# 1 How Classes are Found

The java command is called the Java launcher because it launches Java applications. When the Java launcher is called, it gathers input from the user and the user's environment (such as the class path and the boot class path), interfaces with the Virtual Machine (VM), and gets it started via some bootstrapping. The Java Virtual Machine (JVM) does the rest of the work.

This chapter covers the following topics:

* How the Java Runtime Finds Classes
* How the javac and javadoc Commands Find Classes
* Class Loading and Security Policies

## How the Java Runtime Finds Classes

The JVM searches for and loads classes in this order:

1. Bootstrap classes, which are classes that comprise the Java platform, including the classes in rt.jar and several other important JAR files.
2. Extension classes, which use the Java Extension mechanism. These classes are bundled as JAR files and located in the extensions directory.
3. User classes are classes that are defined by developers and third parties and that do not take advantage of the extension mechanism. You identify the location of these classes with the -classpath option on the command line (preferred) or with the CLASSPATH environment variable. See Setting the Class Path.

In effect, the three search paths together form a simple class path. This is similar to the flat class path previously used, but the current model has the following improvements:

* It is relatively difficult to accidentally hide or omit the bootstrap classes.
* In general, you only have to specify the location of user classes. Bootstrap classes and extension classes are found automatically.
* The tools classes are now in a separate archive (tools.jar) and can only be used if included in the user class path described in How the Java Runtime Finds Bootstrap Classes.

### How the Java Runtime Finds Bootstrap Classes

Bootstrap classes are the classes that implement Java SE. Bootstrap classes are in the rt.jar file and several other JAR files in the jre/lib directory. These archives are specified by the value of the bootstrap class path that is stored in the sun.boot.class.path system property. This system property is for reference only and should not be directly modified.It is unlikely that you will need to redefine the bootstrap class path. The nonstandard option, -Xbootclasspath, allows you to do so in those rare circumstances in which it is necessary to use a different set of core classes.

Note that the classes that implement the JDK tools are in a separate archive from the bootstrap classes. The tools archive is the JDK /lib/tools.jar file. The development tools add this archive to the user class path when invoking the launcher. However, this augmented user class path is only used to execute the tool. The tools that process source code, the javac command, and the javadoc command, use the original class path, not the augmented version. For more information, see How the javac and javadoc Commands Find Classes.

### How the Java Runtime Finds Extension Classes

Extension classes are classes that extend the Java platform. Every .jar file in the extension directory, jre/lib/ext, is assumed to be an extension and is loaded with the Java Extension Framework. Loose class files in the extension directory are not found. They must be contained in a JAR file or Zip file. There is no option provided for changing the location of the extension directory.

If the jre/lib/ext directory contains multiple JAR files, and those files contain classes with the same name such as in the following example, the class that actually gets loaded is undefined.

smart-extension1\_0.jar contains class smart.extension.Smart

smart-extension1\_1.jar contains class smart.extension.Smart

### How the Java Runtime Finds User Classes

To find user classes, the launcher refers to the user class path, which is a list of directories, JAR files, and Zip files that contain class files.

A class file has a subpath name that reflects the fully-qualified name of the class. For example, if the class com.mypackage.MyClass is stored under myclasses, then myclasses must be in the user class path, and the full path to the class file must be /myclasses/com/mypackage/MyClass.class on Oracle Solaris or in \myclasses\com\mypackage\MyClass.class on Windows.

If the class is stored in an archive named myclasses.jar, then myclasses.jar must be in the user class path, and the class file must be stored in the archive as com/mypackage/MyClass.class on Windows or in com\mypackage\MyClass.class on Oracle Solaris.

The user class path is specified as a string, with a colon (:) to separate the class path entries on Oracle Solaris, and a semicolon (;) to separate the entries on Windows systems. The Java launcher puts the user class path string in the java.class.path system property. The possible sources of this value are:

* The default value, \*.\*, which means that user class files are all the class files in or under the current directory.
* The value of the CLASSPATH environment variable that overrides the default value.
* The value of the -cp or -classpath command-line option that overrides both the default value and the CLASSPATH value.
* The JAR archive specified by the -jar option overrides all other values if it contains a Class-Path entry in its manifest. If this option is used, all user classes must come from the specified archive.

### How the Java Runtime Finds JAR-class-path Classes

A JAR file usually contains a manifest, which is a file that lists the contents of the JAR file. The manifest can define a JAR class path, which further extends the class path, but only while loading classes from that JAR file. Classes accessed by a JAR class path are found in the following order:

1. In general, classes referenced by a JAR class path entry are found as though they are part of the JAR file. The JAR files that appear in the JAR-class-path are searched after any earlier class path entries and before any entries that appear later in the class path.
2. If the JAR class path points to a JAR file that was already searched, for example, an extension or a JAR file that was listed earlier in the class path, then that JAR file is not searched again. This optimization improves efficiency and prevents circular searches. This type of JAR file is searched at the point that it appears, which is earlier in the class path.
3. If a JAR file is installed as an extension, then any JAR class path it defines is ignored. All of the classes required by an extension are presumed to be part of the JDK or to have been installed as extension.

## How the javac and javadoc Commands Find Classes

The javac and javadoc commands use class files in the following two ways:

* To run, the javac and javadoc commands must load various class files.
* To process the source code they operate on, the javac and javadoc commands must obtain information on object types used in the source code.

The class files used to resolve source code references are mostly the same class files used to run the javac and javadoc commands. But there are some important exceptions, as follows:

* Both the javac and javadoc commands often resolve references to classes and interfaces that have nothing to do with the implementation of the javac or javadoc commands. Information on referenced user classes and interfaces might be present in the form of class files, source code files, or both.
* The tools classes in the tools.jar file are only used to run the javac and javadoc commands. The tools classes are not used to resolve source code references unless the tools.jar file is in the user class path.
* A programmer might want to resolve boot class or extension class references with an alternative Java platform implementation. Both the javac and javadoc commands support this with their -bootclasspath and -extdirs options. Use of these options does not modify the set of class files used to run the javac or javadoc commands themselves.

If a referenced class is defined in both a class file and a source file, the javadoc command always uses the source file. The javadoc command never compiles source files. In the same situation the javac command uses class files, but automatically recompiles any class files it determines to be out of date. The rules for automatic recompilation are documented in javac.

By default, the javac and javadoc commands search the user class path for both class files and source code files. If the -sourcepath option is specified, the javac and javadoc commands search for source files only on the specified source file path, while still searching the user class path for class files.

## Class Loading and Security Policies

To be used, a class or interface must be loaded by a class loader. Use of a particular class loader determines a security policy associated with the class loader.

A program can load a class or interface by calling the loadClass method of a class loader object. But usually a program loads a class or interface by referring to it. This invokes an internal class loader, which can apply a security policy to extension and user classes. If the security policy has not been enabled, all classes are trusted. Even if the security policy is enabled, it does not apply to bootstrap classes, which are always trusted.

When enabled, security policy is configured by system and user policy files. The Java platform SDK includes a system policy file that grants trusted status to extension classes and places basic restrictions on user classes.

To enable or configure the security policy, refer to Security Features.

**Note**: Some security programming techniques that worked with earlier releases of Java SE are incompatible with the class loading model of the current release.