Applause from Michelé Clarke, Nicholas Pinn, and 7 others

Adrian D. Finlay
@thewiprogrammer. Writer @hackernoon. Code, LOTS of it. Mangos, LOVE THEM! Barbering. Health. Travel. Business. & more! Network w/ me @ adriandavid.me/network
Oct 5, 2017 · 10 min read

Cool Java 9 Features You Might've Missed

Source: http://www.java9countdown.xyz/

While the predominant features of Java SE 9 were the modularity efforts of Project Jigsaw, Java 9 introduced several other nifty features that will enhance the way we approach Java development. Aside from modules, core library updates, and a few language updates, Java SE 9 brought in other features like **jshell** that will help developers be more effective.

By and large, the features discussed below are not an exhaustive list of the rest of the Java 9 features. Rather, they are a few of the features that caught my attention when reading the JDK 9 release notes.

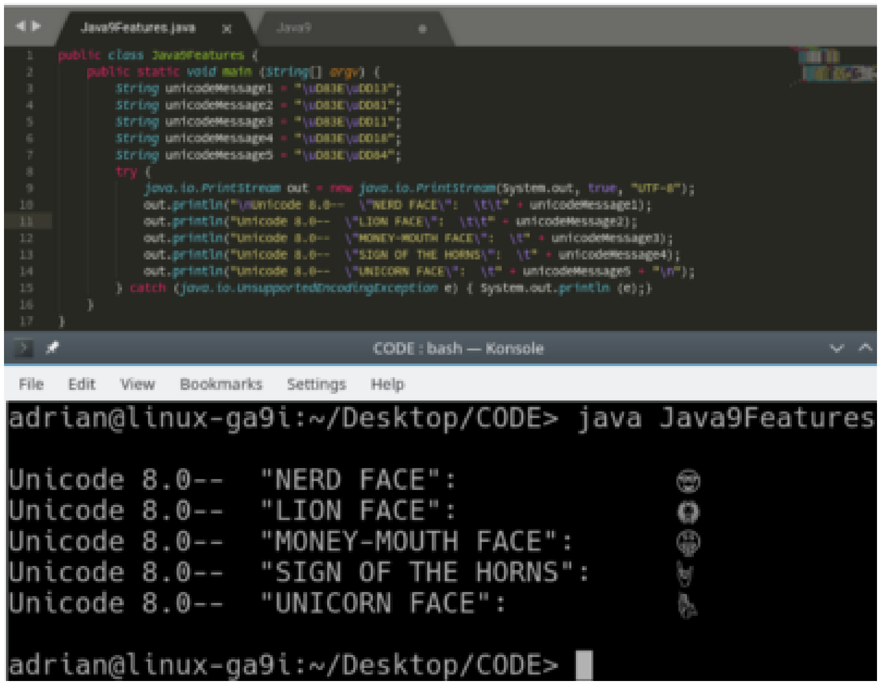
Let's begin.

Unicode 8.0 Support —

JDK 9 shipped with updates to Java's Unicode support bringing in the additional ~7,716 characters, 10 blocks, and 6 scripts defined by the Unicode 8 Standard[1]. JEP 267 was the community effort behind this feature.

Here's an example. I am running SUSE Linux—the latest version of openSUSE Tumbleweed with a KDE desktop (I highly recommend SUSE Linux! Mint & Fedora are great too:)).

Note: Due to limitations in operating systems and applications, Unicode characters are often not available in color on many systems. Additionally, while many systems provide Unicode support (with UTF-8 being the most widely used implementation), most systems do not fully implement the entire 1.1 million code points of the Unicode standard. Practically speaking, operating systems and applications will typically print a small blank box when tasked with displaying a Unicode glyph that is unavailable. Consequently, in order to get support for certain Unicode characters, I went ahead and grabbed the Fedora project—*gdouros-symbola-fonts* by George Douros. On SUSE linux you can grab the package on YaST or with zypper from the terminal by executing **zypper install gdouros-symbola-fonts** as **root** (prefix this command with **sudo**).



```java
public class Java9Features {
    public static void main (String[] argv) {
        String unicodeMessage1 = "\uD83E\uDD13";
        String unicodeMessage2 = "\uD83E\uDD81";
        String unicodeMessage3 = "\uD83E\uDD11";
        String unicodeMessage4 = "\uD83E\uDD18";
        String unicodeMessage5 = "\uD83E\uDD84";
        try {
            java.io.PrintStream out = new java.io.PrintStream(System.out, true, "UTF-8");
            out.println("\nUnicode 8.0--  \"NERD FACE\":  \t\t" + unicodeMessage1);
            out.println("Unicode 8.0--  \"LION FACE\":  \t\t" + unicodeMessage2);
            out.println("Unicode 8.0--  \"MONEY-MOUTH FACE\":  \t" + unicodeMessage3);
            out.println("Unicode 8.0--  \"SIGN OF THE HORNS\":  \t" + unicodeMessage4);
            out.println("Unicode 8.0--  \"UNICORN FACE\":  \t" + unicodeMessage5 + "\n");
        } catch (java.io.UnsupportedEncodingException e) { System.out.println (e);}
    }
}
```

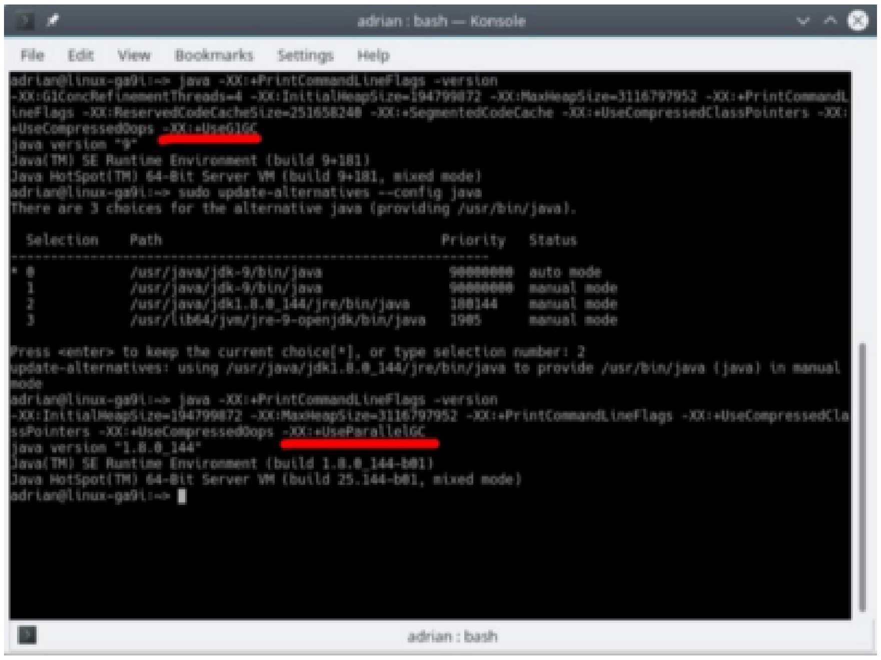
```
adrian@linux-ga9i:~/Desktop/CODE> java Java9Features

Unicode 8.0--  "NERD FACE":                 🤓
Unicode 8.0--  "LION FACE":                 🦁
Unicode 8.0--  "MONEY-MOUTH FACE":          🤑
Unicode 8.0--  "SIGN OF THE HORNS":         🤘
Unicode 8.0--  "UNICORN FACE":              🦄

adrian@linux-ga9i:~/Desktop/CODE> █
```

Printing some Unicode 8 Characters to the Console

Parallel Garbage Collector replaced as default in favor of G1 —

JDK 9 brings updates to Java's Garbage Collection functionality by replacing the Parallel/Throughput Garbage Collector with the Garbage-first (G1) Garbage Collector as the default. JEP 248 was the community effort behind this feature.

To figure out which Garbage Collector you are using,
java -XX:+PrintCommandLineFlags -version



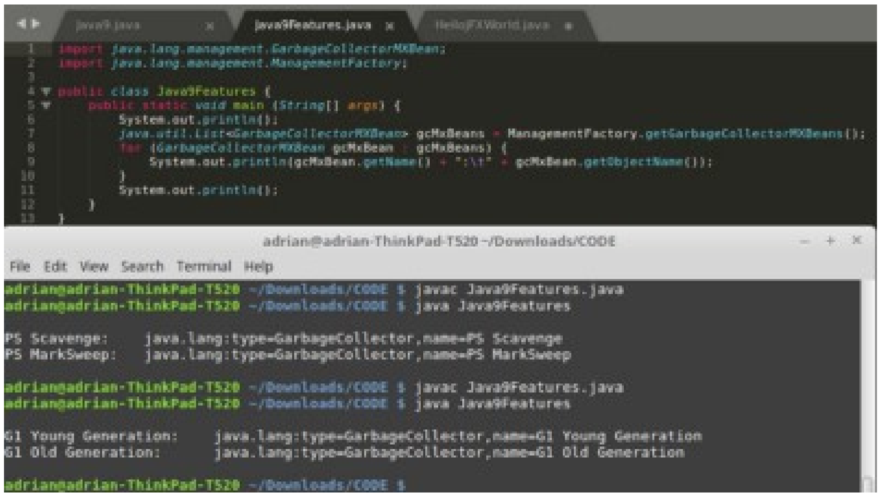
```
adrian@linux-ga9i:~> java -XX:+PrintCommandLineFlags -version
-XX:G1ConcRefinementThreads=4 -XX:InitialHeapSize=194799872 -XX:MaxHeapSize=3116797952 -XX:+PrintCommandL
ineFlags -XX:ReservedCodeCacheSize=251658240 -XX:+SegmentedCodeCache -XX:+UseCompressedClassPointers -XX:
+UseCompressedOops -XX:+UseG1GC
java version "9"
Java(TM) SE Runtime Environment (build 9+181)
Java HotSpot(TM) 64-Bit Server VM (build 9+181, mixed mode)
adrian@linux-ga9i:~> sudo update-alternatives --config java
There are 3 choices for the alternative java (providing /usr/bin/java).

  Selection    Path                                      Priority   Status
------------------------------------------------------------
* 0            /usr/java/jdk-9/bin/java                   90000000   auto mode
  1            /usr/java/jdk-9/bin/java                   90000000   manual mode
  2            /usr/java/jdk1.8.0_144/jre/bin/java        180144     manual mode
  3            /usr/lib64/jvm/jre-9-openjdk/bin/java      1905       manual mode

Press <enter> to keep the current choice[*], or type selection number: 2
update-alternatives: using /usr/java/jdk1.8.0_144/jre/bin/java to provide /usr/bin/java (java) in manual
mode
adrian@linux-ga9i:~> java -XX:+PrintCommandLineFlags -version
-XX:InitialHeapSize=194799872 -XX:MaxHeapSize=3116797952 -XX:+PrintCommandLineFlags -XX:+UseCompressedCla
ssPointers -XX:+UseCompressedOops -XX:+UseParallelGC
java version "1.8.0_144"
Java(TM) SE Runtime Environment (build 1.8.0_144-b01)
Java HotSpot(TM) 64-Bit Server VM (build 25.144-b01, mixed mode)
adrian@linux-ga9i:~> 
```

On SUSE Linux

In particular, pay attention to the **-XX:+UseG1GC and -XX:+UseParallelGC** flags. They show the change to the default garbage collector by JDK 9.

One of the primary motivations behind the switch to the G1 garbage collector was to reduce application pauses (Stop-The-World pauses) due to GC activity. Time will tell whether lessening application pauses is more valuable than throughput. The decision to set G1 as the default GC in JDK9 is not without controversy, however, as many in the community feel that the Concurrent Mark and Sweep (CMS) Garbage Collector should have been selected as the default GC instead of G1 as some benchmarks indicate that CMS has demonstrated better performance for the purposes that G1 is slated to do better than the Parallel/Throughput Garbage Collector.

> *How to programmatically determine the Garbage Collector*


Java9.java
Java9Features.java
HelloJFXWorld.java
adrian@adrian-ThinkPad-T520 ~/Downloads/CODE
File Edit View Search Terminal Help

```java
import java.lang.management.GarbageCollectorMXBean;
import java.lang.management.ManagementFactory;

public class Java9Features {
    public static void main (String[] args) {
        System.out.println();
        java.util.List<GarbageCollectorMXBean> gcMxBeans = ManagementFactory.getGarbageCollectorMXBeans();
        for (GarbageCollectorMXBean gcMxBean : gcMxBeans) {
            System.out.println(gcMxBean.getName() + ":\t" + gcMxBean.getObjectName());
        }
        System.out.println();
    }
}
```

```
adrian@adrian-ThinkPad-T520 ~/Downloads/CODE $ javac Java9Features.java
adrian@adrian-ThinkPad-T520 ~/Downloads/CODE $ java Java9Features

PS Scavenge:    java.lang:type=GarbageCollector,name=PS Scavenge
PS MarkSweep:   java.lang:type=GarbageCollector,name=PS MarkSweep

adrian@adrian-ThinkPad-T520 ~/Downloads/CODE $ javac Java9Features.java
adrian@adrian-ThinkPad-T520 ~/Downloads/CODE $ java Java9Features

G1 Young Generation:    java.lang:type=GarbageCollector,name=G1 Young Generation
G1 Old Generation:      java.lang:type=GarbageCollector,name=G1 Old Generation

adrian@adrian-ThinkPad-T520 ~/Downloads/CODE $
```

On Linux Mint

Applet API, Java Plug In deprecated

I don't have much to say about this other than—**Hurrah**! As much as I love Java, the Java plugin was **troublesome**, to say the least. The deprecation of the Java Applet was the natural consequence of this announcement. In JDK 9, java.applet.* and javax.swing.JApplet were marked deprecated.

JEP 289 was the community effort behind this feature.

Don't plan on using these any time soon :)

Compile for Older Platform versions

This is the really sweet one! Some foreground. By default, the javac compiler compiles the current version of the java APIs. Obvious enough. The consequence of this is that when using the -source and -target command line options to specify an early version of java, **javac will still compile the most recent version of the APIs**, leading to unpredictable or buggy code. Concomitantly, should there be a change in the APIs used between the two versions of java (the current compiler version and the ones targeted) the compiled class files may not successfully run on a JVM of the earlier targeted versions as specified by -source and -target. This used to be the case.

Got it? An example. Say you are using JDK 8. If you specify java 6 on -source and -target and consume a library that has changed in java 8, the resulting code **will not run** on a java 6 JVM because **the java 8 compiler will compile against the version of the current java libraries**.

It's a curious problem indeed. Nevertheless, JDK 9 introduces '-'-'release to solve this problem. Please note that the aforementioned dashes are intended to represent two consecutive dashes, however, unfortunately, due to medium's editor, this formatting cannot be achieved.

" `--release N` is roughly equivalent to:

- for N < 9: `-source N -target N -bootclasspath <documented-APIs-from-N>`,
- for N >= 9: `-source N -target N --system <documented-APIs-from-N>`.

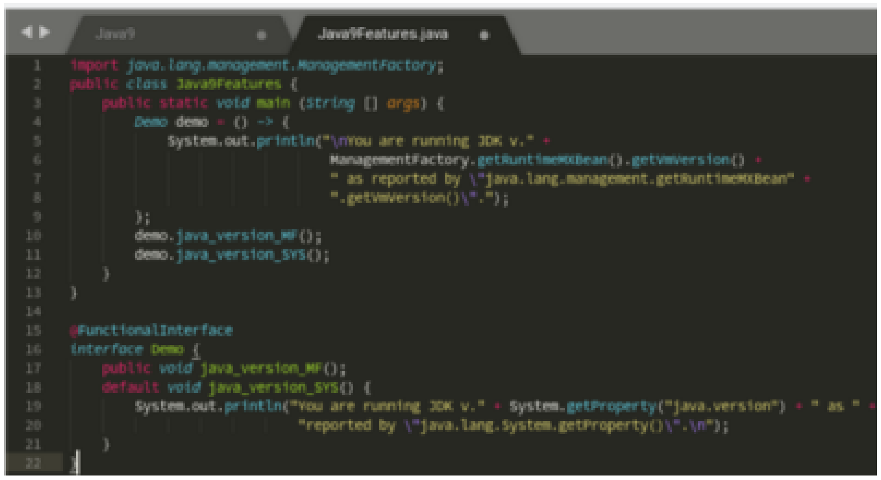
The `--release N` option is incompatible with other options that affect the set of platform or system classes." [2]

The command differences reflect the impact of Java's modules. It should be obvious to the reader, also, that the version N of -source N, -target N are automatically equivalent to the version N of '-'-'release N.

One small caveat: The one thing that does not change is the Unicode version. The Unicode version of the current platform will always be the one used.

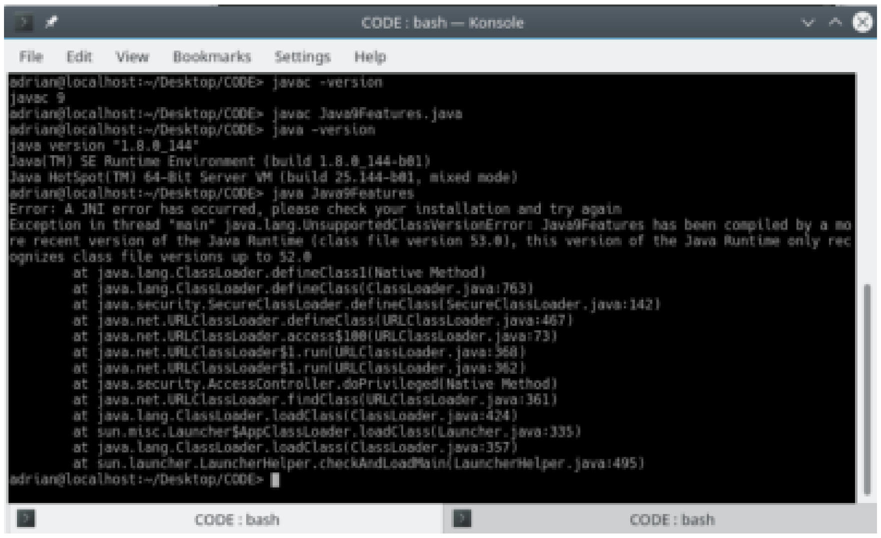
L**et's look at an Example.**

First, the code:



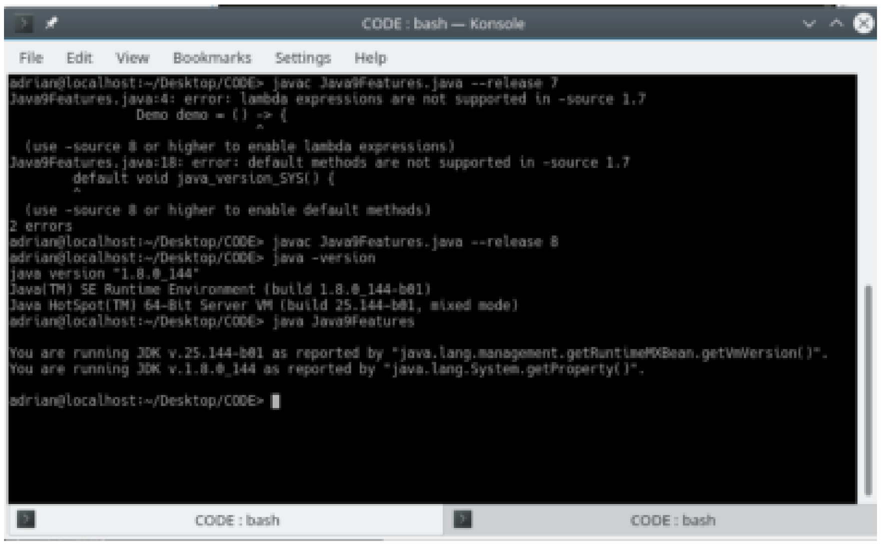
```java
import java.lang.management.ManagementFactory;
public class Java9Features {
    public static void main (String [] args) {
        Demo demo = () -> {
            System.out.println("\nYou are running JDK v." +
                                                ManagementFactory.getRuntimeMXBean().getVmVersion() +
                                                " as reported by \"java.lang.management.getRuntimeMXBean" +
                                                ".getVmVersion()\".");
        };
        demo.java_version_MF();
        demo.java_version_SYS();
    }
}

@FunctionalInterface
interface Demo {
    public void java_version_MF();
    default void java_version_SYS() {
        System.out.println("You are running JDK v." + System.getProperty("java.version") + " as " +
                                        "reported by \"java.lang.System.getProperty()\".\n");
    }
}
```

Second, we'll compile the source with javac v.9 and attempt to run the code on a Java 8 VM to demonstrate it's failure.



```
adrian@localhost:~/Desktop/CODE> javac -version
javac 9
adrian@localhost:~/Desktop/CODE> javac Java9Features.java
adrian@localhost:~/Desktop/CODE> java -version
java version "1.8.0_144"
Java(TM) SE Runtime Environment (build 1.8.0_144-b01)
Java HotSpot(TM) 64-Bit Server VM (build 25.144-b01, mixed mode)
adrian@localhost:~/Desktop/CODE> java Java9Features
Error: A JNI error has occurred, please check your installation and try again
Exception in thread "main" java.lang.UnsupportedClassVersionError: Java9Features has been compiled by a mo
re recent version of the Java Runtime (class file version 53.0), this version of the Java Runtime only rec
ognizes class file versions up to 52.0
        at java.lang.ClassLoader.defineClass1(Native Method)
        at java.lang.ClassLoader.defineClass(ClassLoader.java:763)
        at java.security.SecureClassLoader.defineClass(SecureClassLoader.java:142)
        at java.net.URLClassLoader.defineClass(URLClassLoader.java:467)
        at java.net.URLClassLoader.access$100(URLClassLoader.java:73)
        at java.net.URLClassLoader$1.run(URLClassLoader.java:368)
        at java.net.URLClassLoader$1.run(URLClassLoader.java:362)
        at java.security.AccessController.doPrivileged(Native Method)
        at java.net.URLClassLoader.findClass(URLClassLoader.java:361)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:424)
        at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:335)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:357)
        at sun.launcher.LauncherHelper.checkAndLoadMain(LauncherHelper.java:495)
adrian@localhost:~/Desktop/CODE> 
```

Lastly, we'll demonstrate that it understands the differences between versions (compiling with lambdas against javac '-'-'7 fails), and we will demonstrate that compiling it with the '-'-'release flag works when we run it on a Java 8 VM.



```
adrian@localhost:~/Desktop/CODE> javac Java9Features.java --release 7
Java9Features.java:4: error: lambda expressions are not supported in -source 1.7
                Demo demo = () -> {
                                ^
  (use -source 8 or higher to enable lambda expressions)
Java9Features.java:18: error: default methods are not supported in -source 1.7
        default void java_version_SYS() {
        ^
  (use -source 8 or higher to enable default methods)
2 errors
adrian@localhost:~/Desktop/CODE> javac Java9Features.java --release 8
adrian@localhost:~/Desktop/CODE> java -version
java version "1.8.0_144"
Java(TM) SE Runtime Environment (build 1.8.0_144-b01)
Java HotSpot(TM) 64-Bit Server VM (build 25.144-b01, mixed mode)
adrian@localhost:~/Desktop/CODE> java Java9Features

You are running JDK v.25.144-b01 as reported by "java.lang.management.getRuntimeMXBean.getVmVersion()".
You are running JDK v.1.8.0_144 as reported by "java.lang.System.getProperty()".

adrian@localhost:~/Desktop/CODE> 
```

The source really isn't that important as it will always simply tell us the version of the current VM that is executing the code, no matter which version of javac we compile with. Also, note that the tool supports compilation going back to Java 6 only.

The compiler gives us "warning: [options] source value 1.6 is obsolete and will be removed in a future release"[3] when we use the release flag with JDK 6, indicating that support for compiling against JDK 6 may be gone soon. It does not give such an error with JDK's 7 or 8. Maybe they'll always support 4 versions moving forward—the current version and the three behind it. Maybe not.

JEP 247 was the community effort behind this feature.

Java REPL (jshell)

This one is exciting. The JShell tool is a Java REPL (Read-Evaluate-Print-Loop) tool used to interactively evaluate java statements, declarations, and expressions. If you are familiar with Python this may seem analogous to Python's interpreter. While this is not exactly the case it is similar. JDK 9 also ships JShell with an API for JShell integration into third party apps and services. **Semicolons are not required to end statements except inside code blocks** (so far as I have observed).

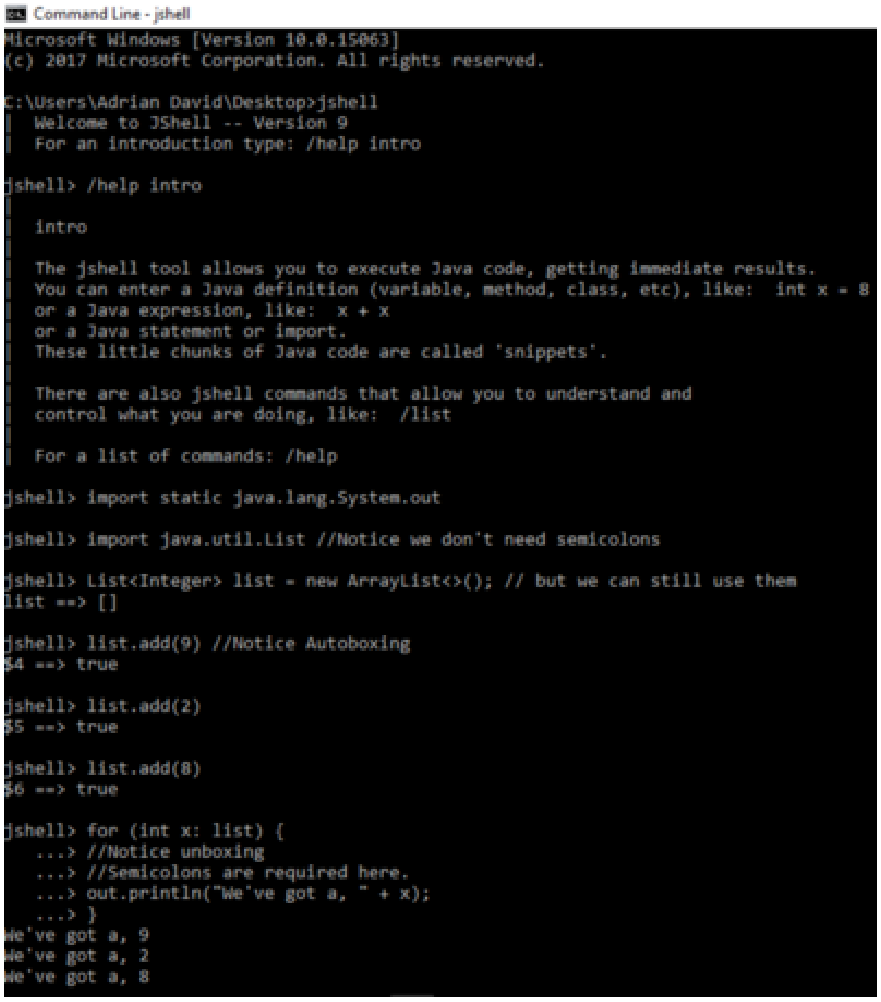
The two main advantages of JShell (in my opinion) are **prototyping** and **teaching**. JShell provides a quick and easy way to prototype java code without having to encapsulate it in a public class along with the main(String [] args) method and provides a simpler, more intuitive solution than the traditional edit →compile →run cycle for this means.

In terms of teaching, this is great because beginner's can see their code working in action. Java will now join the ranks of Scala, Ruby, JavaScript, Haskell, Clojure, Python, LISP, etc in having a REPL tool.

According to Robert Field (openJDK developer)—"The number one reason schools cite for moving away from Java as a teaching language is that other languages have a "REPL" and have far lower bars to an initial "Hello, world!" program."[4]

Personally, this is wacky to me, as it is simple enough to construct a basic "Hello, World" program in Java. While you can do a basic "Hello, World" in a language such as python, for example, in one line, one should probably not get comfortable with things being that easy in Software Development. If grappling "Hello, World" in Java presents a challenge, it may not be a good indicator of one's prospects in the field. Maybe I'm wrong. After all, the notion of a class, method arguments, packages, access modifiers, etc are all present in even the simplest "Hello World" Java example. But this finding does surprise me, because in my view, the matter is trivial and inconsequential. I consider it *de minimis*. The concepts should be easy enough to understand for most people and software development will only get harder.

Nevertheless, have a look at JShell.



```
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\Adrian David\Desktop>jshell
|  Welcome to JShell -- Version 9
|  For an introduction type: /help intro

jshell> /help intro
|
|  intro
|
|  The jshell tool allows you to execute Java code, getting immediate results.
|  You can enter a Java definition (variable, method, class, etc), like:  int x = 8
|  or a Java expression, like:  x + x
|  or a Java statement or import.
|  These little chunks of Java code are called 'snippets'.
|
|  There are also jshell commands that allow you to understand and
|  control what you are doing, like:  /list
|
|  For a list of commands: /help

jshell> import static java.lang.System.out

jshell> import java.util.List //Notice we don't need semicolons

jshell> List<Integer> list = new ArrayList<>(); // but we can still use them
list ==> []

jshell> list.add(9) //Notice Autoboxing
$4 ==> true

jshell> list.add(2)
$5 ==> true

jshell> list.add(8)
$6 ==> true

jshell> for (int x: list) {
   ...> //Notice unboxing
   ...> //Semicolons are required here.
   ...> out.println("We've got a, " + x);
   ...> }
We've got a, 9
We've got a, 2
We've got a, 8
```

Windows 10 box

JEP 222 was the community effort behind this feature.

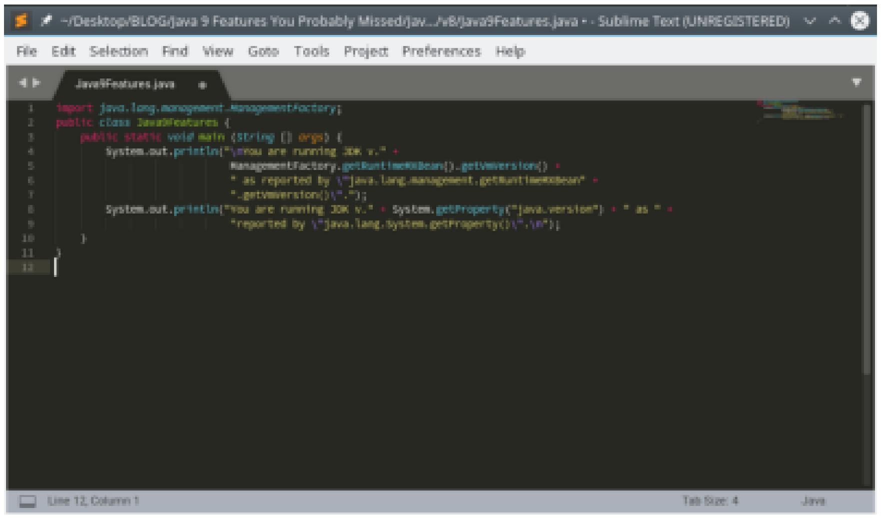
Multi-Release JAR Files

Multi-Release JAR Files (MRJAR) are Application Archives that allow java-version specific class files to coexist in an archive with other java class files of another java version.

The impetus behind this feature is to incentive developers of third party API to upgrade their API to the most recent Java platform. Often library and framework developers keep several versions of their software because the consumers of their software are typically slow to upgrade (the reasons for which are beyond the scope of this article). The main challenge arises in conditionally determining platform dependencies. With MRJARs, third party API developers can more easily ship different versions of their software. More often than not, this tool will mostly be used within build tools such as Maven or Gradle.

An example

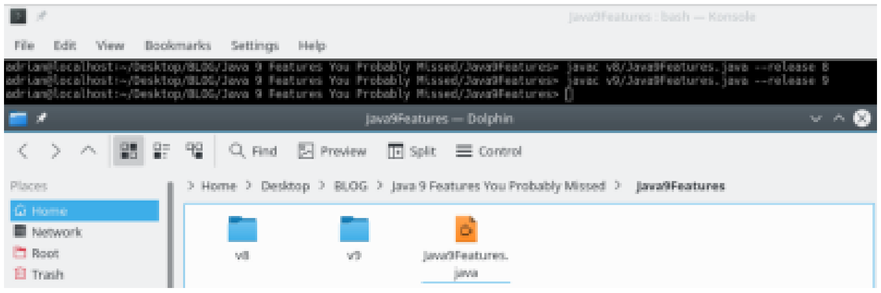
The Code


~/Desktop/BLOG/Java 9 Features You Probably Missed/jav.../v8/Java9Features.java • - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
Java9Features.java
Line 12, Column 1
Tab Size: 4
Java

```
import java.lang.management.ManagementFactory;
public class Java9Features {
    public static void main (String [] args) {
        System.out.println("\nYou are running JDK v." +
                            ManagementFactory.getRuntimeMXBean().getVmVersion() +
                            " as reported by \"java.lang.management.getRuntimeMXBean" +
                            ".getVmVersion()\".");
        System.out.println("You are running JDK v." + System.getProperty("java.version") + " as " +
                            "reported by \"java.lang.System.getProperty()\".\n");
    }
}
```

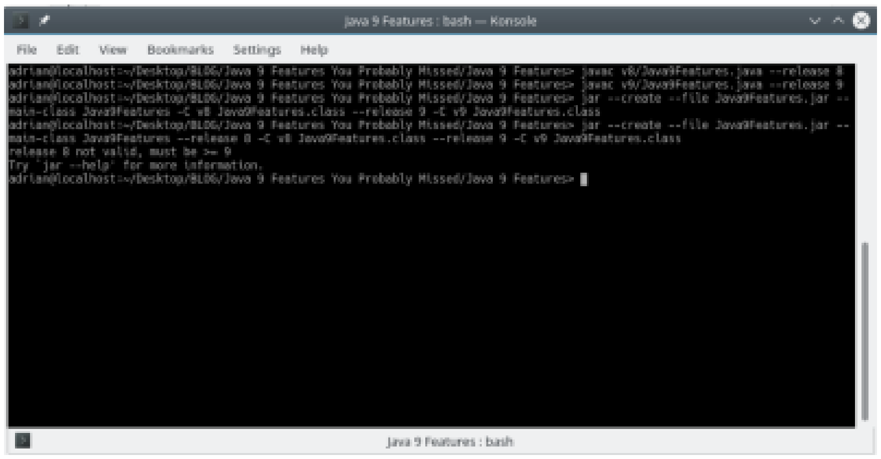
Notice that I have provided only one source file. This is just to show you the basic code that will be compiled. There is one copy of this file per version. Obviously, if I don't make any changes between the versions, you won't be able to tell from the output that there is any difference. Because of this, I will modify the files by adding "System.out.println("Version #1)" in one and "System.out.println("Version #2)" in the other.

Step #1—Compile the various versions of class files


Java9Features : bash — Konsole
File Edit View Bookmarks Settings Help
Java9Features — Dolphin
Find
Preview
Split
Control
Places
Home
Network
Root
Trash
Home > Desktop > BLOG > Java 9 Features You Probably Missed > Java9Features
v8
v9
Java9Features.java

```
adrian@localhost:~/Desktop/BLOG/Java 9 Features You Probably Missed/Java9Features> javac v8/Java9Features.java --release 8
adrian@localhost:~/Desktop/BLOG/Java 9 Features You Probably Missed/Java9Features> javac v9/Java9Features.java --release 9
adrian@localhost:~/Desktop/BLOG/Java 9 Features You Probably Missed/Java9Features> 
```

Step#2—Compile the JAR file


Java 9 Features : bash — Konsole
File Edit View Bookmarks Settings Help
Java 9 Features : bash

```
adrian@localhost:~/Desktop/BLOG/Java 9 Features You Probably Missed/Java 9 Features> javac v8/Java9Features.java --release 8
adrian@localhost:~/Desktop/BLOG/Java 9 Features You Probably Missed/Java 9 Features> javac v9/Java9Features.java --release 9
adrian@localhost:~/Desktop/BLOG/Java 9 Features You Probably Missed/Java 9 Features> jar --create --file Java9Features.jar --
main-class Java9Features -C v8 Java9Features.class --release 9 -C v9 Java9Features.class
adrian@localhost:~/Desktop/BLOG/Java 9 Features You Probably Missed/Java 9 Features> jar --create --file Java9Features.jar --
main-class Java9Features --release 8 -C v8 Java9Features.class --release 9 -C v9 Java9Features.class
release 8 not valid, must be >= 9
Try `jar --help' for more information.
adrian@localhost:~/Desktop/BLOG/Java 9 Features You Probably Missed/Java 9 Features> 
```

Notice that the JAR tool does not support creating using '-'-'release with any version less than 9. This is why we put the Java 8 version in the root. With Java 10 we will be able to use '-'-'release 10 moving into the foreseeable future.

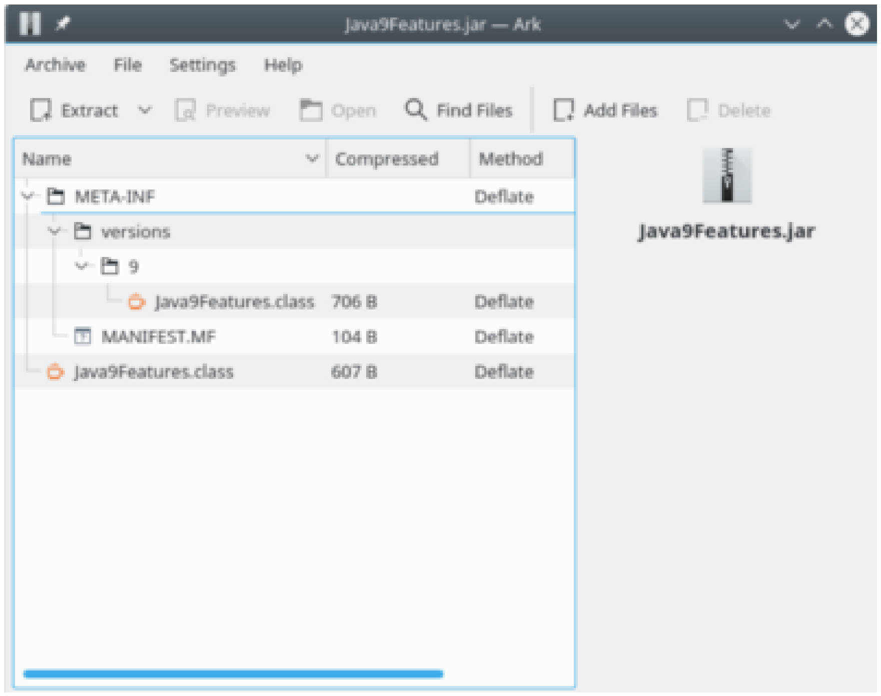
The Commands I ran were:
jar—create—file Java9.jar—main-class Java9Features -C v8 Java9Features.class—release 9 -C v9 Java9Features.class

jar—create—file Java9.jar—main-class Java9Features—release 8-C v8 Java9Features.class—release 9 -C v9 Java9Features.class (**Fails**)

This fails, again, because Java 8 is not supported with—release.

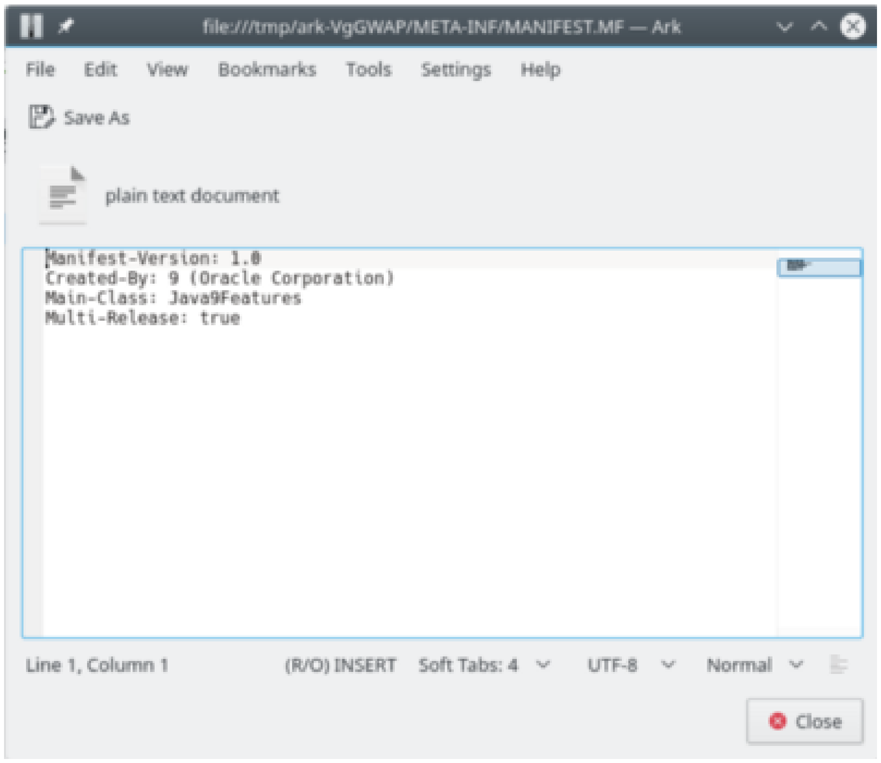
Remember, again, that medium formats two consecutive dashes as " —", so copying and pasting this command will not work.

Notice the JAR Archive Structure



Name	Compressed	Method
META-INF		Deflate
versions		
9		
Java9Features.class	706 B	Deflate
MANIFEST.MF	104 B	Deflate
Java9Features.class	607 B	Deflate

Notice the Application Manifest

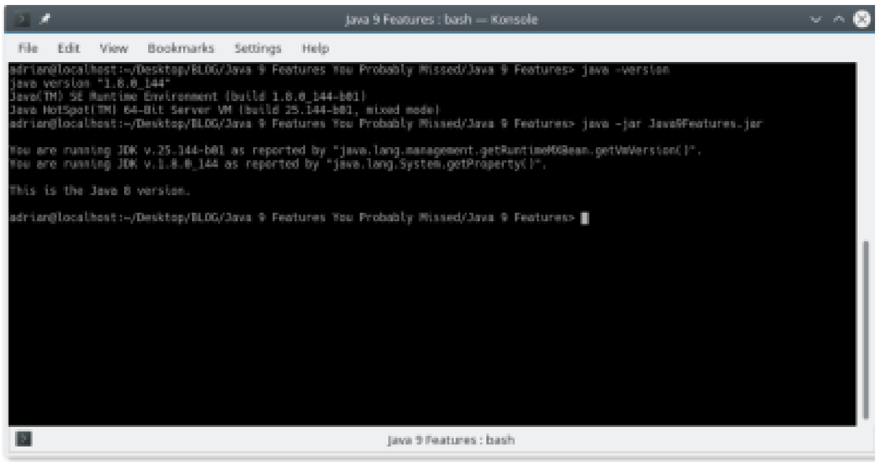


```
Manifest-Version: 1.0
Created-By: 9 (Oracle Corporation)
Main-Class: Java9Features
Multi-Release: true
```

Multi-Release: true is added to let java tools interacting with the archive be aware that the archive has multiple versions.

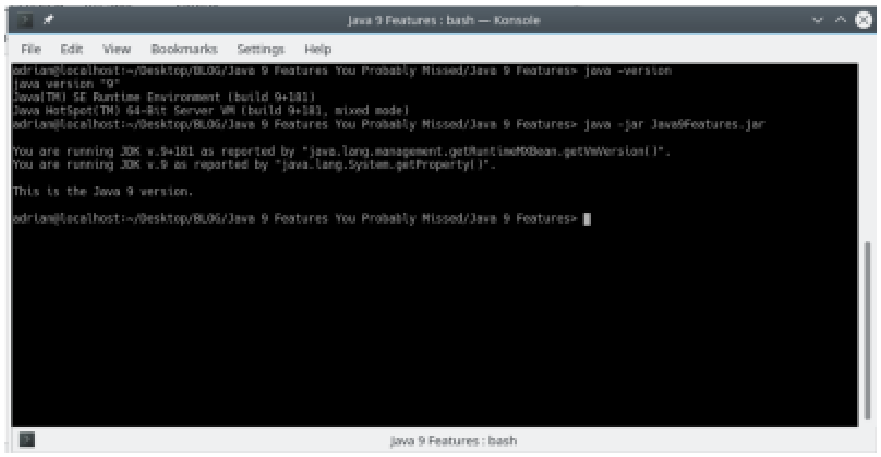
Step #3—Test the code by running it. Run it in both Java 8 and Java 9

In Java 8,



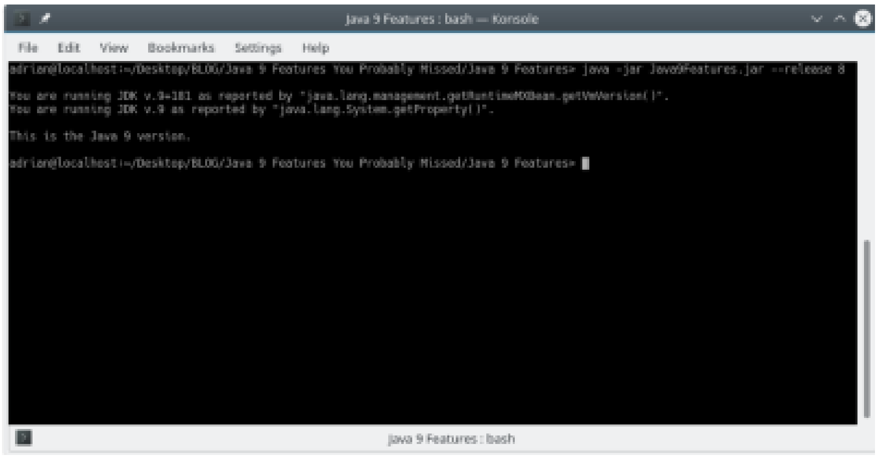
```
adrian@localhost:~/Desktop/BLOG/Java 9 Features You Probably Missed/Java 9 Features> java -version
java version "1.8.0_144"
Java(TM) SE Runtime Environment (build 1.8.0_144-b01)
Java HotSpot(TM) 64-Bit Server VM (build 25.144-b01, mixed mode)
adrian@localhost:~/Desktop/BLOG/Java 9 Features You Probably Missed/Java 9 Features> java -jar Java9Features.jar

You are running JDK v.25.144-b01 as reported by "java.lang.management.getRuntimeMXBean.getVmVersion()".
You are running JDK v.1.8.0_144 as reported by "java.lang.System.getProperty()".

This is the Java 8 version.

adrian@localhost:~/Desktop/BLOG/Java 9 Features You Probably Missed/Java 9 Features>
```

In Java 9,



```
adrian@localhost:~/Desktop/BLOG/Java 9 Features You Probably Missed/Java 9 Features> java -version
java version "9"
Java(TM) SE Runtime Environment (build 9+181)
Java HotSpot(TM) 64-Bit Server VM (build 9+181, mixed mode)
adrian@localhost:~/Desktop/BLOG/Java 9 Features You Probably Missed/Java 9 Features> java -jar Java9Features.jar

You are running JDK v.9+181 as reported by "java.lang.management.getRuntimeMXBean.getVmVersion()".
You are running JDK v.9 as reported by "java.lang.System.getProperty()".

This is the Java 9 version.

adrian@localhost:~/Desktop/BLOG/Java 9 Features You Probably Missed/Java 9 Features>
```

What if we used the Java 9 Compiler to attempt to run the code as Java 8 with java -jar Java9Features.java—release 8 ? It won't work.


Java 9 Features : bash — Konsole
File Edit View Bookmarks Settings Help
Java 9 Features : bash

```
adrian@localhost:~/Desktop/BLOG/Java 9 Features You Probably Missed/Java 9 Features> java -jar Java9Features.jar --release 8

You are running JDK v.9+181 as reported by "java.lang.management.getRuntimeMXBean.getVmVersion()".
You are running JDK v.9 as reported by "java.lang.System.getProperty()".

This is the Java 9 version.

adrian@localhost:~/Desktop/BLOG/Java 9 Features You Probably Missed/Java 9 Features>
```

See the documentation for more information.

JEP 238 was the community effort behind this feature.

Nashorn ES6 Support

While I won't go deep into ECMAScript (the most popular implementation being JavaScript), Java 9 brought significant support for using JavaScript in Java.

The first feature is the Parser API for Nashorn which will provide a parser for ECMAScript's Abstract Synatx Tree. This relieves developers from having to resort to the Nashorn internal APIs.

Secondly, JDK 9 is bringing in (some) ES6 Support. ES6 is a very exciting release bringing in much needed improvements to the standard. JDK 9 will implement many ES 6 features in Nashorn.

JEP 236 & 292 were the community efforts behind these features.

Some Other Java 9 Features to check out

jlink—"Create a tool that can assemble and optimize a set of modules and their dependencies into a custom run-time image as defined in JEP 220."[6]

jdeprscan—Scan a JAR for uses of deprecated JDK API

Don't see your favourite? Let me know in the comments!

Want the source? Grab it here.

afinlay5/CoolJava9Ft

Gradle source code repository for Java 9 source code examples posted on personal blog...

github.com

Looney Tunes Ending [8]

Works Cited

[1]—http://unicode.org/versions/Unicode8.0.0/

[2]—JEP247: Compile for Older Platform Versions

[3]—Oracle JDK javac Compiler

[4]—JEP 222: jshell: The Java Shell (Read-Eval-Print Loop)

[5]—https://www.youtube.com/watch?v=0FHEeG_uq5Y

[6]—JEP 282: jlink: The Java Linker