NAME

jdeprscan − static analysis tool that scans a jar file (or some other aggregation of class files) for uses of deprecated API elements

SYNOPSIS

`jdeprscan` [*options*] {*dir*|*jar*|*class*}

options See **Options for the jdeprscan Command**

dir|*jar*|*class*
: **`jdeprscan`** command scans each argument for usages of deprecated APIs. The arguments can be a:

- *dir*: Directory
- *jar*: JAR file
- *class*: Class name or class file

The class name should use a dot (`.`) as a separator. For example:

```
java.lang.Thread
```

For nested classes, the dollar sign **`$`** separator character should be used. For example:

```
java.lang.Thread$State
```

A class file can also be named. For example:

```
build/classes/java/lang/Thread$State.class
```

DESCRIPTION

The **`jdeprscan`** tool is a static analysis tool provided by the JDK that scans a JAR file or some other aggregation of class files for uses of deprecated API elements. The deprecated APIs identified by the **`jdeprscan`** tool are only those that are defined by Java SE. Deprecated APIs defined by third−party libraries aren't reported.

To scan a JAR file or a set of class files, you must first ensure that all of the classes that the scanned classes depend upon are present in the class path. Set the class path using the **`−−class−path`** option described in **Options for the jdeprscan Command**. Typically, you would use the same class path as the one that you use when invoking your application.

If the **`jdeprscan`** can't find all the dependent classes, it will generate an error message for each class that's missing. These error messages are typically of the form:

```
error: cannot find class ...
```

If these errors occur, then you must adjust the class path so that it includes all dependent classes.

OPTIONS FOR THE JDEPRSCAN COMMAND

The following options are available:

`−−class−path` *path*
: Provides a search path for resolution of dependent classes.

path can be a search path that consists of one or more directories separated by the system−specific path separator. For example:

- **Linux and OS X:**

```
    −−class−path /some/directory:/another/different/dir
```

Note:

On Windows, use a semicolon (`;`) as the separator instead of a colon (`:`).

- **Windows:**

```
    −−class−path \some\directory;\another\different\dir
```

--for-removal
: Limits scanning or listing to APIs that are deprecated for removal. Can't be used with a release value of 6, 7, or 8.

--full-version
: Prints out the full version string of the tool.

--help or **-h**
: Prints out a full help message.

--list or **-l**
: Prints the set of deprecated APIs. No scanning is done, so no directory, jar, or class arguments should be provided.

--release **6**|**7**|**8**|**9**
: Specifies the Java SE release that provides the set of deprecated APIs for scanning.

--verbose or **-v**
: Enables additional message output during processing.

--version
: Prints out the abbreviated version string of the tool.

EXAMPLE OF JDEPRSCAN OUTPUT

The JAR file for this library will be named something similar to **commons-math3-3.6.1.jar**. To scan this JAR file for the use of deprecated APIs, run the following command:

```
jdeprscan commons-math3-3.6.1.jar
```

This command produces several lines of output. For example, one line of output might be:

```
class org/apache/commons/math3/util/MathUtils uses deprecated method java/l
```

Note:

The class name is specified using the slash−separated binary name as described in JVMS 4.2.1. This is the form used internally in class files.

The deprecated API it uses is a method on the **java.lang.Double** class.

The name of the deprecated method is **<init>**, which is a special name that means that the method is actually a constructor. Another special name is**<clinit>**, which indicates a class static initializer .

Other methods are listed just by their method name. Following the method name is the argument list and return type:

```
(D)V
```

This indicates that it takes just one double value (a primitive) and returns void. The argument and return types can become cryptic. For example, another line of output might be:

```
class org/apache/commons/math3/util/Precision uses deprecated method java/m
```

In this line of output, the deprecated method is on class **java.math.BigDecimal**, and the method is **setScale()**. In this case, the**(II)** means that it tak es two **int** arguments. The**Ljava/math/BigDecimal;** after the parentheses means that it returns a reference to **java.math.BigDecimal**.

JDEPRSCAN ANALYSIS CAN BE VERSION−SPECIFIC

You can use **jdeprscan** relative to the previous three JDK releases. For example, if you are running JDK 9, then you can check against JDK 8, 7, and 6.

As an example, look at this code snippet:

```
public class Deprecations {
   SecurityManager sm = new RMISecurityManager();    // deprecated in 8
   Boolean b2 = new Boolean(true);          // deprecated in 9
}
```

The complete class compiles without warnings in JDK 7.

If you run **jdeprscan** on a system with JDK 9, then you see:

```
$ jdeprscan --class-path classes --release 7 example.Deprecations
(no output)
```

Run **jdeprscan** with a release value of 8:

```
$ jdeprscan --class-path classes --release 8 example.Deprecations
class example/Deprecations uses type java/rmi/RMISecurityManager deprecated
class example/Deprecations uses method in type java/rmi/RMISecurityManager d
```

Run **jdeprscan** on JDK 9:

```
$ jdeprscan --class-path classes example.Deprecations
class example/Deprecations uses type java/rmi/RMISecurityManager deprecated
class example/Deprecations uses method in type java/rmi/RMISecurityManager d
class example/Deprecations uses method java/lang/Boolean <init> (Z)V depreca
```