Java Fundamentals

Lecture 12 - Java Bytecode and Javassist

Sven Laanela
@svenlaanela

17.04.2017

Previously on ...

- Metaprogramming
- Reflection
- Dynamic Proxies
- Homework

Agenda for Today

- Java Bytecode
- Javassist
- Homework :)

Java Bytecode

Why do we care?

- Want to understand what happens under the hood
- Want to solve problems when the JVM blows up with a cryptic looking message
- Want to be better engineers

Goal

- We have a simple **.class** file
- What is inside the class file?
- Lets modify a method and add an exclamation mark
- But lets do it directly on the .class file, without any decompiling, etc.

Simple class file

```
package org.zeroturnaround.jf;

public class HelloWorld {
  public String sayHello() {
    return "Hello world";
  }
}
```

javac org/zeroturnaround/jf/HelloWorld.java

Class runner

```
package org.zeroturnaround.jf;

public class Main {
  public static void main(String[] args) {
    HelloWorld hello = new HelloWorld();
    System.out.println(hello.sayHello());
  }
}
```

Demo time

Class file format

- u4 - magic; 0x**CAFEBABE**
- u2 - minor_version; 0x**0000**
- u2 - major_version; 0x**0034** -> Java 8; (0x**0035** hex) -> Java 9
- u2 - constant_pool_count; 0x**0014** -> // 20 for normal people

Constant Pool

- Most of the information in a class file is disseminated into the constant pool
- String constants, integer constants, class info, method info, etc.

Constant pool tags

- **CONSTANT_Class** 7
- **CONSTANT_Methodref** 10
- **CONSTANT_String** 8
- **CONSTANT_NameAndType** 12
- **CONSTANT_Utf8** 1
- **CONSTANT_MethodHandle** 15
- More: https://docs.oracle.com/javase/specs/jvms/se7/html/jvms-4.html#jvms-4.4

Const. pool #1

- **CONSTANT_Methodref_info {**
  - **u1** tag; 0x**0A**
  - **u2** class_index; 0x**0004**
  - **u2** name_and_type_index; 0x**0010**
- **}**

Const. pool #2

- **CONSTANT_String_info {**
  - **u1** tag; 0x**08**
  - **u2** string_index; 0x**0011** -> 17
- **}**

Const. pool #3

- **CONSTANT_Class_info {**
  - **u1** tag; 0x**07**
  - **u2** name_index; 0x**0012** -> 18
- **}**

Const. pool #4

- **CONSTANT_Class_info {**
  - **u1** tag; 0x**07**
  - **u2** name_index; 0x**0013** -> 19
- **}**

Const. pool #5

- **CONSTANT_Utf8_info {**
  - **u1** tag; 0x**01**
  - **u2** length; 0x**0006**
  - **u1** bytes[length]; 0x**3C69...743E** -> “<init>”
- **}**

Const. pool #6

- **CONSTANT_Utf8_info {**
  - **u1** tag; 0x**01**
  - **u2** length; 0x**0003**
  - **u1** bytes[length]; 0x**282956** -> “()V”
- **}**

Const. pool #7

- **CONSTANT_Utf8_info {**
  - **u1** tag; 0x**01**
  - **u2** length; 0x**0004**
  - **u1** bytes[length]; 0x**436F6465** -> “Code”
- **}**

Const. pool #8

- **CONSTANT_Utf8_info {**
  - **u1** tag; 0x**01**
  - **u2** length; 0x**000F**
  - **u1** bytes[length]; 0x**4C69...6C65** -> “LineNumberTable”
- **}**

Const. pool #9

- **CONSTANT_Utf8_info {**
  - **u1** tag; 0x**01**
  - **u2** length; 0x**0012**
  - **u1** bytes[length]; 0x**4C6F...6C65** -> “LocalVariableTable”
- **}**

Const. pool #10

- **CONSTANT_Utf8_info {**
  - **u1** tag; 0x**01**
  - **u2** length; 0x**0004**
  - **u1** bytes[length]; 0x**74686973** -> “this”
- **}**

Const. pool #11

- **CONSTANT_Utf8_info {**
  - **u1** tag; 0x**01**
  - **u2** length; 0x**0022**
  - **u1** bytes[length]; 0x**4C6F...643B** -> “Lorg/zeroturnaround/jf/HelloWorld;”
- **}**

Const. pool #12

- **CONSTANT_Utf8_info {**
  - **u1** tag; 0x**01**
  - **u2** length; 0x**0008**
  - **u1** bytes[length]; 0x**7361...6C6F** -> “sayHello”
- **}**

Const. pool #13

- **CONSTANT_Utf8_info {**
  - **u1** tag; 0x**01**
  - **u2** length; 0x**0014**
  - **u1** bytes[length]; 0x**2829...673B** -> “()Ljava/lang/String;”
- **}**

Const. pool #14

- **CONSTANT_Utf8_info {**
  - **u1** tag; 0x**01**
  - **u2** length; 0x**000A**
  - **u1** bytes[length]; 0x**536F...6C65** -> “SourceFile”
- **}**

Const. pool #15

- **CONSTANT_Utf8_info {**
  - **u1** tag; 0x**01**
  - **u2** length; 0x**000F**
  - **u1** bytes[length]; 0x**4865...7661** -> “HelloWorld.java”
- **}**

Const. pool #16

- **CONSTANT_NameAndType_info {**
  - **u1** tag; 0x**0C**
  - **u2** name_index; 0x**0005** -> “<init>”
  - **u2** descriptor_index; 0x**0006** “()V”
- **}**

Const. pool #17

- **CONSTANT_Utf8_info {**
  - **u1** tag; 0x**01**
  - **u2** length; 0x**000B**
  - **u1** bytes[length]; 0x**4865...6C64** -> “Hello world”
- **}**

Const. pool #18

- **CONSTANT_Utf8_info {**
  - **u1** tag; 0x**01**
  - **u2** length; 0x**0020**
  - **u1** bytes[length]; 0x**6F72...6C64** -> “org/zeroturnaround/jf/HelloWorld”
- **}**

Const. pool #19

- **CONSTANT_Utf8_info {**
  - **u1** tag; 0x**01**
  - **u2** length; 0x**0010**
  - **u1** bytes[length]; 0x**6A61...6374** -> “java/lang/Object”
- **}**

Class file format cont.d

- **u2** - access_flags; 0x**0021**
- **u2** - this_class; 0x**0003**
- **u2** - super_class; 0x**0004**
- **u2** - interfaces_count; 0x**0000**
- **u2** - interfaces[interfaces_count]; none

Class file format cont.d

- **u2** - fields_count; 0x**0000**
- **field_info** - fields[fields_count];
- **u2** - methods_count; 0x**0002**
- **method_info** - methods[methods_count]; ...

Method info #1

- **method_info {**
  - **u2** - access_flags; 0x**0001**
  - **u2** - name_index; 0x**0005** -> “<init>”
  - **u2** - descriptor_index; 0x**0006** -> “()V”
  - **u2** - attributes_count; 0x**0001**
  - **attribute_info** - attributes[attributes_count];
- **}**

Attributes info #1

- **attribute_info {**
  - **u2** - attribute_name_index; 0x**0007** -> “Code” :)
  - **u4** - attribute_length; 0x**0000002F**
  - **u1** - info[attribute_length]; 0x**00010001 00000005 2AB70001 B1000000 02000800 00000600 01000000 03000900 00000C00 01000000 05000A00 0B0000**
- **}**

Code

- **u2** - max_stack; 0x**0001**
- **u2** - max_locals; 0x**0001**
- **u4** - code_length; 0x**00000005**
- **u1** - code[code_length]; 0x**2AB70001B1**
- **u2** - exception_table_length; 0x**0000**
- **u2** - attributes_count; 0x**0002**
- **attribute_info** - attributes[attributes_count];

Actual code

- 0x**2AB70001B1**
- 0x**2A** - aload_0
- 0x**B7** ***0001*** - invokespecial “java/lang/Object” “<init>” “()V”
- 0x**B1** - return

Code attributes info #1

- **attribute_info {**
  - **u2** - attribute_name_index; 0x**0008** -> “LineNumberTable”
  - **u4** - attribute_length; 0x**00000006**
  - **u1** - info[attribute_length]; 0x**000100000003**
- **}**

Code attributes info #2

- **attribute_info {**
  - **u2** - attribute_name_index; 0x**0009** -> “LocalVariableTable”
  - **u4** - attribute_length; 0x**0000000C**
  - **u1** - info[attribute_length]; 0x**00010000 0005000A 000B0000**
- **}**

Method info #2

- **method_info {**
  - **u2** - access_flags; 0x**0001**
  - **u2** - name_index; 0x**000C** -> “sayHello”
  - **u2** - descriptor_index; 0x**000D** -> “()Ljava/lang/String;”
  - **u2** - attributes_count; 0x**0001**
  - **attribute_info** - attributes[attributes_count];
- **}**

Attributes info #2

- **attribute_info {**
  - **u2** - attribute_name_index; 0x**0007** -> “Code” :)
  - **u4** - attribute_length; 0x**0000002D**
  - **u1** - info[attribute_length]; 0x**00010001 00000003 1202B000 00000200 08000000 06000100 00000500 09000000 0C000100 00000300 0A000B00 00**
- **}**

Code

- **u2** - max_stack; 0x**0001**
- **u2** - max_locals; 0x**0001**
- **u4** - code_length; 0x**00000003**
- **u1** - code[code_length]; 0x**1202B0**
- **u2** - exception_table_length; 0x**0000**
- **u2** - attributes_count; 0x**0002**
- **attribute_info** - attributes[attributes_count];

Actual code

- 0x**1202B0**
- 0x**12** ***02*** - ldc “Hello world”; Push a constant from a constant pool onto stack
- 0x**B0** - areturn; return a reference from a method

Code attributes info #1

- **attribute_info {**
  - **u2** - attribute_name_index; 0x**0008** -> “LineNumberTable”
  - **u4** - attribute_length; 0x**00000006**
  - **u1** - info[attribute_length]; 0x**000100000005**
- **}**

Code attributes info #1

- **attribute_info {**
  - **u2** - attribute_name_index; 0x**0008** -> “LineNumberTable”
  - **u4** - attribute_length; 0x**00000006**
  - **u1** - info[attribute_length]; 0x**000100000005**
- **}**

Code attributes info #2

- **attribute_info {**
  - **u2** - attribute_name_index; 0x**0009** -> “LocalVariableTable”
  - **u4** - attribute_length; 0x**0000000C**
  - **u1** - info[attribute_length]; 0x**00010000 0003000A 000B0000**
- **}**

Attributes info #3

- **attribute_info {**
  - **u2** - attribute_name_index; 0x**000E** -> “SourceFile”
  - **u4** - attribute_length; 0x**00000002**
  - **u1** - info[attribute_length]; 0x**000F** -> “HelloWorld.java”
- **}**

Wasn’t that fun!?

https://docs.oracle.com/javase/specs/jvms/se7/html/jvms-4.html

javap -verbose HelloWorld_simple.java

Lets add an exclamation mark

- Modify Const. pool #17
- **CONSTANT_Utf8_info {**
  - **u1** tag; 0x**01**
  - **u2** length; 0x**000B** -> 0x**000C**
  - **u1** bytes[length]; 0x**4865...6C64** -> 0x**4865...6C6421**
- **}**

Add “bytecode” string

- **“Hello world!”** -> **“bytecode”**
- Add **“bytecode”** string to **const. pool**
- Modify code attribute to return that constant instead before returning

Add “bytecode” to const pool

- const pool length **+2**

Const. pool #20

- **CONSTANT_String_info {**
  - **u1** tag; 0x**08**
  - **u2** string_index; 0x**0015** -> 21
- **}**

Const. pool #21

- **CONSTANT_Utf8_info {**
  - **u1** tag; 0x**01**
  - **u2** length; 0x**0008**
  - **u1** bytes[length]; 0x**62797465636F6465** -> “bytecode”
- **}**

Modify “Code” attribute

- **u2** - max_stack; 0x**0001**
- **u2** - max_locals; 0x**0001**
- **u4** - code_length; 0x**00000003**
- **u1** - code[code_length]; 0x**1202B0** -> 0x**1214B0**
- **u2** - exception_table_length; 0x**0000**
- **u2** - attributes_count; 0x**0002**
- attribute_info attributes[attributes_count];

Lets add a method to concat two strings

OK, let’s not… :)

JVM as a stack machine

- The **JVM** is a Stack Machine
- Each method invocation creates a **new Frame**
- Each **frame** has their own
  - **Operand stack**
  - **Array of locals**

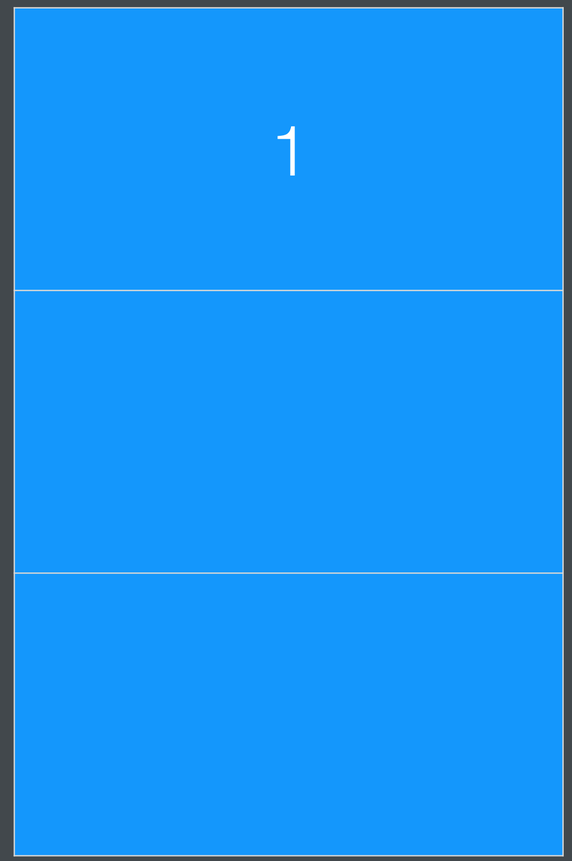
JVM as a stack machine

- Regular notation
  - **1 + 2**
- Reverse Polish notation
  - **1 2 +**

JVM as a stack machine

- Regular notation
  - **1 + 2**
- Reverse Polish notation
  - **1 2 +**

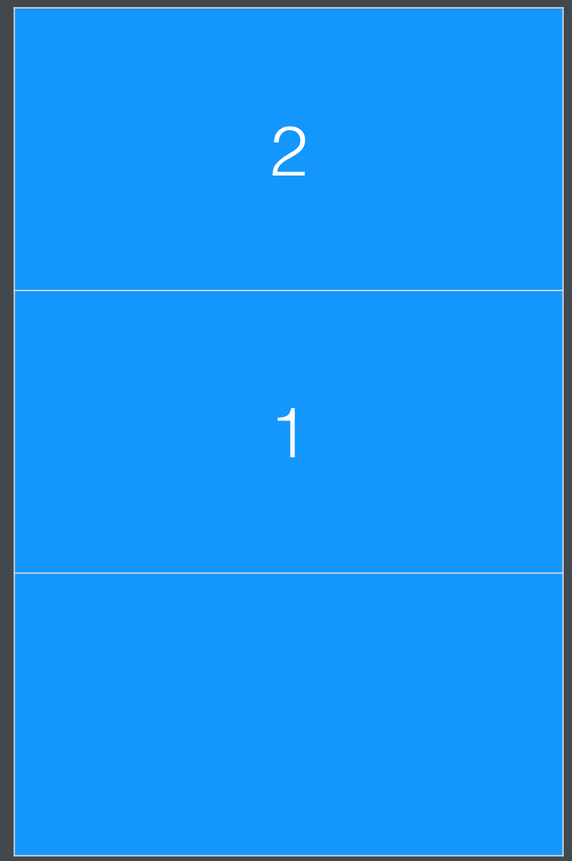
PUSH 1



JVM as a stack machine

- Regular notation
  - **1 + 2**
- Reverse Polish notation
  - **1 2 +**

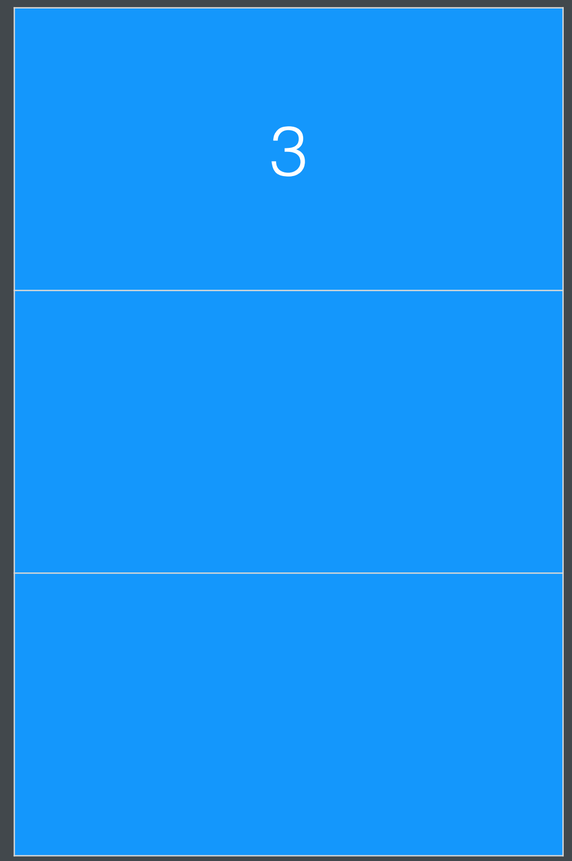
PUSH 1
PUSH 2



JVM as a stack machine

- Regular notation
  - **1 + 2**
- Reverse Polish notation
  - **1 2 +**

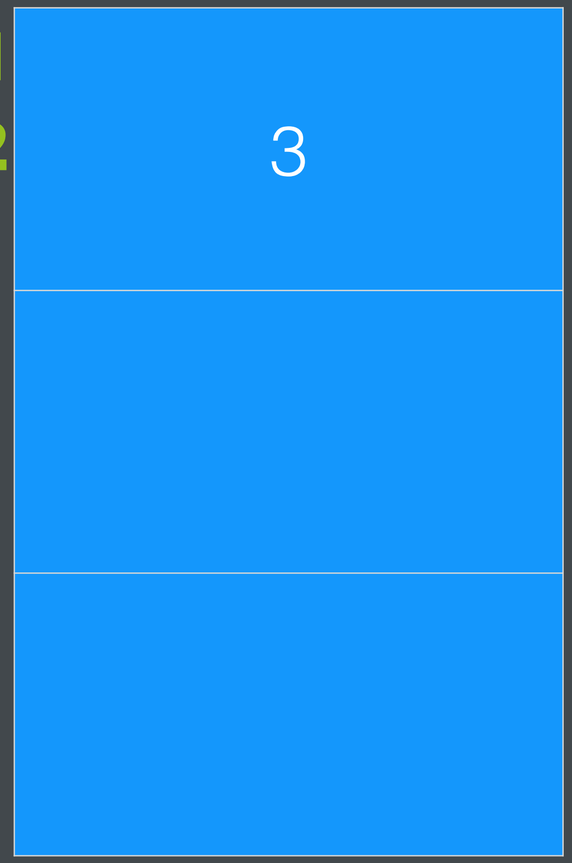
PUSH 1
PUSH 2
ADD



JVM as a stack machine

- Regular notation
  - **1 + 2**
- Reverse Polish notation
  - **1 2 +**

ICONST_1
ICONST_2
IADD



Type categories

- JVM is typesafe
  - Opcodes must match type
- Opcode categories
  - **I** 8-32 bit integer (1 stack slot)
  - **L** 64 bit integer (2 stack slots)
  - **F** 32 bit float (1 stack slot)
  - **D** 64 bit float (2 stack slots)
  - **A** Reference types (1 stack slot)
  - **?A** Arrays (1 stack slot)

Locals

- Method parameters are stored in **locals**
- For **instance**-methods
  - **“this”** is stored in **slot 0**
  - First parameter is in **slot 1**
- For **static** methods
  - First parameter is in **slot 0**

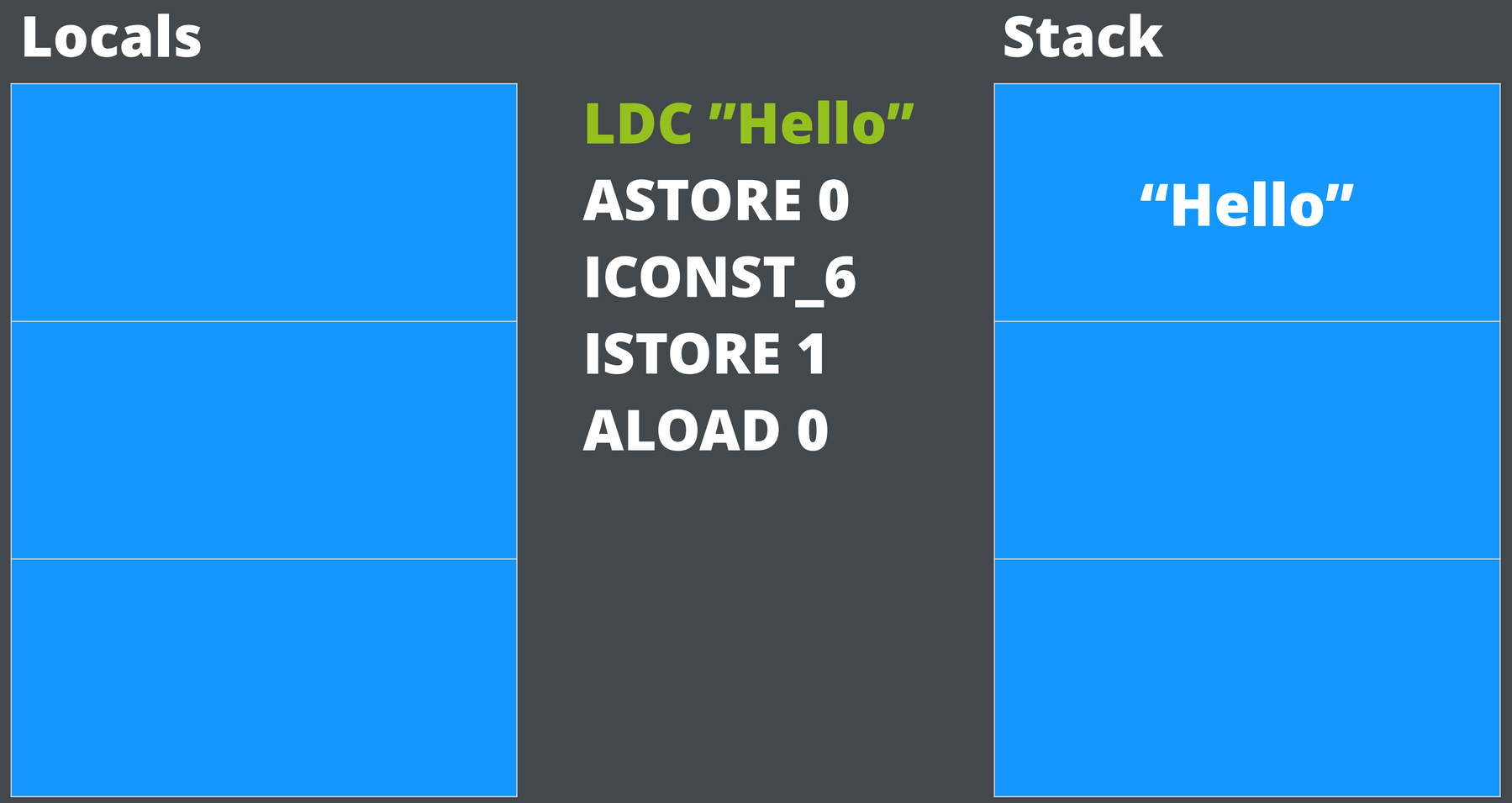
Locals

- Local variables are also stored in locals
- double and long take up two locals

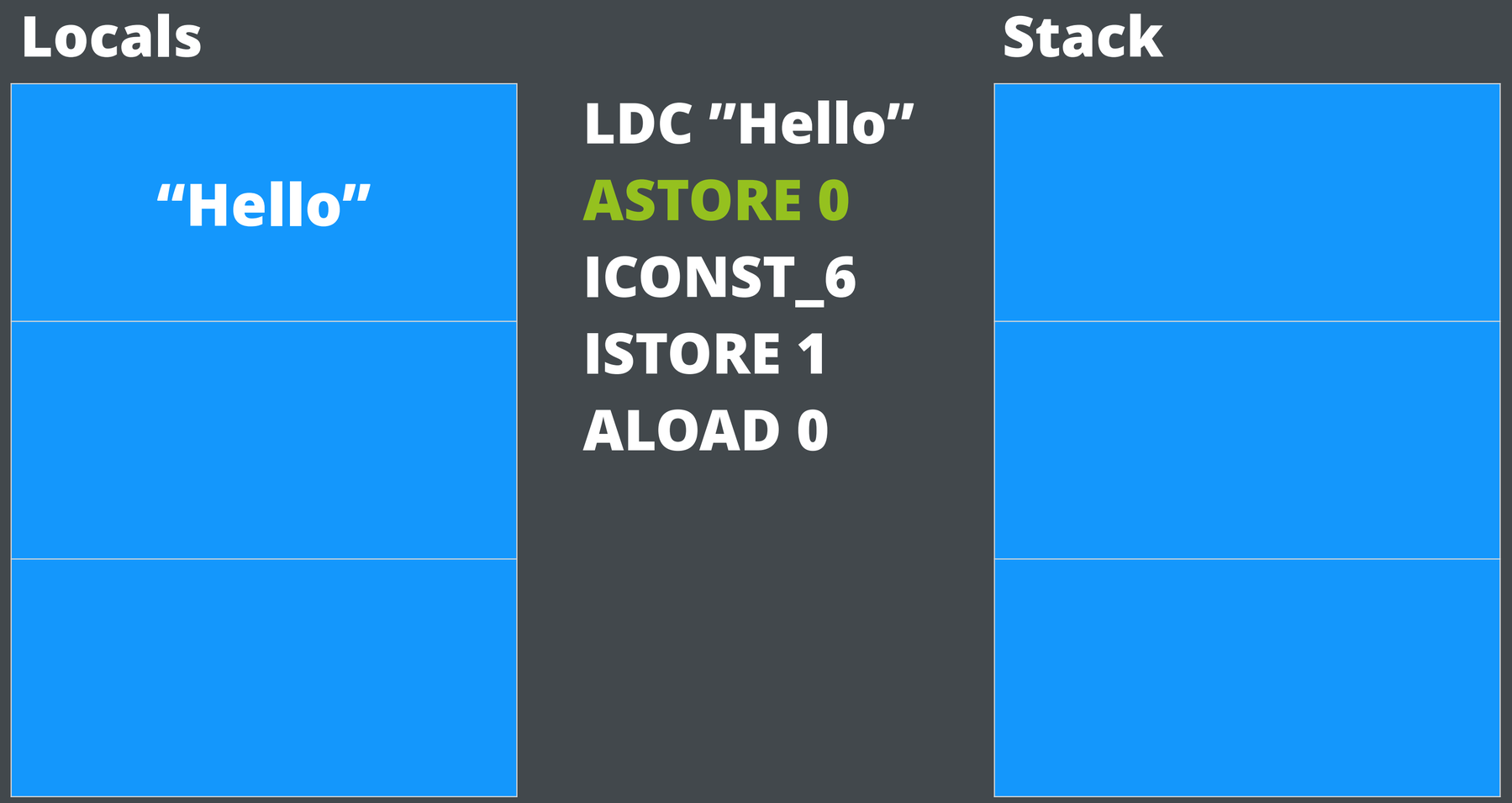
Locals

- Locals are confined to the frame
  - Entering a new frame creates a new list of locals exclusive to that frame
  - Same applies to the operand stack
- Locals retain value while the frame is alive
  - A frame is destroyed when the method exits

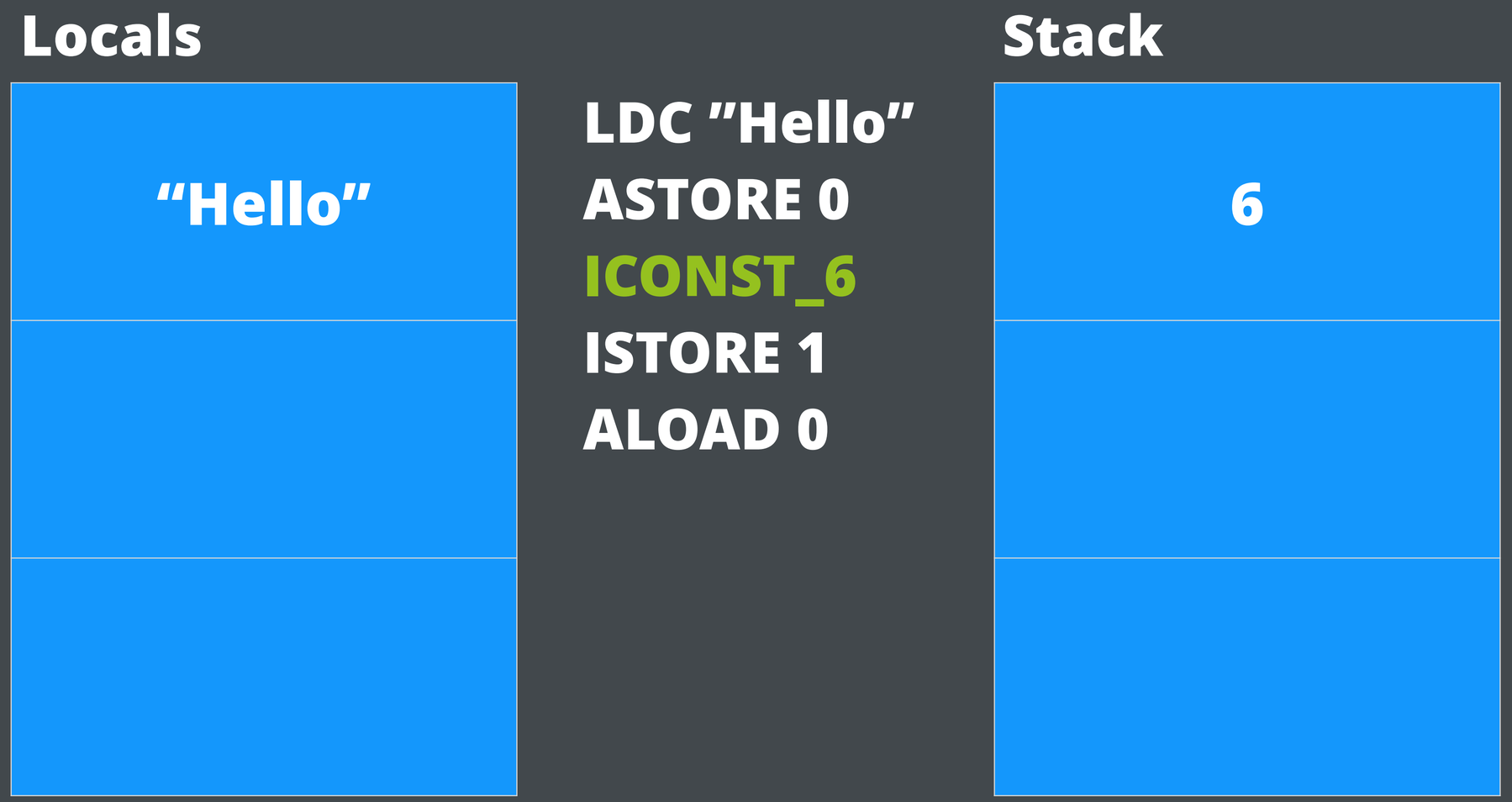
Locals


Locals
Stack
LDC ”Hello”
ASTORE 0
ICONST_6
ISTORE 1
ALOAD 0
“Hello”

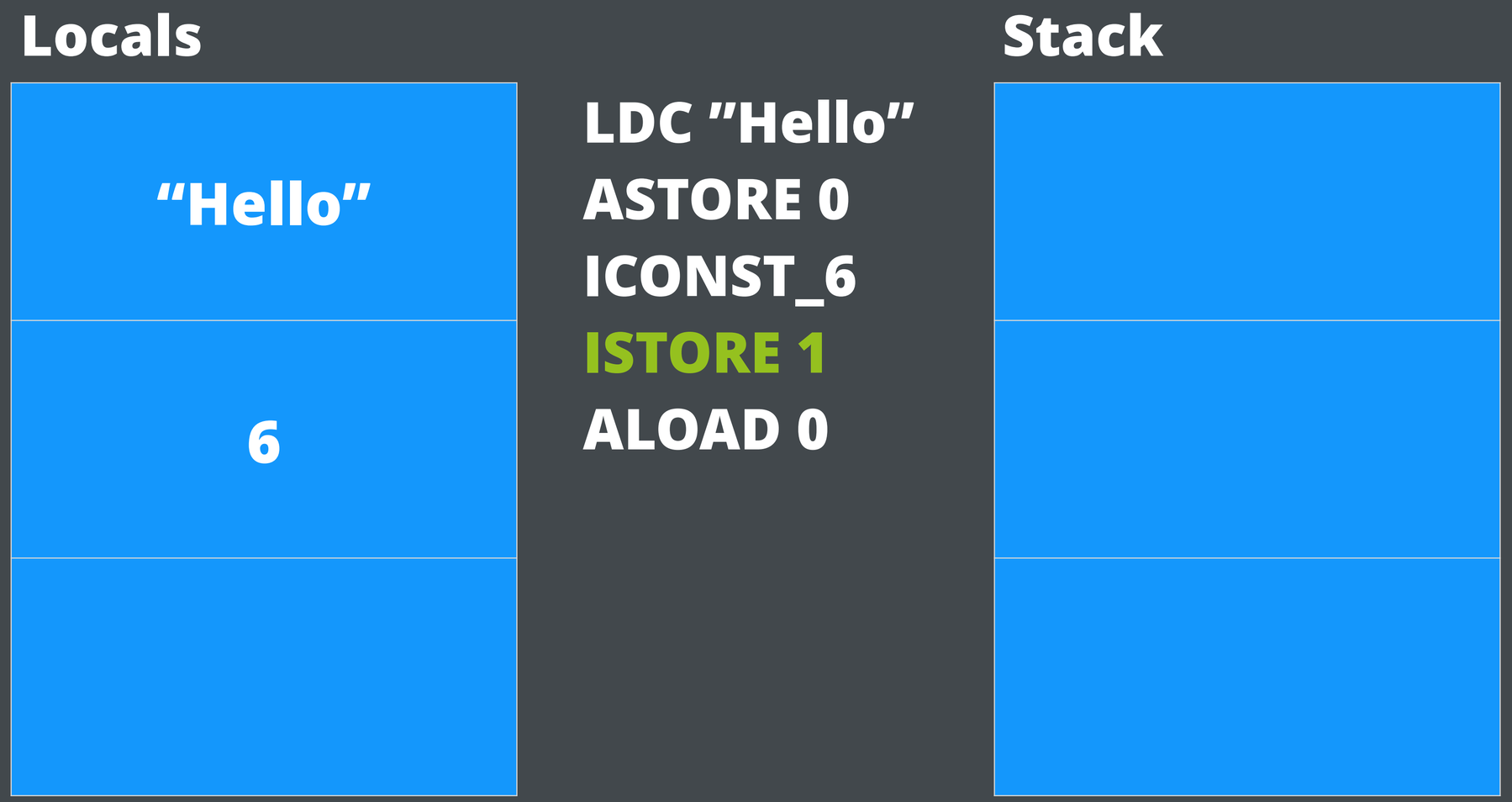
Locals


Locals
“Hello”
LDC ”Hello”
ASTORE 0
ICONST_6
ISTORE 1
ALOAD 0
Stack

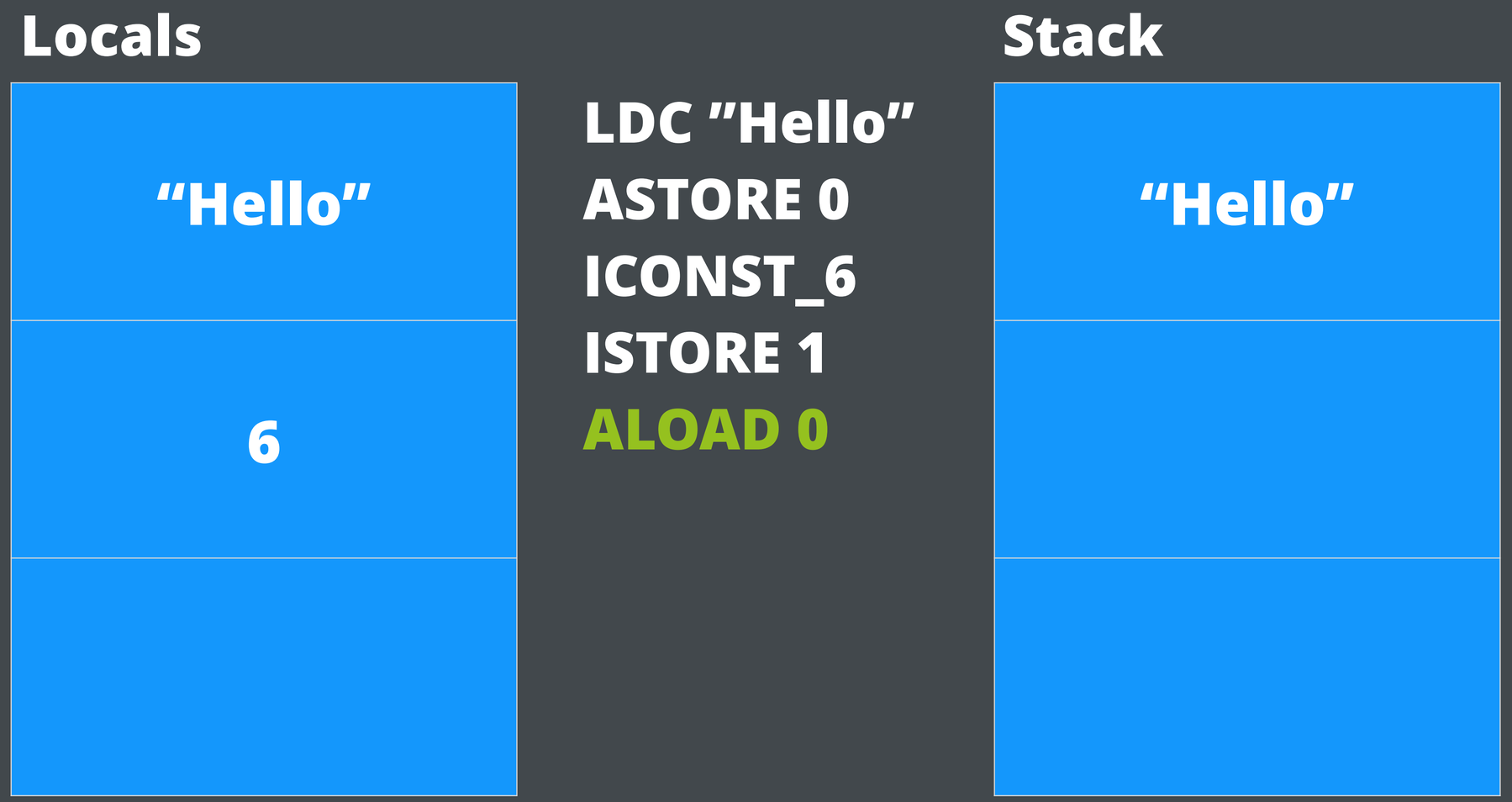
Locals


Locals
“Hello”
LDC ”Hello”
ASTORE 0
ICONST_6
ISTORE 1
ALOAD 0
Stack
6

Locals


Locals
“Hello”
6
LDC ”Hello”
ASTORE 0
ICONST_6
ISTORE 1
ALOAD 0
Stack

Locals


Locals
“Hello”
6
LDC ”Hello”
ASTORE 0
ICONST_6
ISTORE 1
ALOAD 0
Stack
“Hello”

Javassist

Reflection recap

- Introspection of loaded classes
- List class members (fields, methods, constructors)
- List member type, parameters, attributes

Introspection

- **.class** files are bytecode
- **ClassLoader** reads bytecode to **load classes**
- **Bytecode libs** can parse class bytecode directly
- Introspection of unloaded classes!

Javassist vs. Reflection API

- **javassist.ClassPool** - java.lang.ClassLoader
- **javassist.CtClass** - java.lang.Class
- **javassist.CtMethod** - java.lang.reflect.Method
- **javassist.CtConstructor** - java.lang.reflect.Constructor
- **javassist.CtField** - java.lang.reflect.Field

ClassPool

- Container for **CtClass** objects
- Comparable to **java.lang.ClassLoader** for loaded classes
- Contains methods to set up class path, find existing classes, or create new ones

ClassPool

```
ClassPool cp =
  new ClassPool();          // root ClassPool
  new ClassPool(true);      // with system appended
  new ClassPool(parent);    // with parent
  ClassPool.getDefault(); // global singleton
```

ClassPool appendClassPath

```
ClassPath classPath =
  new ClassClassPath(this.getClass());
cp.appendClassPath(classPath);
```

ClassPool appendClassPath

```
byte[] classBytes = new byte[] {
    (byte) 0xCA,
    (byte) 0xFE,
    (byte) 0xBA,
    (byte) 0xBE,
    // ...
};
ClassPath classPath = new ByteArrayClassPath(
    "MySpecialClass",
    classBytes);
cp.appendClassPath(classPath);
```

CtClass

- Comparable to **java.lang.Class**
- Introspective methods to **get declared members**
- Modification methods to **add/remove members**

CtClass

```
private CtClass getClass(ClassPool cp)
    throws NotFoundException {

  CtClass clazz = cp.get("java/lang/String");
  return clazz;
}
```

CtClass

```
private CtClass makeClass(ClassPool cp) {
   CtClass clazz = cp.makeClass("MyClass");
   return clazz;
}
```

CtClass members

```
String name = clazz.getName();
String simpleName = clazz.getSimpleName();
String packageName = clazz.getPackageName();

CtClass[] interfaces = clazz.getInterfaces();
CtClass superclass = clazz.getSuperclass();

int modifiers = clazz.getModifiers();
boolean isPublic = Modifier.isPublic(modifiers);
```

CtField

- Comparable to **java.lang.reflect.Field**
- Access to **access and modify field type, name, etc**

CtField get

```
try {
  CtField field = clazz.getDeclaredField("myField");
  CtField[] fields = clazz.getDeclaredFields();
  // ...
} catch (NotFoundException e) {
  // ...
}
```

CtField add

```
private void addField(CtClass clazz)
    throws CannotCompileException {

  CtField field = CtField.make(
    "private int count;",
    clazz);

  clazz.addField(field);
}
```

CtField members

```
String name = field.getName();
CtClass type = field.getType();

int modifiers = field.getModifiers();
boolean isFinal = Modifier.isFinal(modifiers);
```

CtConstructor

- Comparable to **java.lang.reflect.Constructor**
- Ability to **inspect and modify** **constructor attributes, body, etc**

CtConstructor get

```
try {
  CtConstructor constructors =
    clazz.getDeclaredConstructor(new CtClass[]{...});
  CtConstructor[] constructor =
    clazz.getDeclaredConstructors();
  // ...
} catch (NotFoundException e) {
  // ...
}
```

CtConstructor add

```
private void addConstructor(CtClass clazz)
    throws CannotCompileException {

  CtConstructor constructor = CtNewConstructor.make(
    "public MyClass() {" +
      "System.out.println(\"Constructing!\");" +
    "}", clazz);
  clazz.addConstructor(constructor);
}
```

CtConstructor add #2

```
private void addConstructor2(CtClass clazz)
    throws CannotCompileException {

  CtConstructor constructor = CtNewConstructor.make(
    new CtClass[]{}, // Parameter types
    new CtClass[]{}, // Exception types
    "System.out.println(\"Constructing!\");",
    clazz);
  clazz.addConstructor(constructor);
}
```

CtConstructor members

```
CtClass[] parameterTypes = constr.getParameterTypes();
CtClass[] exceptionTypes = constr.getExceptionTypes();

int modifiers = constr.getModifiers();
boolean isPrivate = Modifier.isPrivate(modifiers);
```

CtMethod

- Comparable to **java.lang.reflect.Method**
- Ability to **inspect and modify** **method attributes, body, etc**

CtMethod get

```
try {
  CtMethod method1 =
    clazz.getDeclaredMethod("sayHello");
  CtMethod method2 =
    clazz.getDeclaredMethod("sayHello", new CtClass[]{...});
  CtMethod[] methods1 = clazz.getDeclaredMethods();
  CtMethod[] methods2 =
    clazz.getDeclaredMethods("sayHello");
  // ...
} catch (NotFoundException e) {
  // ...
}
```

CtMethod add

```
try {
  CtMethod method = CtNewMethod.make(
    "public String sayHello() {" +
      "return \"Hello world\";" +
    "}", clazz);
  clazz.addMethod(method);
  // ...
}
catch (CannotCompileException e) {
  // ...
}
```

CtMethod add #2

```
try {
  CtClass jls = ClassPool.getDefault().get("java/lang/String");

  CtNewMethod.make(
    jls,                  // return type
    "sayHello",           // method name
    new CtClass[]{}, // parameter types
    new CtClass[]{}, // exception types
    "return \"Hello world\";", // body
    clazz);
}
catch (NotFoundException | CannotCompileException e) {
  // ...
}
```

CtMethod members

```
String name = method.getName();
CtClass returnType = method.getReturnType();
CtClass[] parameterTypes = method.getParameterTypes();
CtClass[] exceptionTypes = method.getExceptionTypes();

int modifiers = method.getModifiers();
boolean isAbstract = Modifier.isAbstract(modifiers);
```

Modifying methods

```
try {
  CtMethod method = clazz.getDeclaredMethod("sayHello");
  method.setBody("return \"bytecode\";");
  method.insertBefore("System.out.println(\"Before!\");");
  method.insertAfter("System.out.println(\"After!\");");
}
catch (NotFoundException | CannotCompileException e) {
  // ...
}
```

Substitution variables

- this and Arguments: **$0**, **$1**, **$2** ...
- All arguments: **$$**
- Result type: **$r**
- Resulting value: **$_**
- Expression being evaluated: **$proceed**
- See Javassist tutorial for full list: http://www.csg.ci.i.u-tokyo.ac.jp/~chiba/javassist/tutorial/tutorial2.html

Using substitution variables

```
private int add(int x, int y) {
   return x+y;
}

private int subtract(int x, int y) {
   return x-y;
}
```

Using substitution variables

```
try {
  CtMethod method = clazz.getDeclaredMethod("subtract");
  method.setBody("return add($$);");
  // ...
  method.insertBefore("if ($1 < 0) $1 = 0;");
  // ...
  method.insertAfter("if ($_ < 0) $_ = 0;");
}
catch (NotFoundException | CannotCompileException e) {
  // ...
}
```

Modifiers

```
int modifiers = method.getModifiers();
Modifier.isFinal(modifiers);

Modifier.isPublic(modifiers);
Modifier.isPrivate(modifiers);
Modifier.isProtected(modifiers);

Modifier.isStatic(modifiers);
Modifier.isAbstract(modifiers);

Modifier.isVolatile(modifiers);
Modifier.isSynchronized(modifiers);

Modifier.isNative(modifiers);
Modifier.isStrict(modifiers);
Modifier.isInterface(modifiers);
Modifier.isTransient(modifiers);
```

Example creating class

```
public class Main {
  public static void main(String[] args) throws Exception {
    ClassPool cp = new ClassPool(true);
    CtClass myClass = cp.makeClass("MyClass");
    myClass.addInterface(cp.get("java.lang.Runnable"));
    myClass.addMethod(CtNewMethod.make(
      "public void run() { " +
        "System.out.println(\"Hello world!\");" +
      "}", myClass));
    Runnable runnable = (Runnable) myClass.toClass().newInstance();
    runnable.run();
  }
}
```

Homework #12

- Generate **dynamic proxy** for a class using **Javassist**
- https://github.com/JavaFundamentalsZT/jf-hw-javassist

Questions?

jf@zeroturnaround.com