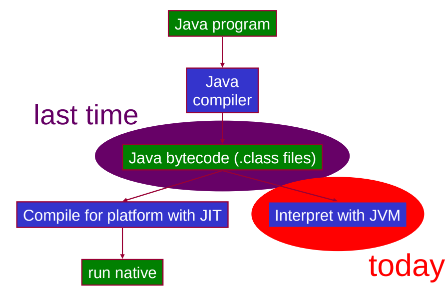
Under the Hood: The Java Virtual Machine Part II



A Whirlwind Tour

- Class loading and initialization
- Object initialization
- Method dispatch
- Exception handling
- Java security model
  - Bytecode verification
  - Stack inspection

Class Loading

Java class loading is *lazy*

- A class is loaded and initialized when it (or a subclass) is first accessed
- Classname must match filename so class loader can find it
- Superclasses are loaded and initialized before subclasses
- Loading = reading in class file, verifying bytecode, integrating into the JVM

Class Initialization

- Prepare static fields with default values
  - 0 for primitive types
  - **`null`** for reference types
- Run static initializer **`<clinit>`**
  - performs programmer-defined initializations
  - only time **`<clinit>`** is ever run
  - only the JVM can call it

Class Initialization

```
class Instructor {
  static Instructor Dexter = new Instructor();
  static Instructor Rich = new Instructor();
  static Instructor Dave = new Instructor();
  static Hashtable h = new Hashtable();
  static {
    h.put(Dexter,"Java");
    h.put(Rich,"Data structures");
    h.put(Dave,"GUI statics and dynamics");
  }
  ...
}
```

Compiled to **`Instructor.<clinit>`**

Initialization Dependencies

```
class A {
  static int a = B.b + 1; //code in A.<clinit>
}

class B {
  static int b = 42; //code in B.<clinit>
}
```

Initialization of **A** will be suspended while **B** is loaded and initialized

Initialization Dependencies

```
class A {
  static int a = B.b + 1; //code in A.<clinit>
}

class B {
  static int b = A.a + 1; //code in B.<clinit>
}
```

Q) Is this legal Java? If so, does it halt?

Initialization Dependencies

```
class A {
  static int a = B.b + 1; //code in A.<clinit>
}

class B {
  static int b = A.a + 1; //code in B.<clinit>
}
```

Q) Is this legal Java? If so, does it halt?

A) yes and yes

Initialization Dependencies

```
class A {
  static int a = B.b + 1; //code in A.<clinit>
}

class B {
  static int b = A.a + 1; //code in B.<clinit>
}
```

Q) So what are the values of `A.a` and `B.b`?

Initialization Dependencies

```
class A {
  static int a = B.b + 1; //code in A.<clinit>
}

class B {
  static int b = A.a + 1; //code in B.<clinit>
}
```

Q) So what are the values of `A.a` and `B.b`?

A) `A.a` = 1 `B.b` = 2

Initialization Dependencies

```
class A {
  static int a = B.b + 1; //code in A.<clinit>
}

class B {
  static int b = A.a + 1; //code in B.<clinit>
}
```

Q) So what are the values of `A.a` and `B.b`?

A) `A.a` = ~~1~~ 2 `B.b` = ~~2~~ 1

Object Initialization

- Object creation initiated by **new** (sometimes implicitly, e.g. by +)
- JVM allocates heap space for object – room for all instance (non-static) fields of the class, including inherited fields, dynamic type info
- Instance fields prepared with default values
  - 0 for primitive types
  - `null` for reference types

Object Initialization

- Call to object initializer `<init>(...)` explicit in the compiled code
  - `<init>` compiled from constructor
  - if none provided, use default `<init>()`
  - first operation of `<init>` must be a call to the corresponding `<init>` of superclass
  - either done explicitly by the programmer using `super(...)` or implicitly by the compiler

Object Initialization

```
class A {
  String name;
  A(String s) {
    name = s;
  }
}
```

```
<init>(java.lang.String)V
0: aload_0
1: invokespecial java.lang.Object.<init>()V
4: aload_0
5: aload_1
6: putfield A.name Ljava/lang/String;
9: return
```

Instance Method Dispatch

`x.foo()`

- compiles to **`invokevirtual`**
- Every loaded class knows its superclass
  - name of superclass is in the constant pool
  - like a parent pointer in the class hierarchy
- bytecode evaluates arguments of **`x.foo()`**, pushes them on the stack
- Object **`x`** is always the first argument

Instance Method Dispatch

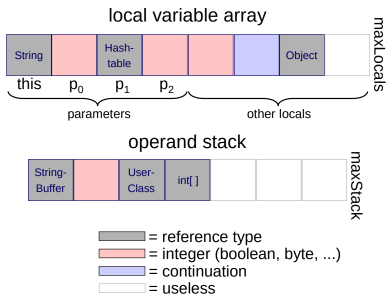
`invokevirtual foo ()V`

- Name and type of **`foo()`** are arguments to **`invokevirtual`** (indices into constant pool)
- JVM retrieves them from constant pool
- Gets the dynamic (runtime) type of **`x`**
- Follows parent pointers until finds **`foo()V`** in one of those classes – gets bytecode from code attribute

Instance Method Dispatch

- Creates a new *stack frame* on runtime stack around arguments already there
- Allocates space in stack frame for locals and operand stack
- Prepares locals, empty stack
- Starts executing bytecode of the method
- When returns, pops stack frame, resumes in calling method after the **`invokevirtual`** instruction

Stack Frame of a Method



Instance Method Dispatch

```
byte[] data;
void getData() {
   String x = "Hello world";
   byte[] data = x.getBytes();
}
```

```
Code(max_stack = 2, max_locals = 2, code_length = 12)
0: ldc "Hello world"
2: astore_1
3: aload_0 //object of which getData is a method
4: aload_1
5: invokevirtual java.lang.String.getBytes ()[B
8: putfield A.data [B
11: return
```

Exception Handling

- Each method has an *exception handler table* (possibly empty)
- Compiled from **`try/catch/finally`**
- An exception handler is just a designated block of code
- When an exception is thrown, JVM searches the exception table for an appropriate handler that is in effect
- **`finally`** clause is executed last

Exception Handling

- Finds an exception handler → empties stack, pushes exception object, executes handler
- No handler → pops runtime stack, returns exceptionally to calling routine
- **`finally`** clause is always executed, no matter what

Exception Table Entry

`startRange`	start of range handler is in effect
`endRange`	end of range handler is in effect
`handlerEntry`	entry point of exception handler
`catchType`	exception handled

- **`startRange`** → **`endRange`** give interval of instructions in which handler is in effect
- **`catchType`** is any subclass of **`Throwable`** (which is a superclass of **`Exception`**) -- any subclass of **`catchType`** can be handled by this handler

Example

```
Integer x = null;
Object y = new Object();

try {
    x = (Integer)y;
    System.out.println(x.intValue());
} catch (ClassCastException e) {
    System.out.println("y was not an Integer");
} catch (NullPointerException e) {
    System.out.println("y was null");
} finally {
    System.out.println("finally!");
}
```

```
0: aconst_null
1: astore_1
2: new java.lang.Object
5: dup
6: invokespecial java.lang.Object.<init> ()V
9: astore_2
10: aload_2
11: checkcast java.lang.Integer
14: astore_1
15: getstatic java.lang.System.out Ljava/io/PrintStream;
18: aload_1
19: invokevirtual java.lang.Integer.intValue ()I
22: invokevirtual java.io.PrintStream.println (I)V
25: getstatic java.lang.System.out Ljava/io/PrintStream;
28: ldc "finally!"
30: invokevirtual java.io.PrintStream.println (Ljava/lang/String;)V
33: goto #89
36: astore_3
37: getstatic java.lang.System.out Ljava/io/P
40: ldc "y was not an Integer"
42: invokevirtual java.io.PrintStream.printl
45: getstatic java.lang.System.out Ljava/io/P
48: ldc "finally!"
50: invokevirtual java.io.PrintStream.printl
53: goto #89
56: astore_3
57: getstatic java.lang.System.out Ljava/io/P
60: ldc "y was null"
62: invokevirtual java.io.PrintStream.printl
65: getstatic java.lang.System.out Ljava/io/P
68: ldc "finally!"
70: invokevirtual java.io.PrintStream.printl
73: goto #89
76: astore 4
78: getstatic java.lang.System.out Ljava/io/P
81: ldc "finally!"
83: invokevirtual java.io.PrintStream.println (Ljava/lang/String;)V
86: aload 4
88: athrow
89: return
```

From	To	Handler	Type
10	25	36	java.lang.ClassCastException
10	25	56	java.lang.NullPointerException
10	25	76	<Any exception>
36	45	76	<Any exception>
56	65	76	<Any exception>
76	78	76	<Any exception>

```
Integer x = null;
Object y = new Object();

try {
   x = (Integer)y;
   System.out.println(x.intValue());
} catch (ClassCastException e) {
   System.out.println("y was not an Integer");
} catch (NullPointerException e) {
   System.out.println("y was null");
} finally {
   System.out.println("finally!");
}
```


```
0: aconst_null
1: astore_1
2: new java.lang.Object
5: dup
6: invokespecial java.lang.Object.<init> ()V
9: astore_2
10: aload_2
11: checkcast java.lang.Integer
14: astore_1
15: getstatic java.lang.System.out Ljava/io/PrintStream;
18: aload_1
19: invokevirtual java.lang.Integer.intValue ()I
22: invokevirtual java.io.PrintStream.println (I)V
25: getstatic java.lang.System.out Ljava/io/PrintStream;
28: ldc "finally!"
30: invokevirtual java.io.PrintStream.println (Ljava/lang/String;)V
33: goto #89
36: astore_3
37: getstatic java.lang.System.out Ljava/io/P
40: ldc "y was not an Integer"
42: invokevirtual java.io.PrintStream.printl
45: getstatic java.lang.System.out Ljava/io/P
48: ldc "finally!"
50: invokevirtual java.io.PrintStream.printl
53: goto #89
56: astore_3
57: getstatic java.lang.System.out Ljava/io/P
60: ldc "y was null"
62: invokevirtual java.io.PrintStream.printl
65: getstatic java.lang.System.out Ljava/io/P
68: ldc "finally!"
70: invokevirtual java.io.PrintStream.printl
73: goto #89
76: astore 4
78: getstatic java.lang.System.out Ljava/io/P
81: ldc "finally!"
83: invokevirtual java.io.PrintStream.println (Ljava/lang/String;)V
86: aload 4
88: athrow
89: return
```

From	To	Handler	Type
10	25	36	java.lang.ClassCastException
10	25	56	java.lang.NullPointerException
10	25	76	<Any exception>
36	45	76	<Any exception>
56	65	76	<Any exception>
76	78	76	<Any exception>

```
Integer x = null;
Object y = new Object();

try {
   x = (Integer)y;
   System.out.println(x.intValue());
} catch (ClassCastException e) {
   System.out.println("y was not an Integer");
} catch (NullPointerException e) {
   System.out.println("y was null");
} finally {
   System.out.println("finally!");
}
```



```
0: aconst_null
1: astore_1
2: new java.lang.Object
5: dup
6: invokespecial java.lang.Object.<init> ()V
9: astore_2
10: aload_2
11: checkcast java.lang.Integer
14: astore_1
15: getstatic java.lang.System.out Ljava/io/PrintStream;
18: aload_1
19: invokevirtual java.lang.Integer.intValue ()I
22: invokevirtual java.io.PrintStream.println (I)V
25: getstatic java.lang.System.out Ljava/io/PrintStream;
28: ldc "finally!"
30: invokevirtual java.io.PrintStream.println (Ljava/lang/String;)V
33: goto #89
36: astore_3
37: getstatic java.lang.System.out Ljava/io/P
40: ldc "y was not an Integer"
42: invokevirtual java.io.PrintStream.printl
45: getstatic java.lang.System.out Ljava/io/P
48: ldc "finally!"
50: invokevirtual java.io.PrintStream.printl
53: goto #89
56: astore_3
57: getstatic java.lang.System.out Ljava/io/P
60: ldc "y was null"
62: invokevirtual java.io.PrintStream.printl
65: getstatic java.lang.System.out Ljava/io/P
68: ldc "finally!"
70: invokevirtual java.io.PrintStream.printl
73: goto #89
76: astore 4
78: getstatic java.lang.System.out Ljava/io/P
81: ldc "finally!"
83: invokevirtual java.io.PrintStream.println (Ljava/lang/String;)V
86: aload 4
88: athrow
89: return
```

From	To	Handler	Type
10	25	36	java.lang.ClassCastException
10	25	56	java.lang.NullPointerException
10	25	76	<Any exception>
36	45	76	<Any exception>
56	65	76	<Any exception>
76	78	76	<Any exception>

```
Integer x = null;
Object y = new Object();

try {
   x = (Integer)y;
   System.out.println(x.intValue());
} catch (ClassCastException e) {
   System.out.println("y was not an Integer");
} catch (NullPointerException e) {
   System.out.println("y was null");
} finally {
   System.out.println("finally!");
}
```



```
0: aconst_null
1: astore_1
2: new java.lang.Object
5: dup
6: invokespecial java.lang.Object.<init> ()V
9: astore_2
10: aload_2
11: checkcast java.lang.Integer
14: astore_1
15: getstatic java.lang.System.out Ljava/io/PrintStream;
18: aload_1
19: invokevirtual java.lang.Integer.intValue ()I
22: invokevirtual java.io.PrintStream.println (I)V
25: getstatic java.lang.System.out Ljava/io/PrintStream;
28: ldc "finally!"
30: invokevirtual java.io.PrintStream.println (Ljava/lang/String;)V
33: goto #89
36: astore_3
37: getstatic java.lang.System.out Ljava/io/P
40: ldc "y was not an Integer"
42: invokevirtual java.io.PrintStream.printl
45: getstatic java.lang.System.out Ljava/io/P
48: ldc "finally!"
50: invokevirtual java.io.PrintStream.printl
53: goto #89
56: astore_3
57: getstatic java.lang.System.out Ljava/io/P
60: ldc "y was null"
62: invokevirtual java.io.PrintStream.printl
65: getstatic java.lang.System.out Ljava/io/P
68: ldc "finally!"
70: invokevirtual java.io.PrintStream.printl
73: goto #89
76: astore 4
78: getstatic java.lang.System.out Ljava/io/P
81: ldc "finally!"
83: invokevirtual java.io.PrintStream.println (Ljava/lang/String;)V
86: aload 4
88: athrow
89: return
```

From	To	Handler	Type
10	25	36	java.lang.ClassCastException
10	25	56	java.lang.NullPointerException
10	25	76	<Any exception>
36	45	76	<Any exception>
56	65	76	<Any exception>
76	78	76	<Any exception>

```
Integer x = null;
Object y = new Object();

try {
   x = (Integer)y;
   System.out.println(x.intValue());
} catch (ClassCastException e) {
   System.out.println("y was not an Integer");
} catch (NullPointerException e) {
   System.out.println("y was null");
} finally {
   System.out.println("finally!");
}
```



```
0: aconst_null
1: astore_1
2: new java.lang.Object
5: dup
6: invokespecial java.lang.Object.<init> ()V
9: astore_2
10: aload_2
11: checkcast java.lang.Integer
14: astore_1
15: getstatic java.lang.System.out Ljava/io/PrintStream;
18: aload_1
19: invokevirtual java.lang.Integer.intValue ()I
22: invokevirtual java.io.PrintStream.println (I)V
25: getstatic java.lang.System.out Ljava/io/PrintStream;
28: ldc "finally!"
30: invokevirtual java.io.PrintStream.println (Ljava/lang/String;)V
33: goto #89
36: astore_3
37: getstatic java.lang.System.out Ljava/io/P
40: ldc "y was not an Integer"
42: invokevirtual java.io.PrintStream.printl
45: getstatic java.lang.System.out Ljava/io/P
48: ldc "finally!"
50: invokevirtual java.io.PrintStream.printl
53: goto #89
56: astore_3
57: getstatic java.lang.System.out Ljava/io/P
60: ldc "y was null"
62: invokevirtual java.io.PrintStream.printl
65: getstatic java.lang.System.out Ljava/io/P
68: ldc "finally!"
70: invokevirtual java.io.PrintStream.printl
73: goto #89
76: astore 4
78: getstatic java.lang.System.out Ljava/io/P
81: ldc "finally!"
83: invokevirtual java.io.PrintStream.println (Ljava/lang/String;)V
86: aload 4
88: athrow
89: return
```

From	To	Handler	Type
10	25	36	java.lang.ClassCastException
10	25	56	java.lang.NullPointerException
10	25	76	<Any exception>
36	45	76	<Any exception>
56	65	76	<Any exception>
76	78	76	<Any exception>

```
Integer x = null;
Object y = new Object();

try {
   x = (Integer)y;
   System.out.println(x.intValue());
} catch (ClassCastException e) {
   System.out.println("y was not an Integer");
} catch (NullPointerException e) {
   System.out.println("y was null");
} finally {
   System.out.println("finally!");
}
```



```
0: aconst_null
1: astore_1
2: new java.lang.Object
5: dup
6: invokespecial java.lang.Object.<init> ()V
9: astore_2
10: aload_2
11: checkcast java.lang.Integer
14: astore_1
15: getstatic java.lang.System.out Ljava/io/PrintStream;
18: aload_1
19: invokevirtual java.lang.Integer.intValue ()I
22: invokevirtual java.io.PrintStream.println (I)V
25: getstatic java.lang.System.out Ljava/io/PrintStream;
28: ldc "finally!"
30: invokevirtual java.io.PrintStream.println (Ljava/lang/String;)V
33: goto #89
36: astore_3
37: getstatic java.lang.System.out Ljava/io/P
40: ldc "y was not an Integer"
42: invokevirtual java.io.PrintStream.printl
45: getstatic java.lang.System.out Ljava/io/P
48: ldc "finally!"
50: invokevirtual java.io.PrintStream.printl
53: goto #89
56: astore_3
57: getstatic java.lang.System.out Ljava/io/P
60: ldc "y was null"
62: invokevirtual java.io.PrintStream.printl
65: getstatic java.lang.System.out Ljava/io/P
68: ldc "finally!"
70: invokevirtual java.io.PrintStream.printl
73: goto #89
76: astore 4
78: getstatic java.lang.System.out Ljava/io/P
81: ldc "finally!"
83: invokevirtual java.io.PrintStream.println (Ljava/lang/String;)V
86: aload 4
88: athrow
89: return
```

From	To	Handler	Type
10	25	36	java.lang.ClassCastException
10	25	56	java.lang.NullPointerException
10	25	76	<Any exception>
36	45	76	<Any exception>
56	65	76	<Any exception>
76	78	76	<Any exception>

```
Integer x = null;
Object y = new Object();

try {
   x = (Integer)y;
   System.out.println(x.intValue());
} catch (ClassCastException e) {
   System.out.println("y was not an Integer");
} catch (NullPointerException e) {
   System.out.println("y was null");
} finally {
   System.out.println("finally!");
}
```

```
0: aconst_null
1: astore_1
2: new java.lang.Object
5: dup
6: invokespecial java.lang.Object.<init> ()V
9: astore_2
10: aload_2
11: checkcast java.lang.Integer
14: astore_1
15: getstatic java.lang.System.out Ljava/io/PrintStream;
18: aload_1
19: invokevirtual java.lang.Integer.intValue ()I
22: invokevirtual java.io.PrintStream.println (I)V
25: getstatic java.lang.System.out Ljava/io/PrintStream;
28: ldc "finally!"
30: invokevirtual java.io.PrintStream.println (Ljava/lang/String;)V
33: goto #89
36: astore_3
37: getstatic java.lang.System.out Ljava/io/P
40: ldc "y was not an Integer"
42: invokevirtual java.io.PrintStream.println
45: getstatic java.lang.System.out Ljava/io/P
48: ldc "finally!"
50: invokevirtual java.io.PrintStream.println
53: goto #89
56: astore_3
57: getstatic java.lang.System.out Ljava/io/P
60: ldc "y was null"
62: invokevirtual java.io.PrintStream.println
65: getstatic java.lang.System.out Ljava/io/P
68: ldc "finally!"
70: invokevirtual java.io.PrintStream.println
73: goto #89
76: astore 4
78: getstatic java.lang.System.out Ljava/io/P
81: ldc "finally!"
83: invokevirtual java.io.PrintStream.println (Ljava/lang/String;)V
86: aload 4
88: athrow
89: return
```

From	To	Handler	Type
10	25	36	java.lang.ClassCastException
10	25	56	java.lang.NullPointerException
10	25	76	<Any exception>
36	45	76	<Any exception>
56	65	76	<Any exception>
76	78	76	<Any exception>

```
Integer x = null;
Object y = new Object();

try {
   x = (Integer)y;
   System.out.println(x.intValue());
} catch (ClassCastException e) {
   System.out.println("y was not an Integer");
} catch (NullPointerException e) {
   System.out.println("y was null");
} finally {
   System.out.println("finally!");
}
```


```
0: aconst_null
1: astore_1
2: new java.lang.Object
5: dup
6: invokespecial java.lang.Object.<init> ()V
9: astore_2
10: aload_2
11: checkcast java.lang.Integer
14: astore_1
15: getstatic java.lang.System.out Ljava/io/PrintStream;
18: aload_1
19: invokevirtual java.lang.Integer.intValue ()I
22: invokevirtual java.io.PrintStream.println (I)V
25: getstatic java.lang.System.out Ljava/io/PrintStream;
28: ldc "finally!"
30: invokevirtual java.io.PrintStream.println (Ljava/lang/String;)V
33: goto #89
36: astore_3
37: getstatic java.lang.System.out Ljava/io/P
40: ldc "y was not an Integer"
42: invokevirtual java.io.PrintStream.println
45: getstatic java.lang.System.out Ljava/io/P
48: ldc "finally!"
50: invokevirtual java.io.PrintStream.println
53: goto #89
56: astore_3
57: getstatic java.lang.System.out Ljava/io/P
60: ldc "y was null"
62: invokevirtual java.io.PrintStream.println
65: getstatic java.lang.System.out Ljava/io/P
68: ldc "finally!"
70: invokevirtual java.io.PrintStream.println
73: goto #89
76: astore 4
78: getstatic java.lang.System.out Ljava/io/P
81: ldc "finally!"
83: invokevirtual java.io.PrintStream.println (Ljava/lang/String;)V
86: aload 4
88: athrow
89: return
```

From	To	Handler	Type
10	25	36	java.lang.ClassCastException
10	25	56	java.lang.NullPointerException
10	25	76	<Any exception>
36	45	76	<Any exception>
56	65	76	<Any exception>
76	78	76	<Any exception>

```
Integer x = null;
Object y = new Object();

try {
   x = (Integer)y;
   System.out.println(x.intValue());
} catch (ClassCastException e) {
   System.out.println("y was not an Integer");
} catch (NullPointerException e) {
   System.out.println("y was null");
} finally {
   System.out.println("finally!");
}
```

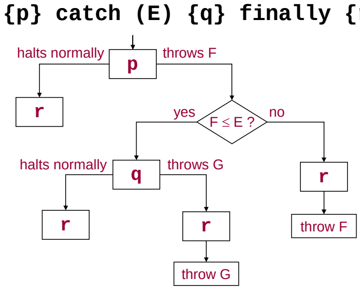

Try/Catch/Finally

`try {p} catch (E) {q} finally {r}`

- **r** is always executed, regardless of whether **p** and/or **q** halt normally or exceptionally
- If **p** throws an exception not caught by the catch clause, or if **q** throws an exception, that exception is *rethrown* upon normal termination of **r**

Try/Catch/Finally

`try {p} catch (E) {q} finally {r}`



Java Security Model

- Bytecode verification
  - Type safety
  - Private/protected/package/final annotations
  - Basis for the entire security model
  - Prevents circumvention of higher-level checks
- Secure class loading
  - Guards against substitution of malicious code for standard system classes
- Stack inspection
  - Mediates access to critical resources

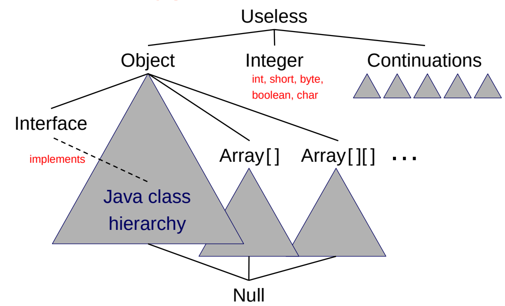
Bytecode Verification

- Performed at load time
- Enforces type safety
  - All operations are well-typed (e.g., may not confuse refs and ints)
  - Array bounds
  - Operand stack overflow, underflow
  - Consistent state over all dataflow paths
- Private/protected/package/final annotations

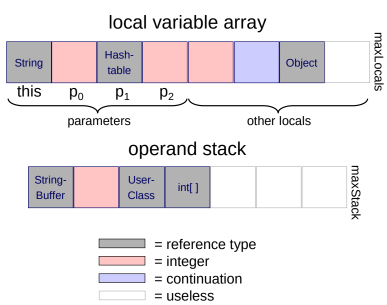
Bytecode Verification

- A form of *dataflow analysis* or *abstract interpretation* performed at load time
- Annotate the program with information about the execution state at each point
- Guarantees that values are used correctly

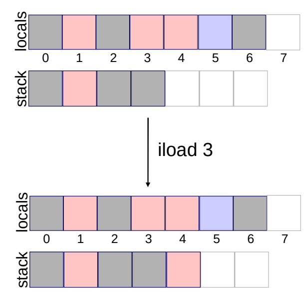
Types in the JVM



Typing of Java Bytecode



Example



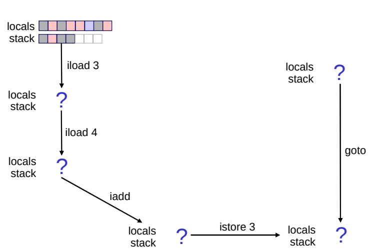
Preconditions for safe execution:

- local 3 is an integer
- stack is not full

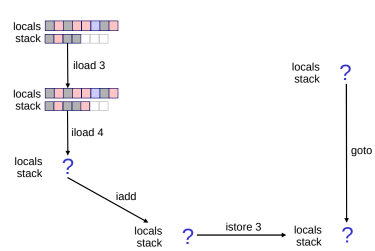
Effect:

- push integer in local 3 on stack

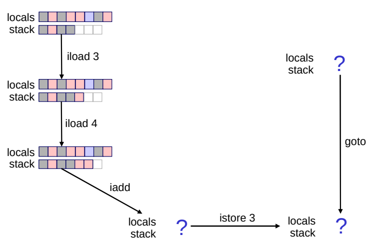
Example



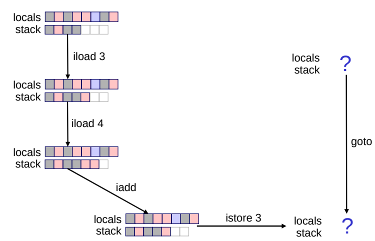
Example



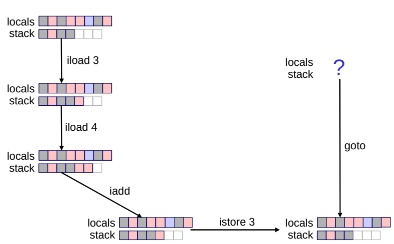
Example



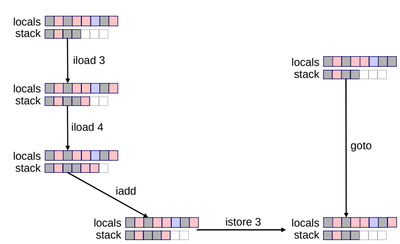
Example



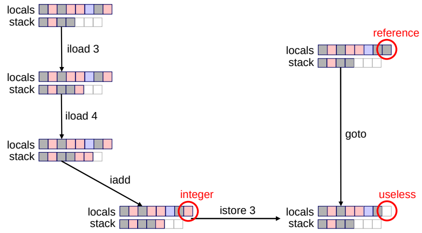
Example



Example

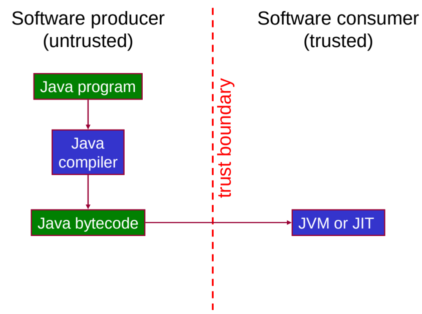


Example



Example

Mobile Code



Mobile Code

Problem: mobile code is not trustworthy!

- We often have *trusted* and *untrusted* code running together in the same virtual machine
  - e.g., applets downloaded off the net and running in our browser
- Do not want untrusted code to perform critical operations (file I/O, net I/O, class loading, security management,...)
- *How do we prevent this?*

Mobile Code

Early approach: *signed applets*

- Not so great
  - everything is either trusted or untrusted, nothing in between
  - a signature can only *verify* an already existing relationship of trust, it cannot *create* trust
- Would like to allow untrusted code to interact with trusted code
  - just monitor its activity somehow

Mobile Code

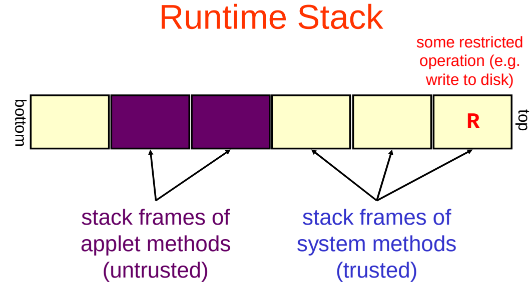
Q) Why not just let trusted (system) code do anything it wants, even in the presence of untrusted code?

Mobile Code

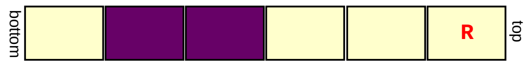
Q) Why not just let trusted (system) code do anything it wants, even in the presence of untrusted code?

A) Because untrusted code calls system code to do stuff (file I/O, etc.) -- System code could be operating on behalf of untrusted code

Runtime Stack

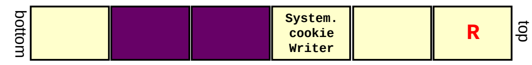


Runtime Stack



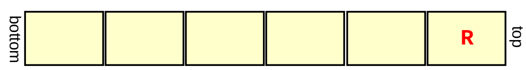
Maybe we want to disallow it
- the malicious applet may be trying to erase our disk
- it's calling system code to do that

Runtime Stack

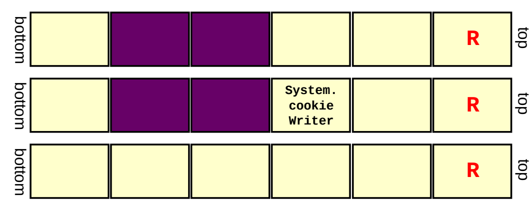


Or, maybe we want to allow it
- it may just want to write a cookie
- it called **`System.cookieWriter`**
- **`System.cookieWriter`** knows it's ok

Runtime Stack



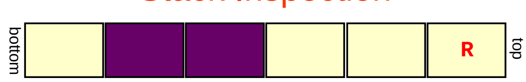
Maybe we want to allow it for another reason
- all running methods are trusted



Q) How do we tell the difference between these scenarios?

A) *Stack inspection!*

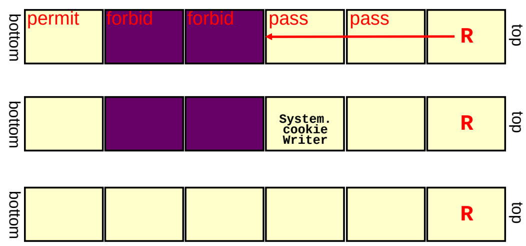
Stack Inspection



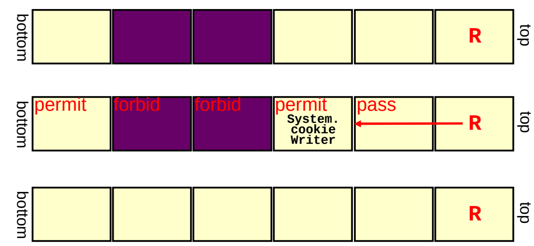
- An invocation of a trusted method, when calling another method, may either:
  - *permit* R on the stack above it
  - *forbid* R on the stack above it
  - *pass* permission from below (be transparent)
- An instantiation of an untrusted method must *forbid* R above it

Stack Inspection

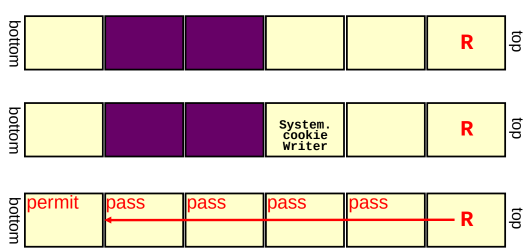
- When about to execute R, look down through the stack until we see either
  - a system method permitting R -- do it
  - a system method forbidding R -- don't do it
  - an untrusted method -- don't do it
- If we get all the way to the bottom, do it (IE, Sun JDK) or don't do it (Netscape)



Case A: R is not executed



Case B: R is executed



Case C: R is executed

Conclusion

Java and the Java Virtual Machine:
Lots of great ideas!