NAME

javap − disassemble one or more class files

SYNOPSIS

`javap` [*options*] *classes*...

options Specifies the command−line options. See **Options for javap**.

classes Specifies one or more classes separated by spaces to be processed for annotations. You can specify a class that can be found in the class path by its file name, URL, or by its fully qualified class name.

Examples:

```
path/to/MyClass.class

jar:file:///path/to/MyJar.jar!/mypkg/MyClass.class

java.lang.Object
```

DESCRIPTION

The **`javap`** command disassembles one or more class files. The output depends on the options used. When no options are used, the **`javap`** command prints the protected and public fields, and methods of the classes passed to it.

The **`javap`** command isn't multirelease JAR aware. Using the class path form of the command results in viewing the base entry in all JAR files, multirelease or not. Using the URL form, you can use the URL form of an argument to specify a specific version of a class to be disassembled.

The **`javap`** command prints its output to **`stdout`**.

Note:

In tools that support **`--`** style options, the GNU−style options can use the equal sign (**`=`**) instead of a white space to separate the name of an option from its value.

OPTIONS FOR JAVAP

`--help`, **`-help`** , **`-h`**, or **`-?`**
Prints a help message for the **`javap`** command.

`-version`
Prints release information.

`-verbose` or **`-v`**
Prints additional information about the selected class.

`-l` Prints line and local variable tables.

`-public`
Shows only public classes and members.

`-protected`
Shows only protected and public classes and members.

`-package`
Shows package/protected/public classes and members (default).

`-private` or **`-p`**
Shows all classes and members.

`-c` Prints disassembled code, for example, the instructions that comprise the Java bytecodes, for each of the methods in the class.

`-s` Prints internal type signatures.

`-sysinfo`
Shows system information (path, size, date, SHA−256 hash) of the class being processed.

-constants
Shows **static final** constants.

--module *module* or **-m** *module*
Specifies the module containing classes to be disassembled.

--module-path *path*
Specifies where to find application modules.

--system *jdk*
Specifies where to find system modules.

--class-path *path*, **-classpath** *path*, or **-cp** *path*
Specifies the path that the **javap** command uses to find user class files. It overrides the default or the **CLASSPATH** environment variable when it's set.

-bootclasspath *path*
Overrides the location of bootstrap class files.

--multi-release *version*
Specifies the version to select in multi−release JAR files.

-J*option*
Passes the specified option to the JVM. For example:

```
javap −J−version

javap −J−Djava.security.manager −J−Djava.security.policy=MyPolicy MyC
```

See *Overview of Java Options* in **java**.

JAVAP EXAMPLE

Compile the following **HelloWorldFrame** class:

```
import java.awt.Graphics;

import javax.swing.JFrame;
import javax.swing.JPanel;

public class HelloWorldFrame extends JFrame {

   String message = "Hello World!";

   public HelloWorldFrame(){
        setContentPane(new JPanel(){
            @Override
            protected void paintComponent(Graphics g) {
                g.drawString(message, 15, 30);
            }
        });
        setSize(100, 100);
    }
    public static void main(String[] args) {
        HelloWorldFrame frame = new HelloWorldFrame();
        frame.setVisible(true);

    }

}
```

The output from the **javap HelloWorldFrame.class** command yields the following:

```
Compiled from "HelloWorldFrame.java"
public class HelloWorldFrame extends javax.swing.JFrame {
  java.lang.String message;
  public HelloWorldFrame();
  public static void main(java.lang.String[]);
}
```

The output from the **`javap -c HelloWorldFrame.class`** command yields the following:

```
Compiled from "HelloWorldFrame.java"
public class HelloWorldFrame extends javax.swing.JFrame {
  java.lang.String message;

  public HelloWorldFrame();
    Code:
       0: aload_0
       1: invokespecial #1        // Method javax/swing/JFrame."<init>":()V
       4: aload_0
       5: ldc           #2        // String Hello World!
       7: putfield      #3        // Field message:Ljava/lang/String;
      10: aload_0
      11: new           #4        // class HelloWorldFrame$1
      14: dup
      15: aload_0
      16: invokespecial #5        // Method HelloWorldFrame$1."<init>":(LHe
      19: invokevirtual #6        // Method setContentPane:(Ljava/awt/Conta
      22: aload_0
      23: bipush        100
      25: bipush        100
      27: invokevirtual #7        // Method setSize:(II)V
      30: return

  public static void main(java.lang.String[]);
    Code:
       0: new           #8        // class HelloWorldFrame
       3: dup
       4: invokespecial #9        // Method "<init>":()V
       7: astore_1
       8: aload_1
       9: iconst_1
      10: invokevirtual #10       // Method setVisible:(Z)V
      13: return
}
```