The Java Class Disassembler

Prepared by Jeff Hunter, Sr. DBA *02-SEP-2002*

Overview

The *javap* command disassembles a class file. It reads the class files specified b the class names on the command line and prints a human-readable version of the API defined by those classes. javap cal also disassemble the specified classes, displaying the Java VM byte code for the mthods they contain. The output from *javap* depends on the options used. If no options are used, *javap* prints out the public fields and methods of the classes passed to it. *javap* prints its output to STDOUT.

Synopsis

```
Javap [options] classnames
```

Options

`-l`	Prints out line and local variable tables, if available in the class file. This option is typically used only when used with the `-c` option. The *javac* compiler does not include local variable information in its class files by default.
`-b`	Ensures backward compatibility with *javap* in JDK 1.1. This option exists or programs that depend on the precise output format of *javap*. This options was introduced in Java 1.2.
`-help`	Prints a usage message and exists.
`-public`	Shows only public classes and members.
`-protected`	Shows only protected and public classes and members.
`-package`	Shows only package, protected, and public classes and members. This is the default.
`-private`	Shows all classes and members.
`-J flag`	Pass flag directly to the runtime system.
`-s`	Prints internal type signatures. Will output the class member declarations using the internal VM type and method signature format, instead of the more readable source-code format.
`-c`	Prints out disassembled code, i.e., the instructions that comprise the Java bytecodes, for each of the methods in the class. This will display the code (Java VM byte code) for each method of each specified class. This option always disassembles all methods regardless of their visibility level.
`-classpath path`	Specifies the path *javap* uses to look up classes. Overrides the default or the CLASSPATH environment variable if it is set. Directories are separated by colons. Thus the general format for path is: `.:<your_path>`

For example:

```
.:/export/home/jhunter/classes:/usr/local/java/classes
```

`-verbose`	Prints stack size, number of locals and args for methods.
`-verify`	Performs a partial verification of the class file. Because this option does not perform many portions of a full verification, its use is not recommended. Instead, `java -verify` should be used to verify class files. This option has been removed in Java 1.2 and later because it does not perform a sufficiently thorough verification.
`-version`	Prints out the *javap* version string.
`-extdirs dirs`	Specifies one or more directories that should be searched for extensions classes. This option is rarely used and was introduced in Java 1.2.

Examples

All of the examples in this section will use the following class declaration:

```
public class testJavap {

  private static void doSubstring() {
    String a = "Alex Michael Hunter";
    System.out.println("Original string: " + a);
    String b = a.substring(5);
    System.out.println("a.Substring(5) : " + b);
    String c = a.substring(5,12);
    System.out.println("a.substring(5,12) : " + c);
    String d = a.substring(13,a.length());
    System.out.println("a.substring(13,a.length()) : " + d);
    System.out.println();
  }

  public static void main(String[] args) {
    doSubstring();
  }

}
```

testJavap.java

- The default output of *javap*:

```
% javap testJavap

   Compiled from testJavap.java
   public class testJavap extends java.lang.Object {
       public testJavap();
       public static void main(java.lang.String[]);
   }
```

- The `verbose` option of *javap*:

```
% javap -verbose testJavap

   Compiled from testJavap.java
```

```
public class testJavap extends java.lang.Object {
    public testJavap();
        /* Stack=1, Locals=1, Args_size=1 */
    public static void main(java.lang.String[]);
        /* Stack=0, Locals=1, Args_size=1 */
}
```

- Output the internal VM type and method signature format of *javap*:

```
% javap -s testJavap

Compiled from testJavap.java
public class testJavap extends java.lang.Object {
    public testJavap();
        /*   ()V   */
    public static void main(java.lang.String[]);
        /*   ([Ljava/lang/String;)V   */
}
```

- Display the byte-code of the class:

```
% javap -c testJavap

Compiled from testJavap.java
public class testJavap extends java.lang.Object {
    public testJavap();
    public static void main(java.lang.String[]);
}

Method testJavap()
   0 aload_0
   1 invokespecial #1 <Method java.lang.Object()>
   4 return

Method void doSubstring()
   0 ldc #2 <String "Alex Michael Hunter">
   2 astore_0
   3 getstatic #3 <Field java.io.PrintStream out>
   6 new #4 <Class java.lang.StringBuffer>
   9 dup
  10 invokespecial #5 <Method java.lang.StringBuffer()>
  13 ldc #6 <String "Original string: ">
  15 invokevirtual #7 <Method java.lang.StringBuffer
append(java.lang.String)>
  18 aload_0
  19 invokevirtual #7 <Method java.lang.StringBuffer
append(java.lang.String)>
  22 invokevirtual #8 <Method java.lang.String toString()>
  25 invokevirtual #9 <Method void println(java.lang.String)>
  28 aload_0
  29 iconst_5
  30 invokevirtual #10 <Method java.lang.String substring(int)>
  33 astore_1
  34 getstatic #3 <Field java.io.PrintStream out>
  37 new #4 <Class java.lang.StringBuffer>
  40 dup
  41 invokespecial #5 <Method java.lang.StringBuffer()>
  44 ldc #11 <String "a.Substring(5) : ">
  46 invokevirtual #7 <Method java.lang.StringBuffer
append(java.lang.String)>
  49 aload_1
  50 invokevirtual #7 <Method java.lang.StringBuffer
append(java.lang.String)>
  53 invokevirtual #8 <Method java.lang.String toString()>
  56 invokevirtual #9 <Method void println(java.lang.String)>
  59 aload_0
  60 iconst_5
  61 bipush 12
  63 invokevirtual #12 <Method java.lang.String substring(int, int)>
  66 astore_2
```

```
  67 getstatic #3 <Field java.io.PrintStream out>
  70 new #4 <Class java.lang.StringBuffer>
  73 dup
  74 invokespecial #5 <Method java.lang.StringBuffer()>
  77 ldc #13 <String "a.substring(5,12) : ">
  79 invokevirtual #7 <Method java.lang.StringBuffer
append(java.lang.String)>
  82 aload_2
  83 invokevirtual #7 <Method java.lang.StringBuffer
append(java.lang.String)>
  86 invokevirtual #8 <Method java.lang.String toString()>
  89 invokevirtual #9 <Method void println(java.lang.String)>
  92 aload_0
  93 bipush 13
  95 aload_0
  96 invokevirtual #14 <Method int length()>
  99 invokevirtual #12 <Method java.lang.String substring(int, int)>
 102 astore_3
103 getstatic #3 <Field java.io.PrintStream out>
 106 new #4 <Class java.lang.StringBuffer>
 109 dup
 110 invokespecial #5 <Method java.lang.StringBuffer()>
 113 ldc #15 <String "a.substring(13,a.length()) : ">
 115 invokevirtual #7 <Method java.lang.StringBuffer
append(java.lang.String)>
 118 aload_3
 119 invokevirtual #7 <Method java.lang.StringBuffer
append(java.lang.String)>
 122 invokevirtual #8 <Method java.lang.String toString()>
 125 invokevirtual #9 <Method void println(java.lang.String)>
 128 getstatic #3 <Field java.io.PrintStream out>
 131 invokevirtual #16 <Method void println()>
 134 return

Method void main(java.lang.String[])
   0 invokestatic #17 <Method void doSubstring()>
   3 return
```