

# Java 11

Oracle released Java 11 in September 2018, only 6 months after its predecessor, version 10. Java 11 is the first long-term support (LTS) release after Java 8. Oracle also stopped supporting Java 8 in January 2019. As a consequence, a lot of us will upgrade to Java 11. Java 10 was the last free Oracle JDK release that we could use commercially without a license. Starting with Java 11, there's no free long-term support (LTS) from Oracle. Thankfully, Oracle continues to provide Open JDK releases, which we can download and use without charge.

## 1.. New String Methods

Java 11 adds a few new methods to the `String` class: **`isBlank`**, **`lines`**, **`strip`**, **`stripLeading`**, **`stripTrailing`**, and **`repeat`**. Let's see how we can make use of the new methods to extract non-blank, stripped lines from a multi-line string.

```
String multilineString = "Baeldung helps \n \n developers \n explore Java.";
List<String> lines = multilineString.lines()
    .filter(line -> !line.isBlank())
    .map(String::strip)
    .collect(Collectors.toList());
assertThat(lines).containsExactly("Baeldung helps", "developers", "explore Java.");
```

These methods can reduce the amount of boilerplate involved in manipulating string objects, and save us from having to import libraries. In the case of the `strip` methods, they provide similar functionality to the more familiar `trim` method; however, with finer control and Unicode support.

## 2.. New File Methods

Additionally, it's now easier to read and write Strings from files. We can use the new **`readString`** and **`writeString`** static methods from the `Files` class.

```
Path filePath = Files.writeString(Files.createTempFile(tempDir, "demo", ".txt"), "Sample text");
String fileContent = Files.readString(filePath);
assertThat(fileContent).isEqualTo("Sample text");
```

## 3.. Collection to an Array

The `java.util.Collection` interface contains a new default `toArray` method which takes an `IntFunction` argument. This makes it easier to create an array of the right type from a collection.

```
List sampleList = Arrays.asList("Java", "Kotlin");
String[] sampleArray = sampleList.toArray(String[]::new);
assertThat(sampleArray).containsExactly("Java", "Kotlin");
```

## 4.. Not Predicate Method

A static `not` method has been added to the `Predicate` interface. We can use it to negate an existing predicate, much like the `negate` method. While `not(isBlank)` reads more naturally than

**isBlank.negate()**, the big advantage is that we can also use not with method references, like `not(String::isBlank)`.

```
List<String> sampleList = Arrays.asList("Java", "\n\n", "Kotlin", " ");
List withoutBlanks = sampleList.stream()
    .filter(Predicate.not(String::isBlank))
    .collect(Collectors.toList());
assertThat(withoutBlanks).containsExactly("Java", "Kotlin");
```

## 5.. Local Variable Syntax for Lambda

Support for using the local variable syntax (`var` keyword) in lambda parameters was added in Java 11. We can make use of this feature to apply modifiers to our local variables, like defining a type annotation.

```
List<String> sampleList = Arrays.asList("Java", "Kotlin");
String resultString = sampleList.stream()
    .map((@NonNull var x) -> x.toUpperCase())
    .collect(Collectors.joining(", "));
assertThat(resultString).isEqualTo("JAVA, KOTLIN");
```

## 6.. HttpClient

The new HTTP client from the **java.net.http** package was introduced in Java 9. It has now become a standard feature in Java 11. The new HTTP API improves overall performance and provides support for both HTTP/1.1 and HTTP/2.

```
HttpClient httpClient = HttpClient.newBuilder()
    .version(HttpClient.Version.HTTP_2)
    .connectTimeout(Duration.ofSeconds(20))
    .build();
HttpRequest httpRequest = HttpRequest.newBuilder()
    .GET()
    .uri(URI.create("http://localhost:" + port))
    .build();
HttpResponse httpResponse = httpClient.send(httpRequest, HttpResponse.BodyHandlers.ofString());
assertThat(httpResponse.body()).isEqualTo("Hello from the server!");
```

## 7.. Nest Based Access Control

Java 11 introduces the notion of nestmates and the associated access rules within the JVM. A nest of classes in Java implies both the outer/main class and all its nested classes.

```
assertThat(MainClass.class.isNestmateOf(MainClass.NestedClass.class)).isTrue();
```

Nested classes are linked to the *NestMembers* attribute, while the outer class is linked to the *NestHost* attribute:

```
assertThat(MainClass.NestedClass.class.getNestHost()).isEqualTo(MainClass.class);
```

JVM access rules allow access to private members between nestmates; however, in previous Java versions, the reflection API denied the same access.

Java 11 fixes this issue and provides means to query the new class file attributes using the reflection API:

```
Set<String> nestedMembers = Arrays.stream(MainClass.NestedClass.class.getNestMembers())
    .map(Class::getName)
    .collect(Collectors.toSet());
assertThat(nestedMembers).contains(MainClass.class.getName(), MainClass.NestedClass.class.getName());
```

## 8.. Running Java Programs

A major change in this version is that we don't need to compile the Java source files with `javac` explicitly anymore.

```
$ javac HelloWorld.java
$ java HelloWorld
Hello Java 8!
```

Instead, we can directly run the file using the `java` command:

```
$ java HelloWorld.java
Hello Java 11!
```

## 9.. Performance Enhancements

Now let's take a look at a couple of new features whose main purpose is improving performance.

### 9.1 Dynamic Class-File Constants

Java class-file format is extended to support a new constant-pool form named **CONSTANT\_Dynamic**. Loading the new constant-pool will delegate creation to a bootstrap method, just as linking an **invokedynamic** call site delegates linkage to a bootstrap method. This feature enhances performance and targets language designers and compiler implementors.

### 9.2 Improved Aarch64 Intrinsics

Java 11 optimizes the existing string and array intrinsics on ARM64 or AArch64 processors. Additionally, new intrinsics are implemented for `sin`, `cos`, and `log` methods of `java.lang.Math`. We use an intrinsic function like any other; however, the intrinsic function gets handled in a special way by the compiler. It leverages CPU architecture-specific assembly code to boost performance.

### 9.3 A No-Op Garbage Collector

A new garbage collector called **Epsilon** is available for use in Java 11 as an experimental feature. It's called a No-Op (no operations) because it allocates memory but does not actually collect any garbage. Thus, Epsilon is applicable for simulating out of memory errors. Obviously, Epsilon won't be suitable for a typical production Java application; however, there are a few specific use-cases where it could be useful:

- Performance testing
- Memory pressure testing
- VM interface testing and
- Extremely short-lived jobs

In order to enable it, use the `-XX:+UnlockExperimentalVMOptions -XX:+UseEpsilonGC` flag.

### 9.4 Flight Recorder

Java Flight Recorder (JFR) is now open-source in Open JDK, whereas it used to be a commercial product in Oracle JDK. JFR is a profiling tool that we can use to gather diagnostics and profiling data from a running Java application. To start a 120 seconds JFR recording, we can use the following parameter:

```
-XX:StartFlightRecording=duration=120s,settings=profile,filename=java-demo-app.jfr
```

We can use JFR in production since its performance overhead is usually below 1%. Once the time elapses, we can access the recorded data saved in a JFR file; however, in order to analyze and visualize the data, we need to make use of another tool called JDK Mission Control (JMC).

## 10.. Removed and Deprecated Modules

As Java evolves, we can no longer use any of its removed features and should stop using any deprecated features. Let's take a quick look at the most notable ones.

### 10.1 Java EE and CORBA

Standalone versions of the Java EE technologies are available on third-party sites; therefore, there is no need for Java SE to include them. Java 9 already deprecated selected Java EE and CORBA modules. In release 11, it has now completely removed.

- Java API for XML-Based Web Services (java.xml.ws)
- Java Architecture for XML Binding (java.xml.bind)
- JavaBeans Activation Framework (java.activation)
- Common Annotations (java.xml.ws.annotation)
- Common Object Request Broker Architecture (java.corba)
- JavaTransaction API (java.transaction)

### 10.2 JMC and JavaFX

JDK Mission Control (JMC) is no longer included in the JDK. A standalone version of JMC is now available as a separate download. The same is true for JavaFX modules; JavaFX will be available as a separate set of modules outside of the JDK.

### 10.3 Deprecated Modules

Furthermore, Java 11 deprecated the following modules:

- Nashorn JavaScript engine, including the JJS tool
- Pack200 compression scheme for JAR files

## 11.. Miscellaneous Changes

Java 11 introduced a few more changes that are important to mention:

- New ChaCha20 and ChaCha20-Poly1305 cipher implementations replace the insecure RC4 stream cipher
- Support for cryptographic key agreement with Curve25519 and Curve448 replace the existing ECDH scheme
- Upgraded Transport Layer Security (TLS) to version 1.3 brings security and performance improvements
- Introduced a low latency garbage collector, ZGC, as an experimental feature with low pause times
- Support for Unicode 10 brings more characters, symbols, and emojis