java – The Java Interpreter

java interprets (executes) Java bytecodes.

SYNOPSIS

```
java [ options ] classname <args>
java_g [ options ] classname <args>
```

DESCRIPTION

The **java** command executes Java `class` files created by a Java compiler, for instance, **javac**.

The `classname` argument is the name of the class to be executed. `classname` must be fully qualified by including its package in the name, for example:

```
% java java.lang.String
```

Note that any arguments that appear after `classname` on the command line are passed to the `main` method of the class.

java expects the binary representation of the class to be in a file called `classname.class` which is generated by compiling the corresponding source file with **javac**. All Java class files end with the filename extension `.class` which the compiler automatically adds when the class is compiled. *classname* must contain a `main` method defined as follows:

```
class Aclass {
    public static void main(String argv[]){
        . . .
    }
}
```

java executes the `main` method and then exits unless `main` creates one or more threads. If any threads are created by `main` then **java** doesn't exit until the last thread exits.

When you define your own classes you need to specify their location. Use CLASSPATH to do this. CLASSPATH consists of a colon separated list of directories that specifies the path. For example:

```
.:/home/xyz/classes
```

Note that the system always appends the location of the system classes onto the end of the class path unless you use the `-classpath` option to specify a path.

Ordinarily, you compile source files with **javac** then run the program using **java**. However, **java** can be used to compile and run programs when the `-cs` option is used. As each class file is loaded its modification date is compared to the modification date of the class source file. If the source has been modified more recently, it is recompiled and the new class file is loaded. **java** repeats this

procedure until all the classes are correctly compiled and loaded.

The interpreter can determine whether a class is legitimate through the mechanism of verification. Verification ensures prior to their execution that class files do not violate any language constraints.

java_g is a non−optimized version of **java** suitable for use with debuggers like **jdb**.

OPTIONS

−debug
: Allows the Java debugger, **jdb**, to attach itself to this **java** session. When `−debug` is specified on the command line **java** displays a password which must be used when starting the debugging session.

−cs, −checksource
: When a compiled class is loaded, this option causes the modification time of the class bytecode file to be compared to that of the class source file. If the source has been modified more recently, it is recompiled and the new class file is loaded.

−classpath *path*
: Specifies the path **java** 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:

```
.:/home/xyz/classes:/usr/local/java/classes
```

−mx *x*
: Sets the maximum size of the memory allocation pool (the garbage collected heap) to *x*. The default is 16 megabytes of memory. *x* must be greater than or equal to 1000 bytes.

By default, *x* is measured in bytes. You can specify *x* in either kilobytes or megabytes by appending the letter "k" for kilobytes or the letter "m" for megabytes.

−ms *x*
: Sets the startup size of the memory allocation pool (the garbage collected heap) to *x*. The default is 1 megabyte of memory. *x* must be > 1000 bytes.

By default, *x* is measured in bytes. You can specify *x* in either kilobytes or megabytes by appending the letter "k" for kilobytes or the letter "m" for megabytes.

−noasyncgc
: Turns off asynchronous garbage collection. When activated no garbage collection takes place unless it is explicitly called or the program runs out of memory. Normally garbage collection runs as an asynchronous thread in parallel with other threads.

−noclassgc
Turns off garbage collection of Java classes. By default, the Java interpreter reclaims space for unused Java classes during garbage collection.

−prof
Starts the Java Runtime with Java profiling enabled. By default, this puts profile results in the file `./java.prof`. This option works only with **java_g**.

−prof: *file*
Starts the Java Runtime with Java profiling enabled. This form of the flag allows the user to specify a different output file for the profile information. For example, the flag `−prof:myprog.prof` enables profiling and puts the profile results in the file `myprog.prof` rather than in the default file `./java.prof`. This option works only with **java_g**.

−version
Print the build version information.

−help
Print a usage message.

−ss *x*
Each Java thread has two stacks: one for Java code and one for C code. The `−ss` option sets the maximum stack size that can be used by C code in a thread to *x*. Every thread that is spawned during the execution of the program passed to **java** has *x* as its C stack size. The default units for *x* are bytes. The value of *x* must be greater than or equal to 1000 bytes.

You can modify the meaning of *x* by appending either the letter "k" for kilobytes or the letter "m" for megabytes. The default stack size is 128 kilobytes ("−ss 128k").

−oss *x*
Each Java thread has two stacks: one for Java code and one for C code. The `−oss` option sets the maximum stack size that can be used by Java code in a thread to *x*. Every thread that is spawned during the execution of the program passed to **java** has *x* as its Java stack size. The default units for *x* are bytes. The value of *x* must be greater than or equal to 1000 bytes.

You can modify the meaning of *x* by appending either the letter "k" for kilobytes or the letter "m" for megabytes. The default stack size is 400 kilobytes ("−oss 400k").

−t
Prints a trace of the instructions executed (**java_g** only).

−v, −verbose
Causes **java** to print a message to stdout each time a class file is loaded.

−verify
Performs byte−code verification on the class file. Beware however, that **java −verify** does not perform a full verification in all situations. Any code path that is not actually executed by

the interpreter is not verified. Therefore, **java −verify** cannot be relied upon to certify class files unless all code paths in the class file are actually run.

−verifyremote
Runs the verifier on all code that is loaded into the system via a classloader. *verifyremote* is the default for the interpreter.

−noverify
Turns verification off.

−verbosegc
Causes the garbage collector to print out messages whenever it frees memory.

−D*propertyName=newValue*
Redefines a property value. *propertyName* is the name of the property whose value you want to change and *newValue* is the value to change it to. For example, this command line

```
% java -Dawt.button.color=green ...
```

sets the value of the property awt.button.color to "green". **java** accepts any number of `-D` options on the command line.

ENVIRONMENT VARIABLES

CLASSPATH
Used to provide the system a path to user−defined classes. Directories are separated by colons, for example,

```
.:/home/xyz/classes:/usr/local/java/classes
```

SEE ALSO

javac, jdb, javah, javap, javadoc, CLASSPATH