Java Bytecode
What’s inside class files.
MIT AITI
July 2005
What are these .class files?

• You’ve created .java source code files.
• You’ve compiled .java source code into .class files.
• You’ve run your .class files.
• But what’s *are* those .class files?
Typical Software Production

• Source code is **ported** to different platform-specific sources.
• Each port is **compiled** with a platform-specific compiler.
• Each compiler produces platform-specific **machine code** (or **binaries**).
• Binaries execute on a single platform.
PORTING  

Windows Source  
- BSD Port  
- OS X Port  
- Linux Port

COMPILATION  

- BSD Compiler  
- Windows Compiler  
- Linux Compiler

EXECUTION  

- BSD Binary  
- Windows Binary  
- OS X Binary  
- Linux Binary

Machine  
- BSD  
- Windows  
- OS X  
- Linux

Figure by MIT OCW.
Write Once, Run Anywhere

• Java source code is compiled into machine-independent bytecode class files.
• `javac` command compiles `.java` source code into `.class` bytecode.
• Bytecode is interpreted by machine-specific Java Virtual Machines (JVMs).
• Bytecode consists of simple, step-by-step instructions for the JVM.
Java Virtual Machines

- JVM is a computer simulated in a computer.
- JVMs are built into most web browsers.
- `java` command gives `.class` file to JVM.
- JVM interprets the bytecode into instructions that a specific platform can understand.
- Different JVMs for different platforms: Windows, Mac OS X, Linux, cell phones.
Using javap

- `javap` is a program that can give you information on `.class` files.
- If you get a class file without documentation, `javap` can tell you what methods you can call.
- `javap` can show you how `javac` compiles your program into simple instructions.
- Good for high-performance optimizations or if you don’t have access to an Application Programming Interface (API).
javap Output

• Given “Mystery.class”:

> javap Mystery
Compiled from "Mystery.java"
class Mystery extends java.lang.Object{
    Mystery();
    public static int unknown(int,int);
}

• One static method called “unknown” that takes two integers and returns an integer.
javap -c Output

• Using the “-c” flag will disassemble the bytecode:

```java
public static int unknown(int, int);
Code:
0:   iload_0
1:   iload_1
2:   iadd
3:   ireturn
}
```

Loads the first integer argument.
Loads the second integer argument.
Adds the two loaded integer values.
Returns the integer sum.

This method just adds two numbers together and returns the sum.
Example: String Appends

• Disassemble the following method using `javap -c`:

```java
public String append(String a, String b) {
    return a+b;
}
```

• The output is surprisingly complicated…
public static java.lang.String
    stringAdd(java.lang.String, java.lang.String, int);

    Code:
    0: new  #2; //class StringBuffer
    3: dup
    4: invokespecial  #3; //Method
        java/lang/StringBuffer."<init>":()V
    7: aload_0
    8: invokevirtual  #4; //Method
        java/lang/StringBuffer.append:(Ljava/lang/String;)Ljava/lang/StringBuffer;
   11: aload_1
   12: invokevirtual  #4; //Method
        java/lang/StringBuffer.append:(Ljava/lang/String;)Ljava/lang/StringBuffer;
   15: invokevirtual  #5; //Method
        java/lang/StringBuffer.toString:()Ljava/lang/String;
   18: areturn
}
Example: String Appends

• All that bytecode is equivalent to:

```java
StringBuffer temp;
temp = new StringBuffer();
temp.append(a);
temp.append(b);
return temp.toString();
```

• One line of code initializes an object and makes three method calls.
String Append Optimization

• What if we had:
  • String a =""; String b = “hello”;
    for (int i=0; i<n; i++)
      a += b;

• This code performs n initializations and 3n method calls.

• Using javap showed us there’s a class called StringBuffer that has an append() function.
String Append Optimization

• A more efficient way:

```java
StringBuffer aBuff = new StringBuffer();
String b = "hello";
for (int i=0; i<n; i++)
    a.append(b);
String a = aBuff.toString();
```

• Only performs two initializations and n+1 method calls -- three times faster.
Questions

• What advantages are there in running programs on simulated computers, like JVMs?
• What disadvantages might there be?
• Could you compile Java directly into native machine code like C or C++, instead of using a JVM?
• Is it possible to generate Java source code from class files, i.e. decompile?