# The road from Java 8 to Java 11

Davide Angelocola

Version v0.6.5, 2018-10-16

# Table of Contents

# Introduction

As Java developers, we will be busy in the upcoming months upgrading to the newest Java releases. It will be a slow, incremental process, especially regarding the migration from `classpath` to the `modulepath`. This document tries to summarize new features in a very minimalistic style:

- focus is on the language level changes, secondarily on deploy/production features
- use standard **Java Class Library** as much as possible
- use **JShell** as much as possible
- provide links to release notes, JEPs and bug reports

# Author


- Davide Angelocola https://dfa.bitbucket.io


# Contributors

- Karrie Moore
- Alessandro Sebastiani

# License


This document is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

# Upgrade to Java 11

https://jdk.java.net/11/release-notes

Java 11 is the new LTS (Long Term Support), so it is über-important to upgrade as soon as possible to this version. These are the main features of Java 11:

- stable `java.net.http` module
- TLS 1.3
- removal of Corba and Java EE
- removal of Web Start, with no clear replacement
- removal of Java applets
- removal of JavaFX: the FX libraries have moved to the OpenJFX project

## Which JDK?

Since JDK11 is a commercial product, to use it in production you have to pay Oracle, details here: https://www.oracle.com/technetwork/java/javaseproducts/overview/javasesubscriptionfaq-4891443.html.

However, there are free, zero-cost, alternatives: OpenJDK binaries with related docker images. Checkout also the adoptopenjdk project, that provides pre-built JDK binaries as well as docker images. More details can be found https://blog.joda.org/2018/09/time-to-look-beyond-oracles-jdk.html.

## Solving migration problems

- https://openjdk.java.net/jeps/320 provide missing JARs
- removal of `Thread.destroy()` and `Thread.stop(Throwable)` methods, see https://bugs.openjdk.java.net/browse/JDK-8204243
- removal of `com.sun.awt.AWTUtilities` class, see https://bugs.openjdk.java.net/browse/JDK-8200149
- removal of removal of `sun.misc.Unsafe.defineClass`, see https://bugs.openjdk.java.net/browse/JDK-8193033
- source incompatibility for `java.util.Stream.toArray(null)`, see https://bugs.openjdk.java.net/browse/JDK-8060192

For example, let's start from JEP 320. Several packages, such as JAXB, are not provided by the JDK anymore so it is required to provide an external dependency for them.

For example using JAXB in an Apache Maven just add:

```
<dependency>
    <groupId>javax.xml.bind</groupId>
    <artifactId>jaxb-api</artifactId>
    <version>2.3.0</version>
</dependency>
<dependency>
    <groupId>org.glassfish.jaxb</groupId>
    <artifactId>jaxb-runtime</artifactId>
    <version>2.3.0.1</version>
</dependency>
```

This is a very good reference: https://stackoverflow.com/questions/48204141/replacements-for-deprecated-jpms-modules-with-java-ee-apis/48204154#48204154

# Launch Single-File Source-Code Programs

https://openjdk.java.net/jeps/330

Historically to run a Java program we had to:

```
dfa@aman:~ $ vim Hello.java
dfa@aman:~ $ cat Hello.java ①
public class Hello { ①
    public static void main(String[] args) {
        System.out.println("hello world!");
    }
}
dfa@aman:~ $ javac Hello.java ②
dfa@aman:~ $ java Hello ②
hello world!
```

① class name and file name must match

② two separate steps: compile and run

Now it is possible to edit and run a single-file Java program with much less ceremony:

```
dfa@aman:~ $ vim demo.java
dfa@aman:~ $ cat demo.java ①
public class Hello { ①
    public static void main(String[] args) {
        System.out.println("hello world!");
    }
}
dfa@aman:~ $ java hello.java ②
hello world!
```

① it is also possible to name file and class differently

② one step to compile and run

## java.net.http

https://openjdk.java.net/groups/net/httpclient/intro.html

Introduced in Java 9, and promoted from incubator in Java 11. This is a rather big API and it is a huge step forward from `java.net.URLConnection`.

Let's start with an example with `httpbin`:

```
dfa@aman:~ $ docker run --rm -p 80:80 kennethreitz/httpbin
[2018-09-29 10:04:20 +0000] [1] [INFO] Starting gunicorn 19.9.0
[2018-09-29 10:04:20 +0000] [1] [INFO] Listening at: http://0.0.0.0:80 (1)
[2018-09-29 10:04:20 +0000] [1] [INFO] Using worker: gevent
[2018-09-29 10:04:20 +0000] [9] [INFO] Booting worker with pid: 9
```

this container exposes well-known resources, check https://www.kennethreitz.org/essays/httpbin for reference.

Let's test it with curl:

```
dfa@aman:~ $ curl localhost/user-agent
{
  "user-agent": "curl/7.54.0"
}
```

Everything looks fine, not let's write a simple HTTP2 client using `java.net.http` module:

```
import java.net.URI;
import java.net.http.*;
import java.time.Duration;

public class HttpClientDemo {
        public static void main(String[] args) throws Exception {
                if (args.length != 1) {
                    System.err.println("missing URL argument");
                    System.exit(1);
                }
                URI uri = URI.create(args[0]);
                HttpClient client = HttpClient.newBuilder()
                        .version(HttpClient.Version.HTTP_2)
                        .build();
                HttpRequest request = HttpRequest.newBuilder()
                        .uri(uri)
                        .timeout(Duration.ofSeconds(1))
                        .build();
                HttpResponse<String> response = client.send(request, HttpResponse
.BodyHandlers.ofString());
                System.out.print(response.body());
        }
}
```

and let's start it using JEP 330:

```
dfa@aman:~ $ java java11_http_client.java http://localhost/user-agent
{
  "user-agent": "Java-http-client/11"
}
```

Let's try simulating a busy web server, that delays each request by 2 seconds (whereas our client timeouts after 1 second):

```
dfa@aman:~ $ java java11_http_client.java http://localhost/delay/2
Exception in thread "main" java.net.http.HttpTimeoutException: request timed out ①
    at
java.net.http/jdk.internal.net.http.HttpClientImpl.send(HttpClientImpl.java:559)
    at
java.net.http/jdk.internal.net.http.HttpClientFacade.send(HttpClientFacade.java:119)
    at HttpClientDemo.main(java11_http_client.java:19)
```

① as expected a timeout is triggered on the client client side

### `WebSocket` and `WebSocketListener`

This modules supports also web sockets:

```
var client = HttpClient.newHttpClient();
var uri = URI.create(...);
var listener = ...;
var ws = client.newWebSocketBuilder().buildAsync(uri, listener);
```

## Security

This version includes several new important and modern crypto features:

- TLS 1.3, see https://bugs.openjdk.java.net/browse/JDK-8202625
- ChaCha20 & Poly1305 crypto algorithms
- Key Agreement with Curve25519 and Curve448

# var in lambda

https://openjdk.java.net/jeps/323

```
(var x, var y) -> x.process(y)
```

now it is equivalent to:

```
(x, y) -> x.process(y)
```

The primary advantage is that now it is possible to annotate parameters, e.g. for static analysis.

## `Optional.isEmpty()`

In addition of `Optional.isPresent` now it is possible to use `Optional.isEmpty`:

```
Optional<String> featureToggle = ...;
if (featureToggle.isEmpty()) {
    logger.warn("feature 'xxx' disabled");
}
```

## `ArrayIndexOutOfBounds`

Improved error message with index and current size of the array:

```
jshell> int[] a = { 1 }
a ==> int[1] { 1 }

jshell> a[4]
|  Exception java.lang.ArrayIndexOutOfBoundsException: Index 4 out of bounds for
length 1
|        at (#2:1)
```

Before this change the message was much more cryptic:

```
jshell> int[] a = { 1 }
a ==> int[1] { 1 }

jshell> a[4]
|  java.lang.ArrayIndexOutOfBoundsException thrown: 4
|        at (#4:1)
```

## Character.toString(int)

This method returns the string representation for the given Unicode code point as shown below:

```
String d = Character.toString(100)
assert d.equals("d")

String a = Character.toString(65);
assert a.equals("A")
```

## String.lines()

Create a `Stream<String>` by lazily splitting string using line separators (e.g. `"\n"` `"\r"` `"\r\n"`):

```
jshell> "a\nb\nc\n".lines().map(String::toUpperCase).toArray()
$1 ==> Object[2] { "A", "B", "C }
```

## String.repeat()

Repeat `String` for the specified number of times:

```
jshell> "ab".repeat(5) ①
$1 ==> "abababababab"

jshell> "ab".repeat(1) ②
$2 ==> "ab"

jshell> "ab".repeat(0) ③
$3 ==> ""

jshell> "ab".repeat(-1) ④
|  Exception java.lang.IllegalArgumentException: count is negative: -1
|        at String.repeat (String.java:3149)
|        at (#1:1)

jshell> "ab".repeat(Integer.MAX_VALUE) ⑤
|  Exception java.lang.OutOfMemoryError: Repeating 4 bytes String 2147483647 times
will produce a String exceeding maximum size.
|        at String.repeat (String.java:3164)
|        at (#2:1)
```

① n=5 as expected result is `"ab"` repeated five times

② corner case `n=1`, result is `"ab"`

③ corner case `n=0`, result is empty `""`

④ error, since `n<0`

⑤ error fail fast method to avoid allocating big chunks of memory

## `String.isBlank()`

https://bugs.openjdk.java.net/browse/JDK-8200437

This is a Unicode-aware alternative to `isEmpty()`:

```
jshell> var halfSpace = "\u0020"
halfSpace ==> " "

jshell> var fullSpace = "\u3000"
fullSpace ==> "　"

jshell> halfSpace.trim().isEmpty()
$1 ==> true ①

jshell> fullSpace.trim().isEmpty()
$2 ==> false ②
```

① working as expected

② not working as expected

To fix this problem let's use `Character.isWhitespace` method, that is aware of the different types of spaces:

```
boolean blank = string.codePoints().allMatch(Character::isWhitespace);
```

This is correct but too technical and, perhaps, the intent is not clear:

As per Java 11 it is possible to just use `String.isBlank`:

```
jshell> var halfSpace = "\u0020"
halfSpace ==> " "

jshell> var fullSpace = "\u3000"
fullSpace ==> "　"

jshell> halfSpace.repeat(10).isBlank()
$1 ==> true

jshell> fullSpace.repeat(10).isBlank()
$2 ==> true
```

Be aware that there are surprising results around the definition of `Character.isWhitespace`, such as non-breaking spaces or newlines:

```
jshell> var nonBreakingSpace = "\u00A0"
nonBreakingSpace ==> " "

jshell> nonBreakingSpace.isBlank()
$1 ==> false
```

## `String.strip()`

https://bugs.openjdk.java.net/browse/JDK-8200378

While almost the same as `trim()`/`trimLeft()`/`trimRight()`, this takes full-width spaces as a space (0x20 ASCII character).

```
jshell> var halfSpace = "\u0020"
halfSpace ==> " "

jshell> halfSpace.trim()
$2 ==> ""

jshell> var fullSpace = "\u3000"
$3 ==> "　"

jshell> fullSpace.trim()
$4 ==> "　" ①

jshell> fullSpace.strip()
$5 ==> "" ②
```

① not working as expected

② working as expected

Finally let's cover quickly `stripLeading()`/`stripTrailing()`:

```
jshell> var text = fullSpace + "foo bar" + fullSpace
text ==> "　 foo bar 　"

jshell> text.stripTrailing()
$7 ==> "　 foo bar"

jshell> text.stripLeading()
$8 ==> "foo bar 　"
```

## CharSequence.compare()

Using this new API it is possible to compare any `CharSequence` implementation:

```
jshell> var builder = new StringBuilder("aaa");
builder ==> aaa

jshell> var buffer = new StringBuffer("aaa");
buffer ==> aaa

jshell> var string = "aaa";
string ==> "aaa"

jshell> CharSequence.compare(builder, buffer); ①
$1 ==> 0

jshell> CharSequence.compare(string, buffer); ②
$2 ==> 0
```

① comparing a `StringBuilder` with `StringBuffer` yields 0

② ditto for comparing a `String` with `StringBuffer`

# Null objects for Reader/Writer and InputStream/OutputStream

https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/io/Reader.html#nullReader()

- `Reader.nullReader()`
- `Writer.nullWriter()`
- `InputStream.nullInputStream()`
- `OutputStream.nullOutputStream()`

These objects are very useful during unit testing.

## java.nio.file.Files

- `String readString(Path, Charset)`: reads all content from a file into a string, decoding from bytes to characters using the specified charset;
- `Path writeString(Path, CharSequence, Charset, OpenOption[])`: write a `CharSequence` (e.g. `String`, `StringBuilder`) to a file. Characters are encoded into bytes using the specified charset.

It is very convenient to use `java.nio.charset.StandardCharsets` (introduced in Java 7, see javadoc):

```
String content = Files.readString(Path.of("main.adoc"), StandardCharsets.UTF_8);
```

## java.nio.Path

This is a very nice shortcut to build paths:

```
jshell> Path.of("dir", "subdir", "file")
$2 ==> dir/subdir/file
```

Returns a `Path` by converting a path string, or a sequence of strings that when joined form a path string. Please note that this API is following the *Item 42* of *Effective Java*:

```
jshell> Path.of()
|  Error:
|  no suitable method found for of(no arguments)
|      method java.nio.file.Path.of(java.lang.String,java.lang.String...) is not
applicable
|        (actual and formal argument lists differ in length)
|      method java.nio.file.Path.of(java.net.URI) is not applicable
|        (actual and formal argument lists differ in length)
|  Path.of()
|  ^-----^
```

because the definition is:

```
static Path of(String first, String... more) {
    ...
}
```

# Unicode

https://openjdk.java.net/jeps/327

This release includes combined support for both *Unicode 9.0* as well as *Unicode 10.0*.

By now it is possible to use the Bitcoin sign (code point `U+20BF`) released on June 2017 as part of *Unicode 10.0*.

# Dynamic Class-File Constants

https://openjdk.java.net/jeps/309

This is mostly for compilers that target the JVM. However it could have interesting ripples in the whole ecosystem, as `invokedynamic` did.

# Upgrade to Java 10

http://openjdk.java.net/projects/jdk/10/

Main features:

- JEP 286 local type inference
- JEP 304 garbage collector interface
- JEP 317 experimental java based JIT compiler
- JEP 307 Parallel full gc for G1
- JEP 310 application class-data sharing
- JEP 312 thread-local handshakes
- JEP 313 javah removal
- JEP 314 unicode extentions
- JEP 319 root certificates, to easy migration from Oracle JDK → OpenJDK
- removal of `policytool`

## local type inference

https://openjdk.java.net/jeps/286

This is the big new feature of Java 10.

```
HashMap<String, String> a = new HashMap<>();
var b = new HashMap<String, String>();
assert a.equals(b);
```

Often using `var` is quite convenient:

```
Map<String, Integer> map = Map.of("a", 1, "b", 2); ①
for (Map.Entry<String, Integer> entry : map.entrySet()) { ①
    System.out.println(entry);
}

var map2 = Map.copyOf(map); ②
for (var entry : map2.entrySet()) { ②
    System.out.println(entry);
}
```

① `Map<String, Integer>` is spread all over

② much more easy to read

It is important to know that `var` is not a keyword, it is a special type: in this way you can continue to

use var as variable name or method name (see section "3.9" of *The Java Language Specification, Java SE 11 Edition*):

```
jshell> var var = 1 ①
var ==> 1

jshell> class var { } ②
|  Error:
|  'var' not allowed here
|    as of release 10, 'var' is a restricted local variable type and cannot be used
for type declarations
|  class var { }
```

① still possible to use `var` as variable name

② not possible anymore to use `var` as `class` or `interface`

**NOTE** now it is possible to use anonymous types, this was not possible before:

```
var a = new Object() {
   void m() {
   }
};
a.m(); ①
```

① the type of `a` is `Object` + method `m()`

By using `var` wisely it could be possible to make easier to perform large scale refactorings, but this idea must be proven. Don't miss the Style Guidelines for Local Variable Type Inference in Java.

# Docker awareness

JVM now can detect CPU/memory settings when run inside a container. Given a docker setup with 4 CPUs and 2GB of memory:

```
dfa@aman ~ $ docker container run -it --rm --cpus 1 openjdk:9-jdk-slim ①
Sep 29, 2018 7:56:31 AM java.util.prefs.FileSystemPreferences$1 run
INFO: Created user preferences directory.
|  Welcome to JShell -- Version 9.0.4
|  For an introduction type: /help intro

jshell> Runtime.getRuntime().availableProcessors()
$1 ==> 4 ②
```

① 1 CPU requested

② 4 CPU available

Before this release JVM was not aware of this `--cpus 1`, so this is why JVM sees 4 CPUs. Whereas in

Java 10:

```
dfa@aman:~ $ docker container run -it --rm --cpus 1 openjdk:10-jdk-slim ①
Sep 29, 2018 8:11:29 AM java.util.prefs.FileSystemPreferences$1 run
INFO: Created user preferences directory.
|  Welcome to JShell -- Version 10.0.2
|  For an introduction type: /help intro

jshell> Runtime.getRuntime().availableProcessors()
$1 ==> 1 ②
```

① 1 CPU requested

② 1 CPU available

It is possible to use also a cpu-set:

```
dfa@aman:~ $ docker container run -it --rm --cpuset-cpus="1,2"  openjdk:10-jdk-slim
Sep 29, 2018 8:14:53 AM java.util.prefs.FileSystemPreferences$1 run
INFO: Created user preferences directory.
|  Welcome to JShell -- Version 10.0.2
|  For an introduction type: /help intro

jshell> Runtime.getRuntime().availableProcessors()
$1 ==> 2
```

Regarding memory setting, by default JVM uses 1/4 of the memory, 2 GB in the following example:

```
dfa@aman:~ $ docker container run -it --rm openjdk:10-jdk-slim
Sep 29, 2018 8:20:47 AM java.util.prefs.FileSystemPreferences$1 run
INFO: Created user preferences directory.
|  Welcome to JShell -- Version 10.0.2
|  For an introduction type: /help intro

jshell> Runtime.getRuntime().maxMemory() / 1024 / 1024
$2 ==> 500 ①
```

① the JVM sees 500MB

Without constraints JVM is going to use 1/4 of the available memory to docker, 500MB.

```
dfa@aman:~ $ docker container run -it --rm --memory 512M openjdk:10-jdk-slim ①
Sep 29, 2018 8:25:12 AM java.util.prefs.FileSystemPreferences$1 run
INFO: Created user preferences directory.
|  Welcome to JShell -- Version 10.0.2
|  For an introduction type: /help intro

jshell> Runtime.getRuntime().maxMemory() / 1024 / 1024
$1 ==> 123 ②
```

① requesting a constraint of memory

② the JVM sees 123MB

It is possible then to fine tune the memory settings by using JVM flags `-Xmx`, `-Xms`, etc.

### References

More on this can be found in the following tickets:

- improve docker container detection and resource configuration usage https://bugs.openjdk.java.net/browse/JDK-8146115
- allow more flexibility in selecting Heap % of available RAM https://bugs.openjdk.java.net/browse/JDK-8186248
- jcmd attach in linux should be relative to /proc/pid/root and namespace aware https://bugs.openjdk.java.net/browse/JDK-8179498

## new javadoc `@summary` tag

https://bugs.openjdk.java.net/browse/JDK-8173425

In order to be precise and avoid ambiguities around the special handling of the first sentence of javadoc, it is possible to use `@summary`:

```
{@summary This is the first sentence.} This is the second sentence.
```

## `java.io.Reader.transferTo(Writer)`

This is a long awaited feature, usually provided by external libraries such as Apache IOUtils.

```
jshell> import java.io.*

jshell> var a = new StringReader("hello world");
a ==> java.io.StringReader@3abbfa04

jshell> var b = new StringWriter();
b ==>

jshell> a.transferTo(b)
$4 ==> 11

jshell> b
b ==> hello world
```

## RuntimeMXBean.getPid()

This new method returns the process ID representing the running JVM:

```
jshell> import java.lang.management.*;

jshell> ManagementFactory.getRuntimeMXBean().getPid()
$8 ==> 11429
```

# Upgrade to Java 9

http://openjdk.java.net/projects/jdk9/

Java 9 delivers an impressive set of features, by far the most important are:

- Project Jigsaw aka modules https://openjdk.java.net/projects/jigsaw/
- JShell https://openjdk.java.net/jeps/222

Describing any of these items is beyond the scope of this document.

| NOTE | it is important to say that **you don't need modules to run on Java 9**, classpath is still supported. |
|---|---|

# Immutable collections

https://openjdk.java.net/jeps/269

JEP 269 introduced some new factory methods for collections. New APIs are:

```
Set<Integer> set = Set.of(1,2,3,4);
List<Integer> list = List.of(1,2,1,2);
Map<String, Integer> map = Map.of("key1", 1, "key2", 2);
Map<String, Integer> mapLonger = Map.ofEntries(Map.entry("key1", 1), Map.entry("key2",
2));
```

| NOTE | *Immutability vs views*<br>`Collections.unmodificable` are just read-only wrappers around original data structure: if the original data structure is mutable, you can still see changes in the wrapper. |
|---|---|

```
jshell> var list = new ArrayList<>()
list ==> []

jshell> var unmodificableList = Collections.unmodifiableList(list)
unmodificableList ==> []

jshell> list.add(1) ①
$3 ==> true

jshell> unmodificableList ①
unmodificableList ==> [1]
```

① changes to `unmodificableList` are still possible

By using these new methods you get immutable data structures:

```
jshell> var immutableList = List.of();
immutableList ==> []

jshell> immutableList.add(1)
|  Exception java.lang.UnsupportedOperationException
|        at ImmutableCollections.uoe (ImmutableCollections.java:71)
|        at ImmutableCollections$AbstractImmutableCollection.add (
ImmutableCollections.java:75)
|        at (#6:1)
```

**NOTE**

*Gotcha:*

sometimes `List.copyOf` is a no-op

```
jshell> List<?> a = List.of(1,2,3)
a ==> [1, 2, 3]

jshell> List<?> b = List.copyOf(a)
b ==> [1, 2, 3]

jshell> a == b
$7 ==> true
```

**NOTE**

*Gotcha:*

cannot use `Set.of` to filter away duplicated.

```
jshell> Set.of(1,1)
|  java.lang.IllegalArgumentException thrown: duplicate element: 1
|        at ImmutableCollections$SetN.<init> (ImmutableCollections.java:463)
|        at Set.of (Set.java:521)
```

Now it is also possible to use `java.util.stream.Collectors` to build an unmodificableList/Set/Map.

## java.util.Optional.stream()

Given:

```
List<Optional<String>> listOfOptionals = something();
```

in Java 8:

```
List<String> filteredList = listOfOptionals.stream()
  .filter(Optional::isPresent) ①
  .map(Optional::get) ②
  .collect(Collectors.toList());
```

① discarding empty optionals

② unwrapping optionals

in Java 9 it is possible to write:

```
List<String> filteredList = listOfOptionals.stream()
  .flatMap(Optional::stream) ①
  .collect(Collectors.toList());
```

① `Optional` as `Stream`

`Optional.stream()` implementation is straightforward:

```
public Stream<T> stream() {
    if (!isPresent()) {
        return Stream.empty();
    } else {
        return Stream.of(value);
    }
}
```

## `Stream.takeWhile()`/`Stream.dropWhile()`

https://bugs.openjdk.java.net/browse/JDK-8071597

Another nice addiction to the Stream API:

```
IntStream
  .iterate(1, n -> n + 1)
  .takeWhile(n -> n < 10)
  .forEach(System.out::println);
```

# Stack walk API

https://openjdk.java.net/jeps/259

This is useful in several occasions and now it is possible to capture a partial stacktrace in a very simple and effective way.

For example getting the caller it is trivial now:

```
StackWalker
    .getInstance(StackWalker.Option.RETAIN_CLASS_REFERENCE)
    .getCallerClass()
```

`StackWalker` instances are thread-safe and thus can be shared between threads: each thread will see its own stack. Additionally a security check is performed on creation of the `StackWalker` instance, no further checks are performed later.

# Milling Project Coin

https://openjdk.java.net/jeps/213

The small language changes included in Project Coin were low hanging fruits but neverthless the project was quite successful. Java 9 introduces the following small changes:

- `@SafeVarags` on private methods, https://bugs.openjdk.java.net/browse/JDK-7196160
- `private` methods in interfaces
- allow final or effectively final variables to be used as resources in try-with-resources https://bugs.openjdk.java.net/browse/JDK-7196163
- `_` identifier now is reserved

Let's quickly dig the last point since it is a trivial source-level incompatibility to fix:

```
jshell> Predicate<Integer> pred = _ -> false;
|  Error:
|  '_' used as an identifier
|    (use of '_' as an identifier is forbidden for lambda parameters)
|  Predicate<Integer> pred = _ -> false;
|                            ^
```

# Version

https://openjdk.java.net/jeps/223

In legacy code bases it is possible to find various ways to determine which java version is running:

```
String version = Runtime.class.getPackage().getImplementationVersion();
```

```
double version = Double.parseDouble(System.getProperty("java.specification.version"));
```

```
String[] javaVersionElements = System.getProperty("java.runtime.version").split("
\\.|_|-b");
```

Now we have a nice standard API to do that:

```
Runtime.Version version = Runtime.version();
```

`Version` it's a value object and it is `Comparable<Version>`. It is even possible to create a `Version` instance using an externally provided version string:

```
import java.lang.Runtime.Version;
import java.lang.System;

String versionString = System.getProperty("java.version");
Version version = Version.parse(versionString);
System.out.printf("java %d.%d%n", version.major(), version.minor());
```

## `java.lang.ProcessHandle`

Obtain information about JVM itself:

```
ProcessHandle self = ProcessHandle.current();
ProcessHandle.Info procInfo = self.info();
System.out.println(procInfo);
```

List all system processes:

```
import java.time.*;

ProcessHandle.allProcesses().map(ProcessHandle::info).forEach(System.out::println);
```

## `@Deprecated` enhancements

https://openjdk.java.net/jeps/277

It is possible to mark a method/class for removal (`forRemoval`) and when the deprecation started (`since`):

```
@Deprecate(forRemoval=true)
public void foo() {

}

@Deprecate(since="1.0")
public void bar() {

}
```

# Unified JVM logging

https://openjdk.java.net/jeps/158

This is invaluable in debugging problems in production problems,by by capture and log interesting events happening inside the JVM.

For example it is possible to include every tag around gc:

```
$ java -Xlog:gc* -jar myapp.jar
[0.012s][info][gc,heap] Heap region size: 1M
[0.018s][info][gc     ] Using G1
[0.019s][info][gc,heap,coops] Heap address: 0x00000006c0000000, size: 4096 MB,
Compressed Oops mode: Zero based, Oop shift amount: 3
... application output
... exit
[1.803s][info][gc,heap,exit ] Heap
[1.803s][info][gc,heap,exit ]  garbage-first heap   total 262144K, used 11264K
[0x00000006c0000000, 0x00000007c0000000)
[1.803s][info][gc,heap,exit ]   region size 1024K, 12 young (12288K), 0 survivors (0K)
[1.803s][info][gc,heap,exit ]  Metaspace       used 10805K, capacity 11186K, committed
11520K, reserved 1058816K
[1.803s][info][gc,heap,exit ]   class space    used 1042K, capacity 1203K, committed
1280K, reserved 1048576K
```

Or restrict to a more specific tag, such as gc+heap:

```
$ java -Xlog:gc,gc+heap -jar myapp.jar
[0.013s][info][gc,heap] Heap region size: 1M
[0.019s][info][gc     ] Using G1
... application output
[89.471s][info][gc,heap] GC(0) Eden regions: 24->0(151)
[89.471s][info][gc,heap] GC(0) Survivor regions: 0->2(3)
[89.471s][info][gc,heap] GC(0) Old regions: 0->0
[89.471s][info][gc,heap] GC(0) Humongous regions: 0->0
[89.471s][info][gc     ] GC(0) Pause Young (G1 Evacuation Pause) 24M->1M(256M) 3.926ms
... exit
```

# Compact Strings

https://openjdk.java.net/jeps/254

This is just a more efficient internal representation of `java.lang.String`, no public methods will be changed.

In practice, the internal representation of string has been changed from `char[]` to `byte[]` + flag. The purpose of the flag is to store which encoding to use:

- `ISO-8859-1/Latin-1` (one byte per character)
- `UTF-16` (two bytes per character).

# Appendix: Upgrade to Java 8

Since some projects and products are stuck in Java 7 let's quickly cover Java 8. The main features were:

- Project Lambda (lambda and streams) https://openjdk.java.net/projects/lambda/
- ThreeTen (new date/time library) https://openjdk.java.net/projects/threeten/

Nevertheless there are hidden gems in this release, that often are ignored.

## `java.io.UncheckedIOException`

This is a little know class that wraps an `IOException` with an unchecked exception.

## `StampedLocks`

A fast alternative to `ReadWriteLock`, that has an "optimistic" mode.

## Concurrent Adders

A `LongAdder` could be a great alternative to `AtomicLong` for high contention use cases.

## Strong algorithm for `SecureRandom`

A new API has been added:

```
public static SecureRandom getInstanceStrong()
                                     throws NoSuchAlgorithmException
```

## Overflow free operations

A great addition is a set of new methods to perform basic math operation, throwing exceptions when overflow are detected:

```
long a = Integer.MAX_VALUE * 2; ①
long b = Math.multiplyExact(Integer.MAX_VALUE, 2) ②
|  Exception java.lang.ArithmeticException: integer overflow
|        at Math.multiplyExact (Math.java:906)
|        at (#2:1)
```

① silent error: the result is -2, that is spectacularly wrong

② loud error

## String.join()

This is a static method on `java.lang.String`:

```
jshell> String.join(",");
$10 ==> ""

jshell> String.join(",", "a");
$11 ==> "a"

jshell> String.join(",", "a", "b");
$12 ==> "a,b"
```

There are two overloads: one for vararg array and another one for `Iterable<? extends CharSequence>`.