**Why is Java 11 important?**

Java 11 is the second LTS release after [Java 8](https://www.journaldev.com/2389/java-8-features-with-examples). Since Java 11, Oracle JDK would no longer be free for commercial use.

You can use it in developing stages but to use it commercially, you need to buy a license. If you don’t, you can get an invoice bill from Oracle any day!

[Java 10](https://www.journaldev.com/20395/java-10-features) was the last free Oracle JDK that could be downloaded.

Oracle stops [Java 8](https://www.journaldev.com/2389/java-8-features-with-examples) support from January 2019. You’ll need to pay for more support.  
You can continue using it, but won’t get any patches/security updates

While Oracle JDK is no longer free, you can always download the Open JDK builds from Oracle or other providers such as AdoptOpenJDK, Azul, IBM, Red Hat etc. In my opinion, unless you are looking for Enterprise level usage with the appetite to pay for the support fees, you can use [OpenJDK](http://OpenJDK) and upgrade them as and when necessary.

## Which JDK build should I download and what are the benefits of each of them?

Since Oracle has created a release train in which a new version would come up every six months, if you are using the free Open JDK by Oracle, you will need to update it every six months, since Oracle won’t provide free updates once the new version is released. This can be challenging to adapt to a company.

Pay for commercial support to Oracle and migrate only from one LTS version to the next LTS version.  
This way you’ll get all the updates and support for Java 11 till 2026. You can download Java 17 in 2022.

Stay on free Java version even after its support ends. Though you won’t get security updates and it can open up security loopholes.

Oracle won’t provide commercial support or updates for [Java 9](https://www.journaldev.com/13121/java-9-features-with-examples) and Java 10. You need to look for other alternative builds in order to keep using them for free.

Having understood the baggage Java 11 comes with, lets now analyze the important features in Java 11 for developers. We’ll discuss some important JEPs too.

## Java 11 Features

Let’s discuss the new features introduced with Java 11 from the JEP Process.

### Running Java File with single command

One major change is that you don’t need to compile the java source file with javac tool first. You can directly run the file with **java** command and it implicitly compiles.  
This feature comes under JEP 330.

Following is a sneak peek at the new methods of [Java String](https://www.journaldev.com/16928/java-string) class introduced in Java 11:

### Java String Methods

**isBlank()** – This instance method returns a boolean value. Empty Strings and Strings with only white spaces are treated as blank.

import java.util.\*;

public class Main {

public static void main(String[] args) throws Exception {

// Your code here!

System.out.println(" ".isBlank()); //true

String s = "Anupam";

System.out.println(s.isBlank()); //false

String s1 = "";

System.out.println(s1.isBlank()); //true

}

}

**lines()**

This method returns a string array which is a collection of all substrings split by lines.

import java.util.stream.Collectors;

public class Main {

public static void main(String[] args) throws Exception {

String str = "JD\nJD\nJD";

System.out.println(str);

System.out.println(str.lines().collect(Collectors.toList()));

}

}

The output of the above code is:

[](https://cdn.journaldev.com/wp-content/uploads/2019/01/java-11-lines-method-output.png)

**strip(), stripLeading(), stripTrailing()**

strip() – Removes the white space from both, beginning and the end of string.

**But we already have trim(). Then what’s the need of strip()?**

strip() is “Unicode-aware” evolution of trim().

When trim() was introduced, Unicode wasn’t evolved. Now, the new strip() removes all kinds of whitespaces leading and trailing(check the method Character.isWhitespace(c) to know if a unicode is whitespace or not)

An example using the above three methods is given below:

public class Main {

public static void main(String[] args) throws Exception {

// Your code here!

String str = " JD ";

System.out.print("Start");

System.out.print(str.strip());

System.out.println("End");

System.out.print("Start");

System.out.print(str.stripLeading());

System.out.println("End");

System.out.print("Start");

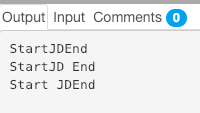
System.out.print(str.stripTrailing());

System.out.println("End");

}

}

The output in the console from the above code is:

[](https://cdn.journaldev.com/wp-content/uploads/2019/01/java-11-strip-method-output.png)

**repeat(int)**

The repeat method simply repeats the string that many numbers of times as mentioned in the method in the form of an int.

public class Main {

public static void main(String[] args) throws Exception {

// Your code here!

String str = "=".repeat(2);

System.out.println(str); //prints ==

}

}

### Local-Variable Syntax for Lambda Parameters

**JEP 323**, Local-Variable Syntax for [Lambda](https://www.journaldev.com/16703/java-lambda-expression) Parameters is the only language feature release in Java 11.  
In Java 10, [Local Variable Type Inference](https://www.journaldev.com/19871/java-10-local-variable-type-inference) was introduced. Thus we could infer the type of the variable from the RHS – var list = new ArrayList<String>();

JEP 323 allows var to be used to declare the formal parameters of an implicitly typed lambda expression.

We can now define :

(var s1, var s2) -> s1 + s2

This was possible in Java 8 too but got removed in Java 10. Now it’s back in Java 11 to keep things uniform.

**But why is this needed when we can just skip the type in the lambda?**  
If you need to apply an annotation just as @Nullable, you cannot do that without defining the type.

**Limitation of this feature** – You must specify the type var on all parameters or none.  
Things like the following are not possible:

(var s1, s2) -> s1 + s2 //no skipping allowed

(var s1, String y) -> s1 + y //no mixing allowed

var s1 -> s1 //not allowed. Need parentheses if you use var in lambda.

### Nested Based Access Control

Before Java 11 this was possible:

public class Main {

public void myPublic() {

}

private void myPrivate() {

}

class Nested {

public void nestedPublic() {

myPrivate();

}

}

}

private method of the main class is accessible from the above-nested class in the above manner.  
But if we use [Java Reflection](https://www.journaldev.com/1789/java-reflection-example-tutorial), it will give an IllegalStateException.

Method method = ob.getClass().getDeclaredMethod("myPrivate");

method.invoke(ob);

Java 11 nested access control addresses this concern in reflection.  
java.lang.Class introduces three methods in the reflection API: getNestHost(), getNestMembers(), and isNestmateOf().

### JEP 309: Dynamic Class-File Constants

The Java class-file format now extends support a new constant pool form, CONSTANT\_Dynamic. The goal of this JEP is to reduce the cost and disruption of developing new forms of materializable class-file constraints, by creating a single new constant-pool form that can be parameterized with user-provided behavior.  
This enhances performance

### JEP 318: Epsilon: A No-Op Garbage Collector

Unlike the JVM GC which is responsible for allocating memory and releasing it, Epsilon only allocates memory.  
It allocates memory for the following things:

* Performance testing.
* Memory pressure testing.
* VM interface testing.
* Extremely short lived jobs.
* Last-drop latency improvements.
* Last-drop throughput improvements.

Now Elipson is good only for test environments. It will lead to [OutOfMemoryError](https://www.journaldev.com/21010/java-lang-outofmemoryerror-java-heap-space) in production and crash the applications.

The benefit of Elipson is no memory clearance overhead. Hence it’ll give an accurate test result of performance and we can no longer GC for stopping it.

Note: This is an experimental feature.

### JEP 320: Remove the Java EE and CORBA Modules

The modules were already deprecated in Java 9. They are now completely removed.

Following packages are removed: java.xml.ws, java.xml.bind, java.activation, java.xml.ws.annotation, java.corba, java.transaction, java.se.ee, jdk.xml.ws, jdk.xml.bind

### JEP 328: Flight Recorder

Flight Recorder which earlier used to be a commercial add-on in Oracle JDK is now open sourced since Oracle JDK is itself not free anymore.

JFR is a profiling tool used to gather diagnostics and profiling data from a running Java application.  
Its performance overhead is negligible and that’s usually below 1%. Hence it can be used in production applications.

### JEP 321: HTTP Client

Java 11 standardizes the Http CLient API.  
The new API supports both HTTP/1.1 and HTTP/2. It is designed to improve the overall performance of sending requests by a client and receiving responses from the server. It also natively supports WebSockets.

### Reading/Writing Strings to and from the Files

Java 11 strives to make reading and writing of String convenient.  
It has introduced the following methods for reading and writing to/from the files:

* readString()
* writeString()

Following code showcases an example of this

Path path = Files.writeString(Files.createTempFile("test", ".txt"), "This was posted on JD");

System.out.println(path);

String s = Files.readString(path);

System.out.println(s); //This was posted on JD

### JEP 329: ChaCha20 and Poly1305 Cryptographic Algorithms

Java 11 provides ChaCha20 and ChaCha20-Poly1305 Cipher implementations. These algorithms will be implemented in the SunJCE provider.

### JEP 315: Improve Aarch64 Intrinsics

Improve the existing string and array intrinsics, and implement new intrinsics for the java.lang.Math sin, cos, and log functions, on AArch64 processors.

### JEP 333: ZGC: A Scalable Low-Latency Garbage Collector (Experimental)

Java 11 has introduced a low latency GC. This is an experimental feature.  
It’s good to see that Oracle is giving importance to GC’s.

### JEP 335: Deprecate the Nashorn JavaScript Engine

Nashorn JavaScript script engine and APIs are deprecated thereby indicating that they will be removed in the subsequent releases.