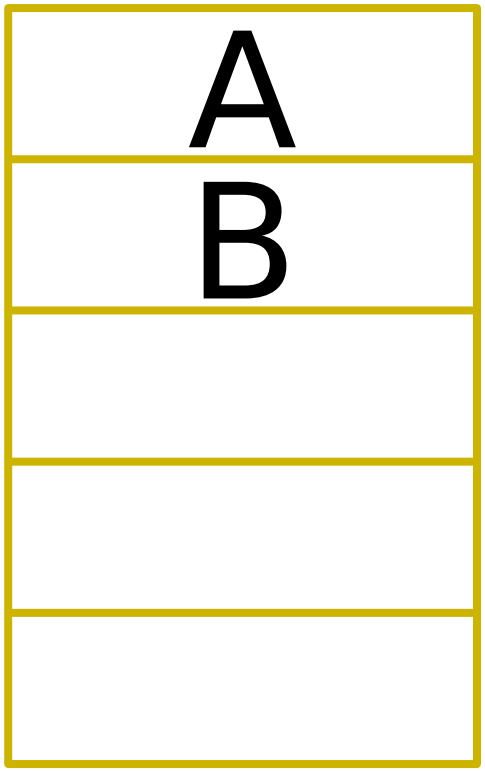
The Frame



Juggling The Stack

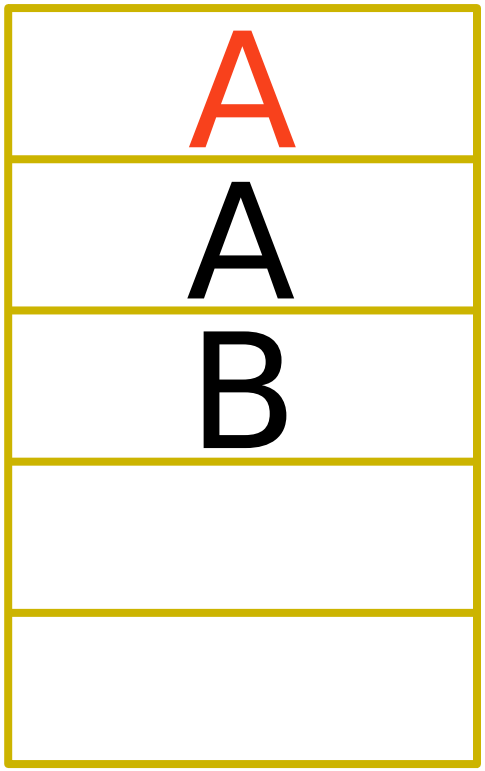
Juggling The Stack

dup
pop
swap
dup_x1
dup2_x1



Juggling The Stack

dup
pop
swap
dup_x1
dup2_x
1



Juggling The Stack

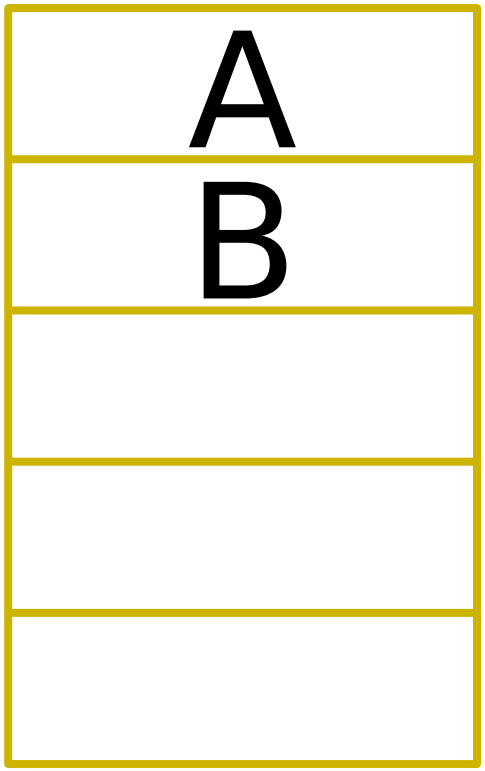
dup

pop

swap

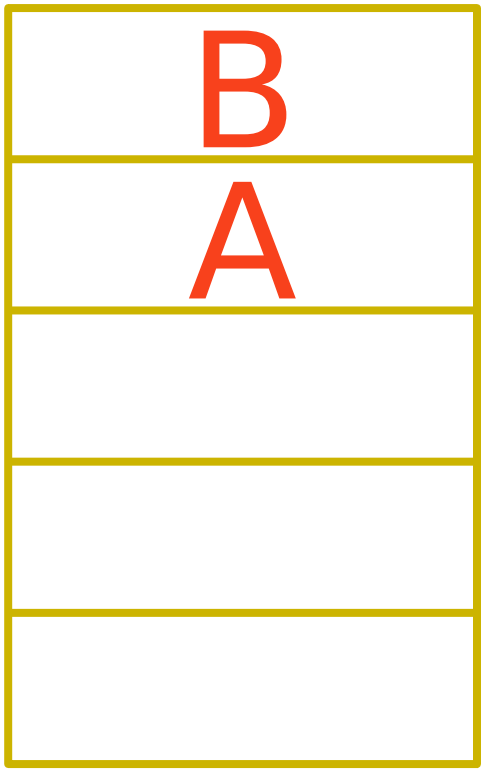
dup_x1

dup2_x1



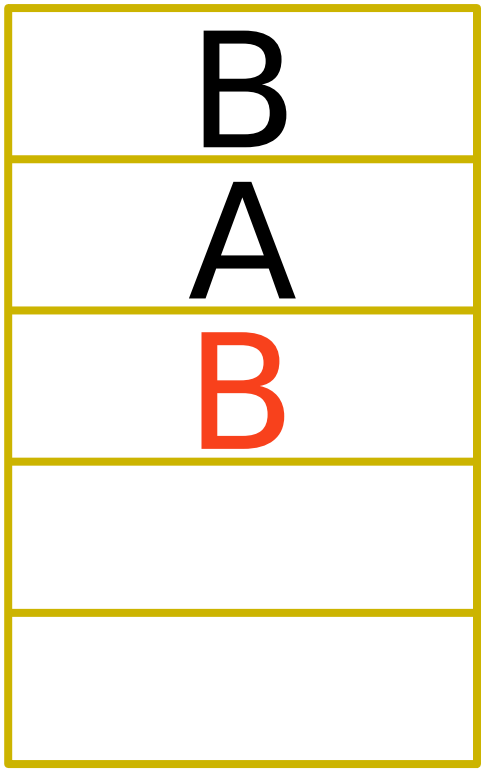
Juggling The Stack

dup
pop
swap
dup_x1
dup2_x
1



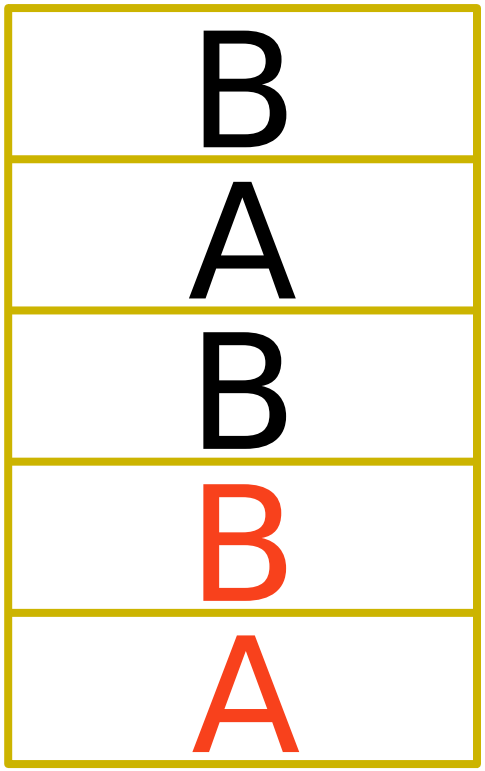
Juggling The Stack

dup
pop
swap
dup_x1
dup2_x
1



Juggling The Stack

dup
pop
swap
dup_x1
dup2_x1



Local Variables

Local Variables

var	value
0	
1	
2	
3	
4	

```
ldc
"Hello"
astore_0
iconst_1
astore_1
aload_0
```

Stack

depth	value
0	
1	
2	
3	
4	

Local Variables

var	value
0	
1	
2	
3	
4	

```
ldc
"Hello"
astore_0
iconst_1
astore_1
aload_0
```

Stack

depth	value
0	"Hello"
1	
2	
3	
4	

Local Variables

var	value
0	"Hello"
1	
2	
3	
4	

```
ldc
"Hello"
astore_0
iconst_1
astore_1
aload_0
```

Stack

depth	value
0	
1	
2	
3	
4	

Local Variables

var	value
0	"Hello"
1	
2	
3	
4	

```
ldc
"Hello"
astore_0
iconst_1
astore_1
aload_0
```

Stack

depth	value
0	1
1	
2	
3	
4	

Local Variables

var	value
0	"Hello"
1	1
2	
3	
4	

```
ldc
"Hello"
astore_0
iconst_1
astore_1
aload_0
```

Stack

depth	value
0	
1	
2	
3	
4	

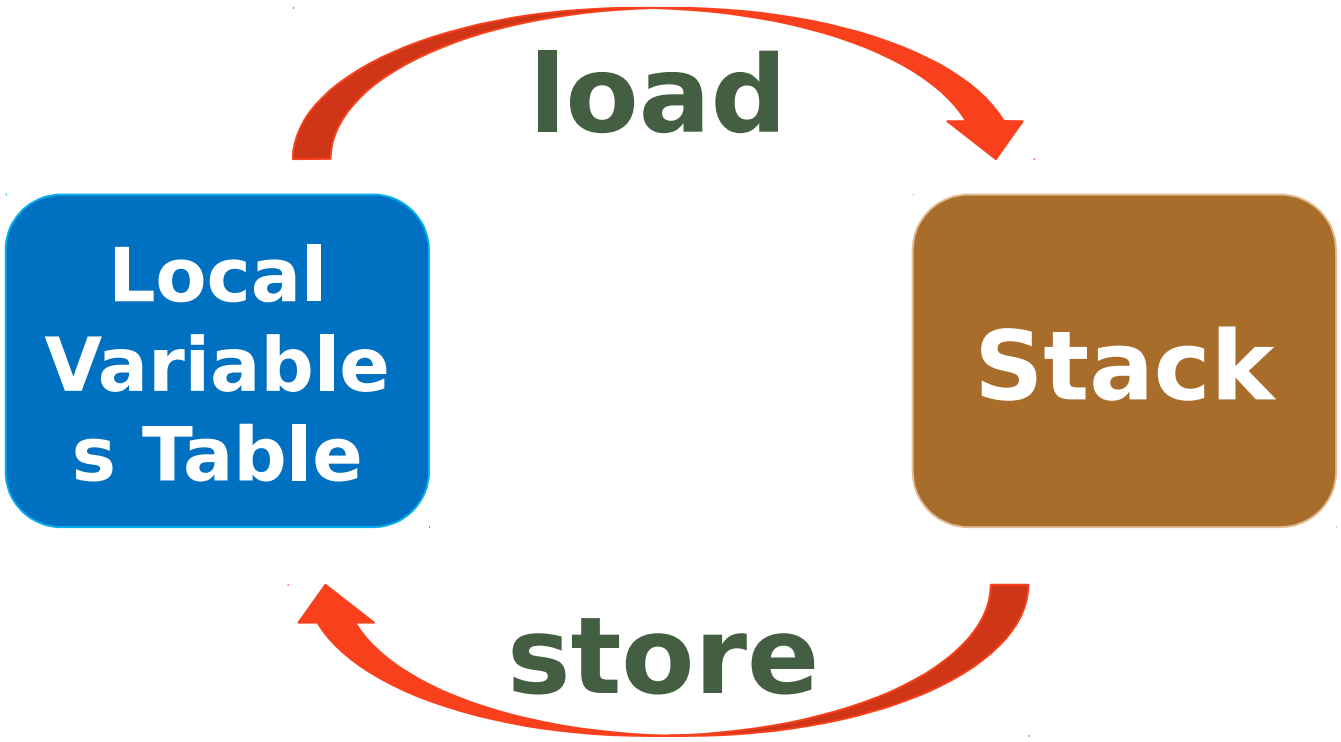
Local Variables

var	value
0	"Hello"
1	1
2	
3	
4	

```
ldc
"Hello"
astore_0
iconst_1
astore_1
aload_0
```

Stack

depth	value
0	"Hello"
1	
2	
3	
4	


load
Local Variables Table
Stack
store

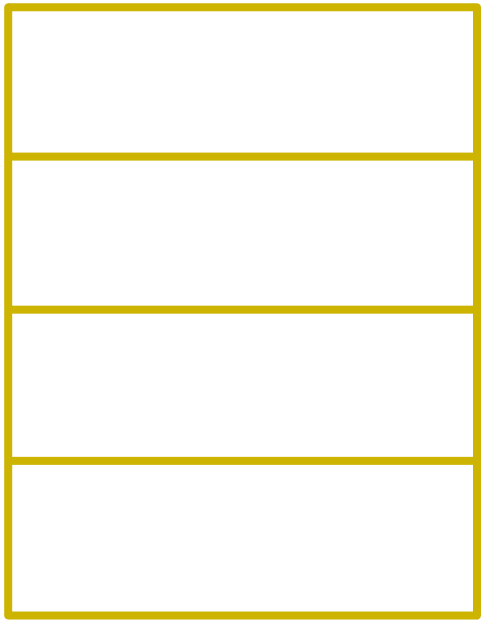
Method Invocation

Method Invocation

```
obj.method(param1,
param2);
```

Method Invocation

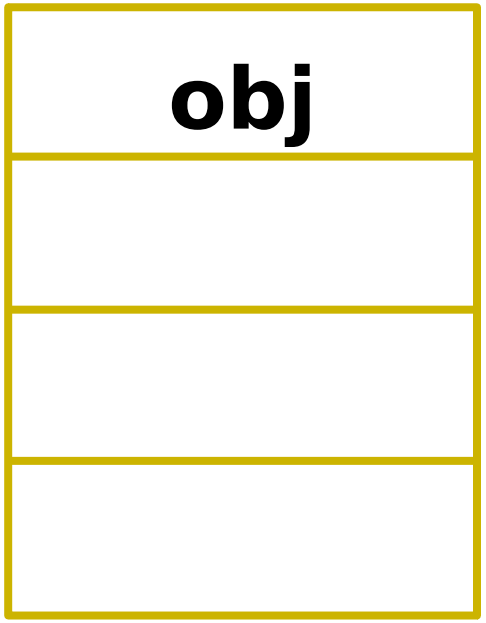
```
obj.method(param1,
param2);
        push obj
        push param1
        push param2
        invoke
        method
```



Method Invocation

```
obj.method(param1,
param2);
```

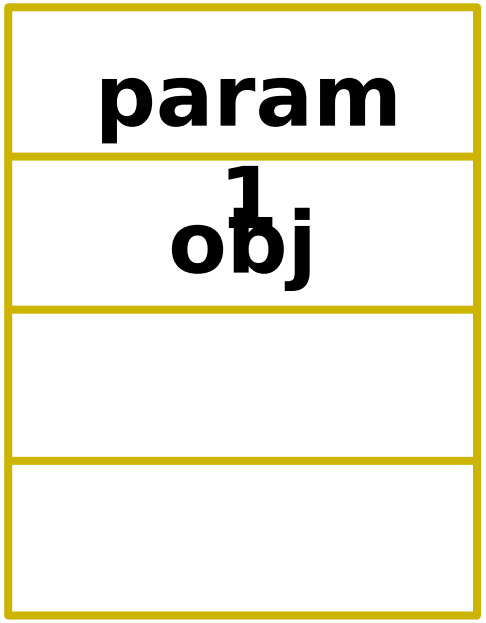
```
push obj
push param1
push param2
invoke
method
```



Method Invocation

```
obj.method(param1,
param2);
```

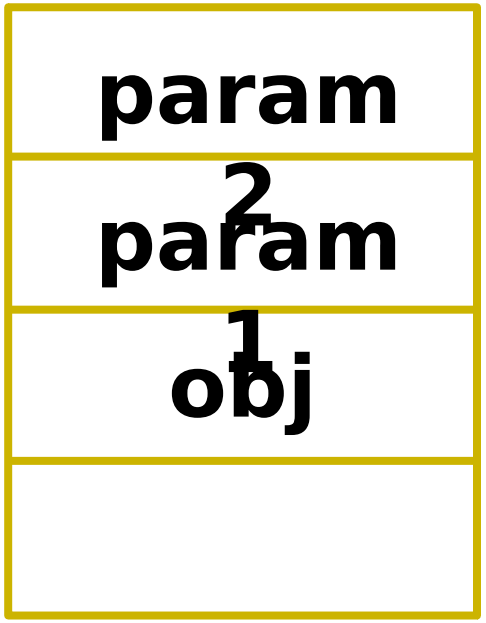
```
push obj
push param1
push param2
invoke
method
```



Method Invocation

```
obj.method(param1,
param2);
```

```
push obj
push param1
push param2
invoke
method
```



Method Invocation

```
obj.method(param1,
param2);
```

```
push obj
push param1
push param2
invoke
method
```

Operator Overloading

Operator Overloading

[int] **A + B**

[Foo] **A.plus(B)**

Operator Overloading

[int] **A + B**

push A
push B
iadd

[Foo] **A.plus(B)**

push A
push B
invokevirtual plus

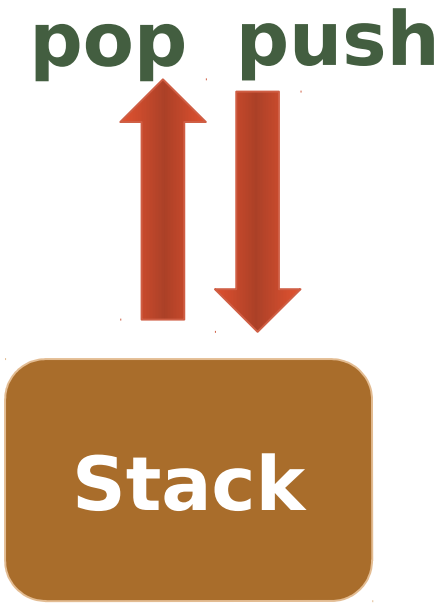
Operator Overloading

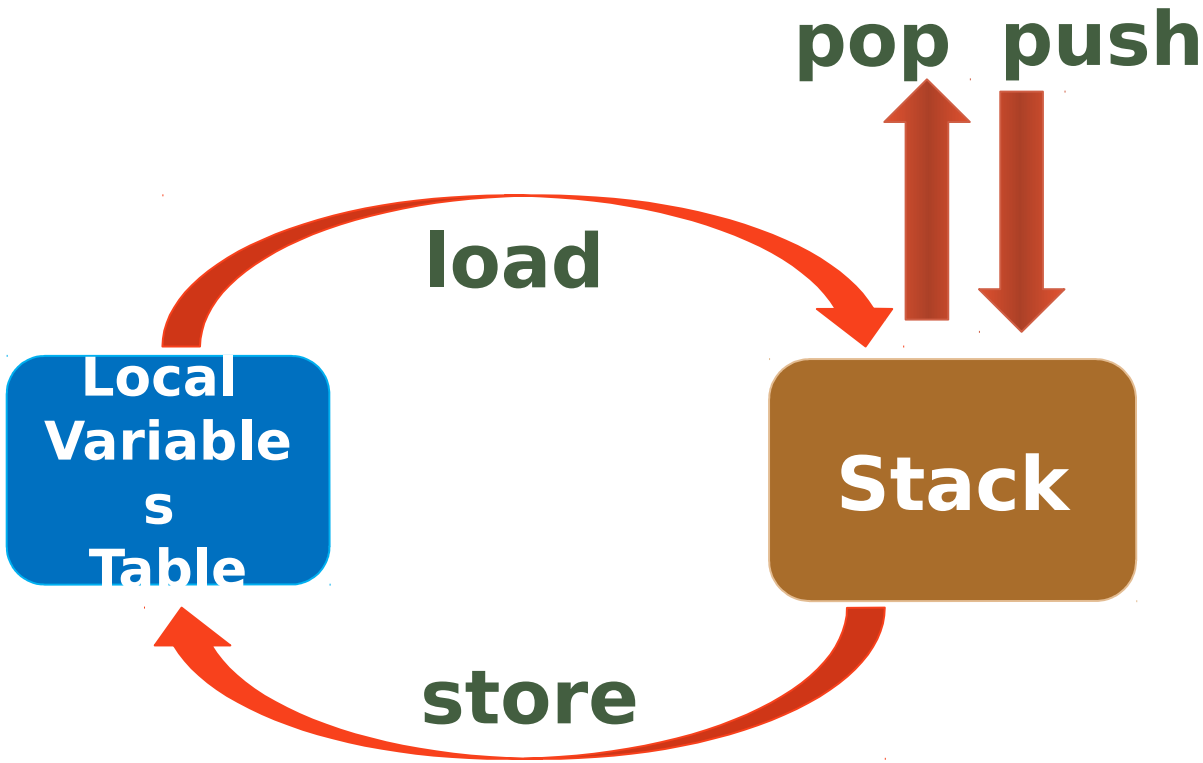
[int] **A + B**

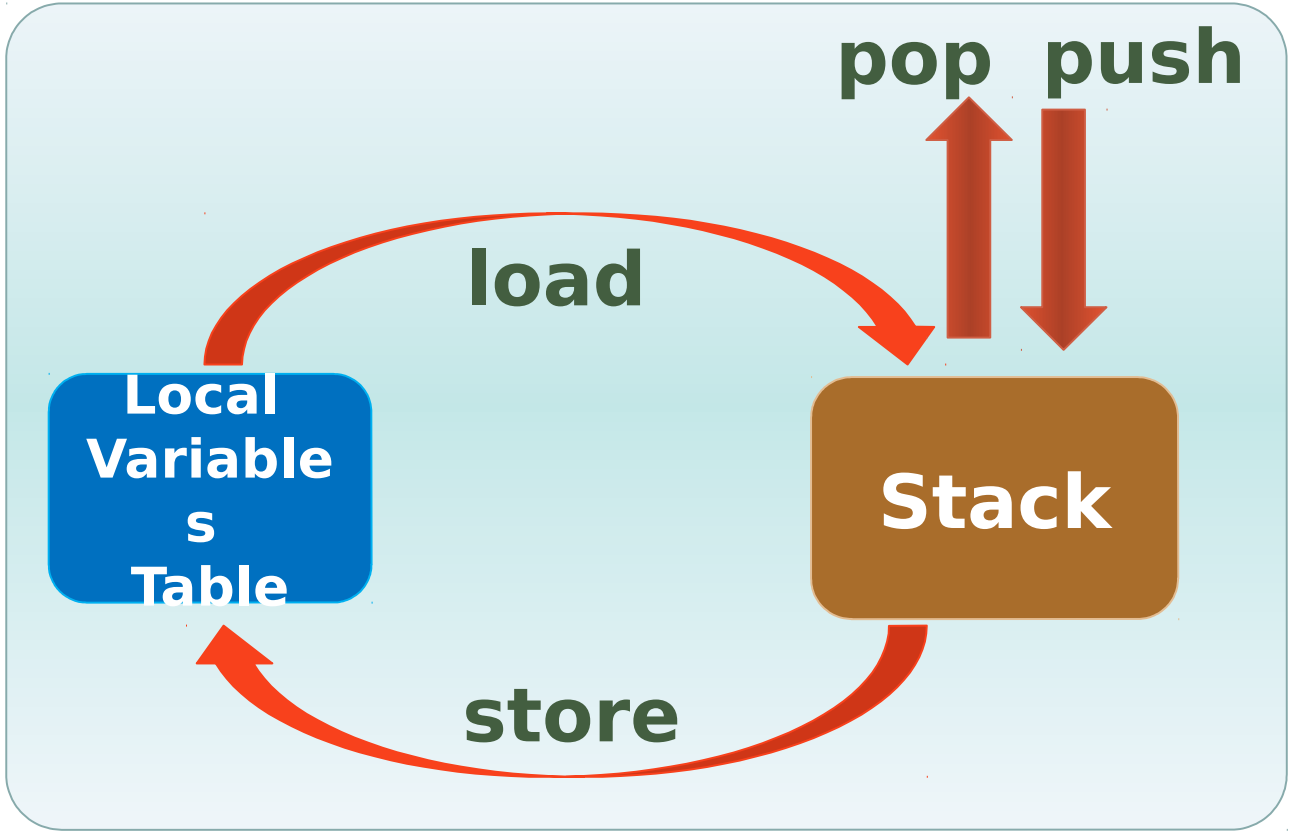
push A
push B
iadd

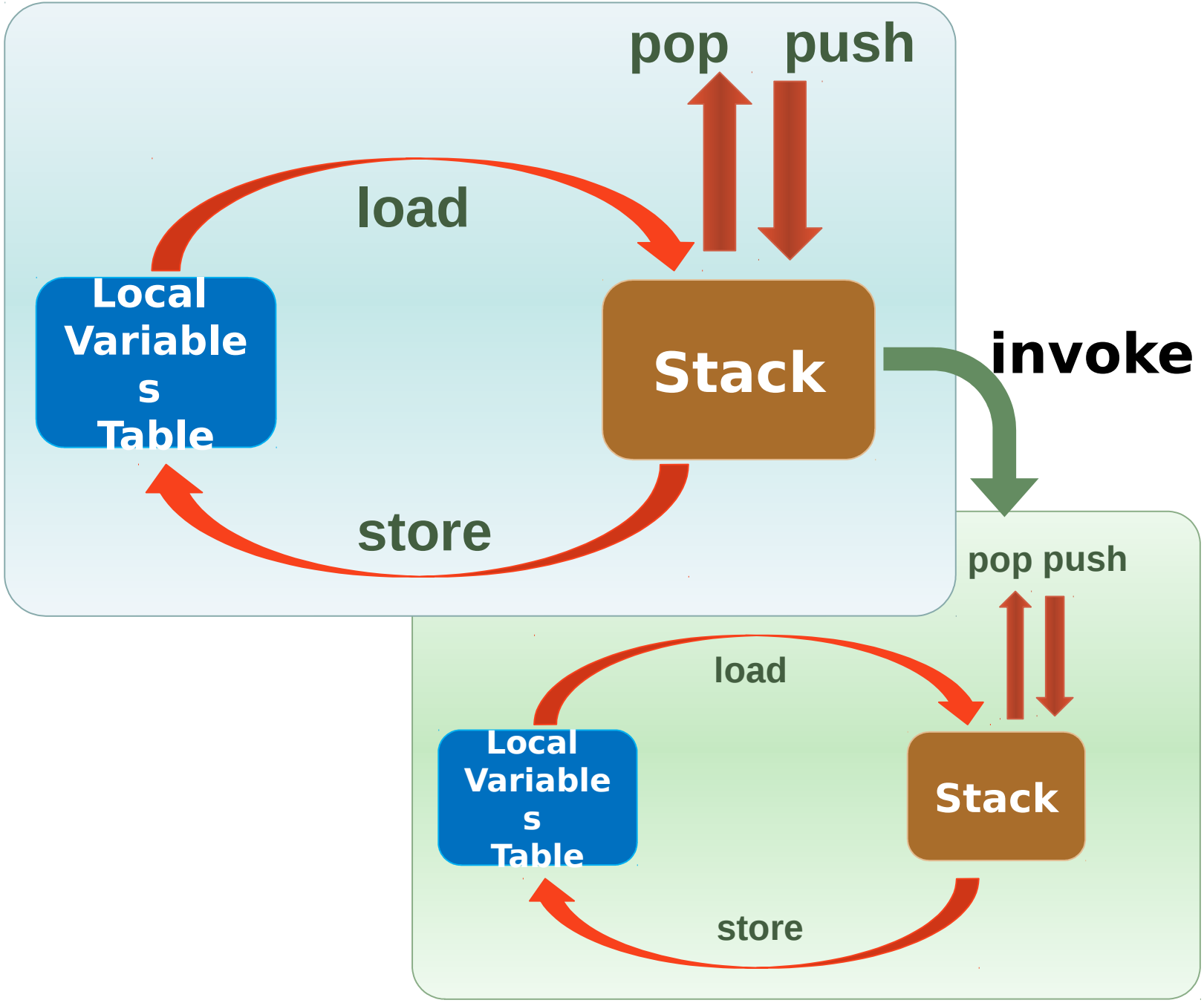
[Foo] **A + B**

push A
push B
invokevirtual plus


pop
push
Stack


pop
push
load
Local
Variable
s
Table
Stack
store


pop
push
load
Local Variables Table
Stack
store


pop
push
load
Local Variables Table
Stack
invoke
store
pop push
load
Local Variables Table
Stack
store

javap

The disassembler

javap

- Java class file disassembler
- Used with no options shows class structure only
  - Methods, superclass, interfaces, etc
- **-c** shows the bytecode
- **-private** shows all methods and members
- **-s** prints internal signatures
- **-l** prints line numbers and local variable tables
- **-verbose** for verbosity ^

```java
public class Hello {

  public static void main(String[] args) {
    System.out.println("Hello, World!");
  }

}

```

```java
public class Hello {

  public static void main(String[] args) {
    System.out.println("Hello, World!");
  }

}
```

C:\work\geecon\classes>javap Hello -c

```java
public class Hello {

  public static void main(String[] args) {
    System.out.println("Hello, World!");
  }

}
```

```
C:\work\geecon\classes>javap Hello -c
Compiled from "Hello.java"
public class Hello extends java.lang.Object{
public Hello();
```

the default constructor

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

```java
public class Hello {

  public static void main(String[] args) {
    System.out.println("Hello, World!");
  }

}
```

```
C:\work\geecon\classes>javap Hello -c
Compiled from "Hello.java"
public class Hello extends java.lang.Object{
public Hello();
   Code:
    0:   aload_0
    1:   invokespecial   #1; //Method java/lang/Object."<init>":
()V
    4:   return
```

push this to stack

```java
public class Hello {

  public static void main(String[] args) {
    System.out.println("Hello, World!");
  }

}
```

```
C:\work\geecon\classes>javap Hello -c
Compiled from "Hello.java"
public class Hello extends java.lang.Object{
public Hello();
   Code:
    0:   aload_0
    1:   invokespecial   #1; //Method java/lang/Object."<init>":
()V
    4:   return
```

Invoke <init> on this

```java
public class Hello {

  public static void main(String[] args) {
    System.out.println("Hello, World!");
  }

}
```

```
C:\work\geecon\classes>javap Hello -c
Compiled from "Hello.java"
public class Hello extends java.lang.Object{
public Hello();
    Code:
     0:   aload_0
     1:   invokespecial   #1; //Method java/lang/Object."<init>":
()V
     4:   return
```

super()

```java
public class Hello {

  public static void main(String[] args) {
    System.out.println("Hello, World!");
  }

}
```

```
C:\work\geecon\classes>javap Hello -c
Compiled from "Hello.java"
public class Hello extends java.lang.Object{
public Hello();
   Code:
    0:   aload_0
    1:   invokespecial   #1; //Method java/lang/Object."<init>":
()V
    4:   return
```

```java
public class Hello {

  public static void main(String[] args) {
    System.out.println("Hello, World!");
  }

}
```

```
C:\work\geecon\classes>javap Hello -c
Compiled from "Hello.java"
public class Hello extends java.lang.Object{
public Hello();
   Code:
    0:   aload_0
    1:   invokespecial   #1; //Method java/lang/Object."<init>":
()V
    4:   return

public static void main(java.lang.String[]);
   Code:
    0:   getstatic       #2; //Field
java/lang/System.out:Ljava/io/PrintStream;
    3:   ldc     #3; //String Hello, World!
    5:   invokevirtual   #4; //Method java/io/PrintStream.println:
```

```java
public class Hello {

  public static void main(String[] args) {
    System.out.println("Hello, World!");
  }

}
```

```
C:\work\geecon\classes>javap Hello -c
Compiled from "Hello.java"
public class Hello extends java.lang.Object{
public Hello();
   Code:
    0:   aload_0
    1:   invokespecial   #1; //Method java/lang/Object."<init>":
()V
    4:   return

public static void main(java.lang.String[]);
   Code:
    0:   getstatic       #2; //Field
java/lang/System.out:Ljava/io/PrintStream;
    3:   ldc     #3; //String Hello, World!
    5:   invokevirtual   #4; //Method java/io/PrintStream.println:
```

get static field

```java
public class Hello {

  public static void main(String[] args) {
    System.out.println("Hello, World!");
  }

}
```

```
C:\work\geecon\classes>javap Hello -c
Compiled from "Hello.java"
public class Hello extends java.lang.Object{
public Hello();
   Code:
    0:   aload_0
    1:   invokespecial   #1; //Method java/lang/Object."<init>":
()V
    4:   return

public static void main(java.lang.String[]);
   Code:
    0:   getstatic       #2; //Field
java/lang/System.out:Ljava/io/PrintStream;
    3:   ldc             #3; //String Hello, World!
    5:   invokevirtual   #4; //Method java/lio/PrintStream.println:
```

load string to the stack

```java
public class Hello {

  public static void main(String[] args) {
    System.out.println("Hello, World!");
  }

}
```

```
C:\work\geecon\classes>javap Hello -c
Compiled from "Hello.java"
public class Hello extends java.lang.Object{
public Hello();
   Code:
    0:   aload_0
    1:   invokespecial   #1; //Method java/lang/Object."<init>":
()V
    4:   return

public static void main(java.lang.String[]);
   Code:
    0:   getstatic       #2; //Field
java/lang/System.out:Ljava/io/PrintStream;
    3:   ldc     #3; //String Hello, World!
    5:   invokevirtual   #4; //Method java/io/PrintStream.println:
```

Invoke method with parameter

```java
public class Hello {

  public static void main(String[] args) {
    System.out.println("Hello, World!");
  }

}
```

```
C:\work\geecon\classes>javap Hello -c
Compiled from "Hello.java"
public class Hello extends java.lang.Object{
public Hello();
   Code:
    0:   aload_0
    1:   invokespecial   #1; //Method java/lang/Object."<init>":
()V
    4:   return

public static void main(java.lang.String[]);
   Code:
    0:   getstatic       #2; //Field
java/lang/System.out:Ljava/io/PrintStream;
    3:   ldc     #3; //String Hello, World!
    5:   invokevirtual   #4; //Method java/io/PrintStream.println:
```

```java
public class Hello {

  public static void main(String[] args) {
    System.out.println("Hello, World!");
  }

}
```

What's #1,#2, etc

```
C:\work\geecon\classes>javap Hello -c
Compiled from "Hello.java"
public class Hello extends java.lang.Object{
public Hello();
   Code:
    0:   aload_0
    1:   invokespecial   #1; //Method java/lang/Object."<init>":
()V
    4:   return

public static void main(java.lang.String[]);
   Code:
    0:   getstatic       #2; //Field
java/lang/System.out:Ljava/io/PrintStream;
    3:   ldc     #3; //String Hello, World!
    5:   invokevirtual   #4; //Method java/io/PrintStream.println:
```

SLIDES

GOTO: IDE

SLIDES

IDE: **JAVAP** DEMO

ASM

The *de facto* standard for bytecode manipulation

ASM

- “All purpose bytecode manipulation and analysis framework”
- De facto standard bytecode library
- http://asm.ow2.org

Basic Process

- Construct ClassWriter
- Stack up the ***visitors*** for:
  - annotations, methods, fields, etc
- Write out bytes

Hello.java

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

ClassWriter

```
ClassWriter cw = new ClassWriter
      ClassWriter.COMPUTE_MAXS |
      ClassWriter.COMPUTE_FRAMES)
```

COMPUTE_***

- COMPUTE_MAXS
  - ASM will calculate max stack/local vars
- COMPUTE_FRAMES
  - ASM will calculate Java 6 stack map

Visit Class

```
cv.visit(V1_6,
  ACC_PUBLIC,
  "X",
  null,
  "java/lang/Object",
  null);
```

Opcodes

- Interface full of constants
  - Bytecodes
  - Visibility modifiers
  - Java versions
  - Other stuff

ACC_***

- Some you know
  - ACC_PUBLIC, ACC_ABSTRACT, etc
- Some you (probably) don’t
  - ACC_BRIDGE, ACC_SYNTHETIC

Class Names

"java/lang/Object"

packageClass.replaceAll('.', '/')

Type Descriptors

Type Descriptors

B	**byte**
C	**char**
S	**string**
I	**int**
J	**long**
F	**float**
D	**double**
Z	**boolean**
V	**void**

Type Descriptors

Lsome/Class;

Type Descriptors

[Lsome/Class;

Method Signatures

()V void foo()

(Ljava/lang/Object;)I int foo(Object)

([Ljava/lang/String;)V void main(String[])

Visit Method

```
MethodVisitor constructor =
    cv.visitMethod(ACC_PUBLIC,
    "<init>",
    "()V",
    null,
    null);

MethodVisitor mv = cv.visitMethod(
  ACC_PUBLIC + ACC_STATIC,
  "main",
  "([Ljava/lang/String;)V",
  null,
  null);
```

Visit Method

```
MethodVisitor constructor =
    cv.visitMethod(ACC_PUBLIC,
    "<init>",
    "()V",
    null,
    null);
```

Wat!? o_O

```
MethodVisitor mv = cv.visitMethod(
  ACC_PUBLIC + ACC_STATIC,
  "main",
  "([Ljava/lang/String;)V",
  null,
  null);
```

Special Methods

- <init>
  - Constructor
- <clinit>
  - Static initializer

MethodVisitor

- Visit annotations
- Visit code
  - Bytecodes, local variables, line numbers, etc
- Visit maxs
  - Pass bogus values if COMPUTE_MAX

Constructor

```
c.visitVarInsn(ALOAD, 0);

c.visitMethodInsn(INVOKESPECIAL,
    "java/lang/Object", "<init>",
"()V");
c.visitInsn(RETURN);
c.visitMaxs(0, 0);
```

Constructor

```
c.visitVarInsn(ALOAD, 0);

c.visitMethodInsn(INVOKESPECIAL,
    "java/lang/Object", "<init>",
"()V");
    c.visitInsn(RETURN);
    c.visitMaxs(0, 0);
```

```
aload_0
invokespeci
al
```

public static void main()

public static void main()

```
mv.visitFieldInsn(GETSTATIC,
    "java/lang/System", "out",
    "Ljava/io/PrintStream;");

mv.visitLdcInsn("Hello");

mv.visitMethodInsn(INVOKEVIRTUAL,
    "java/io/PrintStream", "println",
    "(Ljava/lang/String;)V");

mv.visitInsn(RETURN);
```

public static void main()

```
mv.visitFieldInsn(GETSTATIC,
    "java/lang/System", "out",
    "Ljava/io/PrintStream;");

mv.visitLdcInsn("Hello");

mv.visitMethodInsn(INVOKEVIRTUAL,
    "java/io/PrintStream", "println",
    "(Ljava/lang/String;)V");

mv.visitInsn(RETURN);
```

getstatic
ldc
“Hello”
invokevirtual

Enter Loops

```
public static void main(String[] args) {
    for(int i = 0; i < 10; i ++)
        System.out.println("Hello");
}
```

Enter Loops

```java
public static void main(String[] args) {
    for(int i = 0; i < 10; i ++)
        System.out.println("Hello");
}
```

```
start:
    int i = 0
loop:
    print "Hello"
    i = i + 1
    if i < 10
goto loop
end: return
```

Enter Loops

```
public static void main(String[] args) {
    for(int i = 0; i < 10; i ++)
        System.out.println("Hello");
}
```

```
start:
    int i = 0
loop:
    print "Hello"
    i = i + 1
    if i < 10
goto loop
end: return
```

GOTO isn't harmful ;)

Enter Loops

```
0: iconst_0
1: istore_1
2: iload_1
3: bipush 10
5: if_icmpge  22
```

```
System.out.println("Hell
o")
```

```
16:  iinc  1, 1
```

Enter Loops

```
start:  iconst_0
   1: istore_1
loop:   iload_1
   3: bipush 10
   5: if_icmpge   end
```

```
System.out.println("Hell
o")
```

```
   16:  iinc  1, 1
```

Enter Loops

```
start:  iconst_0
   1: istore_1
loop:   iload_1
   3: bipush 10
   5: if_icmpge   end
```

int i = 0

```
System.out.println("Hell
o")

   16:  iinc  1, 1
```

Enter Loops

```
start:  iconst_0
   1: istore_1
loop:   iload_1
   3: bipush 10
   5: if_icmpge   end
```

i < 10

System.out.println(“Hell
o”)

```
   16:  iinc  1, 1
```

Enter Loops

```
start:  iconst_0
   1: istore_1
loop:   iload_1
   3: bipush 10
   5: if_icmpge   end
```

```
System.out.println("Hell
o")
```

i++

```
   16:  iinc  1, 1
```

Enter Loops

```
start:  iconst_0
   1: istore_1
loop:   iload_1
   3: bipush 10
   5: if_icmpge   end
```

```
System.out.println("Hell
o")
```

```
   16:  iinc  1, 1
```

Enter ASM Loops

```
Label start = new Label();
Label loop = new Label();
Label end = new Label();

// i = 0
mv.visitLabel(start);
mv.visitInsn(ICONST_0);
mv.visitVarInsn(ISTORE, 1);
```

Enter ASM Loops

```
Label start = new Label();
Label loop = new Label();
Label end = new Label();

// i = 0
mv.visitLabel(start);
mv.visitInsn(ICONST_0);
mv.visitVarInsn(ISTORE, 1);

// i < 10
mv.visitLabel(loop);
mv.visitVarInsn(ILOAD, 1);
mv.visitLdcInsn(10);
mv.visitJumpInsn(IF_ICMPGE,
end);
```

Enter ASM Loops

```
Label start = new Label();
Label loop = new Label();
Label end = new Label();

// i = 0
mv.visitLabel(start);
mv.visitInsn(ICONST_0);
mv.visitVarInsn(ISTORE, 1);

// i < 10
mv.visitLabel(loop);
mv.visitVarInsn(ILOAD, 1);
mv.visitLdcInsn(10);
mv.visitJumpInsn(IF_ICMPGE,
end);
```

```
//increment & continue
the loop
mv.visitIincInsn(1, 1);
mv.visitJumpInsn(GOTO,
loop);
mv.visitLabel(end);
```

```java
byte[] classBytes = cv.toByteArray();
FileOutputStream fos =
        new FileOutputStream(new File("./Hello.class"));
fos.write(classBytes);
fos.close();
```

ClassWriter

```
byte[] classBytes = cv.toByteArray();
FileOutputStream fos =
        new FileOutputStream(new File("./Hello.class"));
fos.write(classBytes);
fos.close();
```

```
geecon$ java Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
```

ASMifier

ASMifierClassVisitor

```
a -cp asm-all-3.3.1.jar:asm-util-3.3.1.ja
.objectweb.asm.util.ASMifierClassVisitor
lo.class
```

```
mv = cw.visitMethod(ACC_PUBLIC, "getI", "(I)I", null, null);
mv.visitCode();
Label l0 = new Label();
mv.visitLabel(l0);
mv.visitLineNumber(16, l0);
mv.visitVarInsn(ALOAD, 0);
mv.visitFieldInsn(GETFIELD, "zt/asm/Items", "ids",
"Ljava/util/List;");
mv.visitVarInsn(ILOAD, 1);
mv.visitMethodInsn(INVOKEINTERFACE, "java/util/List",
                                    "get",
"(I)Ljava/lang/Object;");
mv.visitTypeInsn(CHECKCAST, "java/lang/Integer");
mv.visitMethodInsn(INVOKEVIRTUAL, "java/lang/Integer",
                                  "intValue", "()I");
mv.visitInsn(IRETURN);
Label l1 = new Label();
mv.visitLabel(l1);
mv.visitLocalVariable("this", "Lzt/asm/Items;", null, l0, l1, 0);
mv.visitLocalVariable("i", "I", null, l0, l1, 1);
```

Bytecode instrumentation

Some magic for your own good

WAT!?

Ninja.class

101010101
011100010
101010101
010001000
100011101

Ninja.class’

101010101
0111**10000**
101010101
010001000
100011101
111
0

Who?

AspectJ

Containers (Java EE, Spri

Play! Framework

Terracotta

JRebel

FindBugs

Hibernate

Tapestry

Byteman

How?

- Add **–javaagent** to hook into class loading process
- Implement **ClassFileTransformer**
- Use bytecode manipulation libraries (**Javassist**, **cglib**, **asm**) to add any custom logic

java.lang.instrument

How ? (2)

- Use custom ClassLoader
  - Override **ClassLoader#findClass**
  - Use **ClassReader(String)** to read the class in and transform it via *visitor chain*
  - Call **ClassLoader#defineClass** explicitly with the result from the transformation step

java.lang.instrument

```
import java.lang.instrument.ClassFileTransformer;
import java.lang.instrument.Instrumentation;

public class Agent {
public static void premain(String args, Instrumentation
inst)
  { inst.addTransformer(new ClassFileTransformer(), true);
}

public static void agentmain(String args, Instrumentation
inst)
  { premain(args,inst); }
}
```

java.lang.instrument

```
import java.lang.instrument.ClassFileTransformer;
import java.lang.instrument.Instrumentation;

public class Agent {
public static void premain(String args, Instrumentation
inst)
  { inst.addTransformer(new ClassFileTransformer(), true);
}

public static void agentmain(String args,
Instrumentation inst)
  { premain(args,inst); }
}
```

java.lang.instrument

```
import java.lang.instrument.ClassFileTransformer;
import java.lang.instrument.Instrumentation;

public class Agent {
public static void premain(String args, Instrumentation
inst)
  { inst.addTransformer(new ClassFileTransformer(), true);
}

public static void agentmain(String args,
Instrumentation inst)
  { premain(args, inst); }
}
```

META-INF/MANIFEST.MF
Premain-Class: **Agent**
Agent-Class: **Agent**

java –javaagent:agent.jar
…

j.l.instrument.**ClassFileTransformer**

```
new ClassFileTransformer() {
  public byte[] transform(ClassLoader loader, String clas
                          Class<?>classBeingRedefined,
                          ProtectionDomain protectionDomair
                              byte[] classfileBuffer){

  ClassReader cr = new ClassReader(classfileBuffer);
  ClassWriter cw = new ClassWriter(cr,
                   ClassWriter.COMPUTE_MAXS |
                   ClassWriter.COMPUTE_FRAMES);
  MyAdapter ca = new MyAdapter(cw);
  cr.accept(ca, ClassReader.EXPAND_FRAMES);
  return cw.toByteArray();
}
```

j.l.instrument.**ClassFileTransformer**

```
new ClassFileTransformer() {
  public byte[] transform(ClassLoader loader, String clas
                          Class<?>classBeingRedefined,
                          ProtectionDomain protectionDomain
                             byte[] classfileBuffer){

  ClassReader cr = new ClassReader(classfileBuffer);
  ClassWriter cw = new ClassWriter(cr,
                   ClassWriter.COMPUTE_MAXS |
                   ClassWriter.COMPUTE_FRAMES);
  MyAdapter ca = new MyAdapter(cw);
  cr.accept(ca, ClassReader.EXPAND_FRAMES);
  return cw.toByteArray();
}
```

j.l.instrument.**ClassFileTransformer**

```
new ClassFileTransformer() {
  public byte[] transform(ClassLoader loader, String clas
                          Class<?>classBeingRedefined,
                          ProtectionDomain protectionDomain
                             byte[] classfileBuffer){

  ClassReader cr = new ClassReader(classfileBuffer);
  ClassWriter cw = new ClassWriter(cr,
                   ClassWriter.COMPUTE_MAXS |
                   ClassWriter.COMPUTE_FRAMES);
  MyAdapter ca = new MyAdapter(cw)
  cr.accept(ca, ClassReader.EXPAND_FRAMES);
  return cw.toByteArray();
}
```

```
public class MyClassLoader extends ClassLoader {

  protected Class findClass(String name)
                                  throws
ClassNotFoundException {

      ClassReader cr = new ClassReader(name);
      ClassWriter cw = new ClassWriter(cr,
                                  ClassWriter.COMPUTE_MAXS
|

ClassWriter.COMPUTE_FRAMES);

      MyClassAdapter ca =
                  new
MyClassAdapter(cw);

      cr.accept(ca, ClassReader.EXPAND_FRAMES);
```

SLIDES

GOTO: IDE

SLIDES

IDE: **ASM** DEMO

@antonarhipov
anton@zeroturnaround.com
https://github.com/antonarhipov/

asmdemo