REASONS WHY ONE SHOULD UPGRADE FROM JAVA 8 TO JAVA 11

Andreas Kreouzos

1. More garbage collector options for improved Garbage Collection.

G1GC serves as the default garbage collector in Java 11. To enable the G1GC, we need to run the following argument: `java -XX:+UseG1GC -jar Application.java`

More options are available such as Epsilon & ZGC.

Epsilon is a Garbage Collector that handles memory allocation but doesn't reclaim any memory. Hence once the overall java heap is exhausted it will shutdown the JVM throwing an OutOfMemoryError. Epsilon is useful for short-lived services and for applications that are known to be garbage-free.To enable the Epsilon GC, we need to pass the following VM arguments: `-XX:+UnlockExperimentalVMOptions -XX:+UseEpsilonGC`

Z Garbage Collector **ZGC** serves as great alternative because it's a concurrent collector that attempts to keep pause times under 10ms. It is **suitable for applications which require low latency and/or use a very large heap**. To enable the Z Garbage Collector, we can use the following argument in JDK versions lower than 15: `java -XX:+UnlockExperimentalVMOptions -XX:+UseZGC Application.java`.

Example Using the Epsilon GC

```
class GarbageMemoryEpsilon{
    static final int MEGABYTE_IN_BYTES = 1024 * 1024;
    static final int ITERATION_COUNT = 1024 * 10;
    static void main(String[] args) {
        System.out.println("Starting pollution");
        for (int i = 0; i < ITERATION_COUNT; i++) {
            byte[] array = new byte[MEGABYTE_IN_BYTES];
        }
        System.out.println("Terminating");
    }
}
```

The above code snippet creates one-megabyte-arrays in a loop. Since we repeat the loop 10240 times, it means we allocate 10 gigabytes of memory, which is probably higher than the available maximum heap size.

```
`Starting pollution`
`Terminating`
```

BUT when we run the application with **Epsilon GC VM**

```
`Starting pollution`
```

results in

```
`Terminating due to java.lang.OutOfMemoryError: Java heap space`
```

2. Java upgraded to support TLS 1.3, which is more secure than its predecessors.

TLS (Transport Layer Security)

security protocol designed for Internet communication to enhance privacy and data security. It is generally used to encrypt communication among web packages and servers, including a web browser loading page.

3. Java 11 implements more secure ciphers ChaCha20 and ChaCha20-Poly1305.

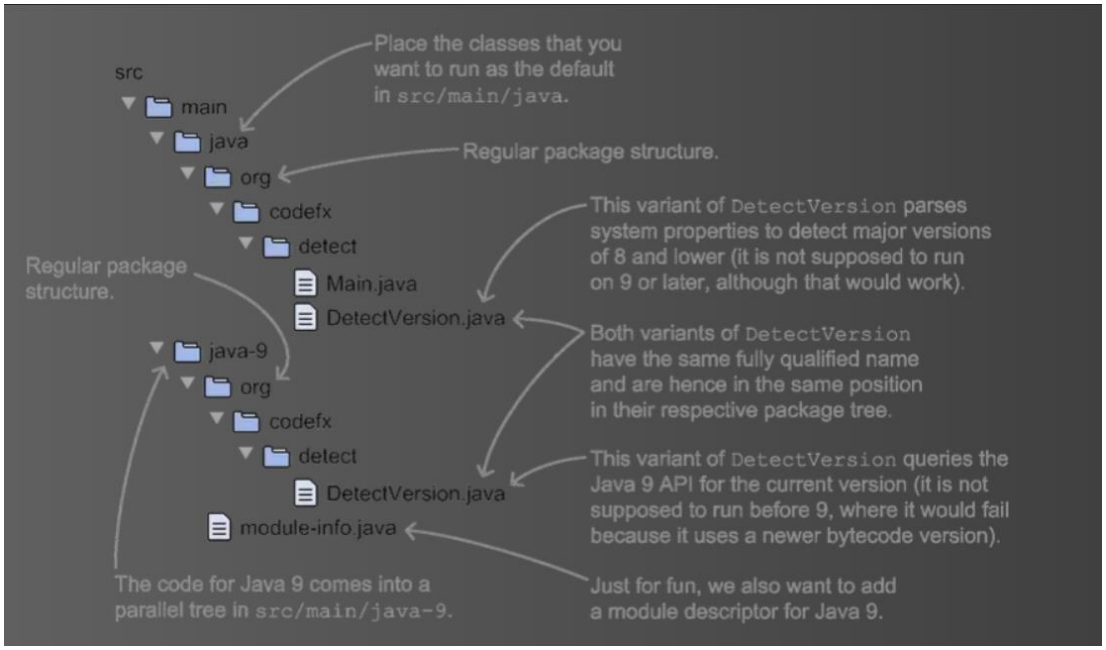
Poly1305 is a universal hash family and can be used as a one-time message authentication code to authenticate a single message using a key shared between sender and recipient.

ChaCha20-Poly1305 is an authenticated encryption with additional data (AEAD) algorithm, that combines the ChaCha20 stream cipher with the Poly1305 message authentication code.

4. Multi-JAR Releases.

Starting from Java 9, it is possible to create a jar file that contains multiple, Java-release-specific versions of class files.

- Multi-release jar files make it possible for library developers to support multiple versions of Java without having to ship multiple versions of jar files.
- For the consumer of these libraries, multi-release jar files solves the issue of having to match specific jar files to specific runtime targets.
- To properly use the Multi-release JAR feature inside of our project, two variants of the same classes we want to work with, should be created. One for Java 8 and earlier and another for Java 9 and later. These two variants need to have the exact same fully-qualified name, which makes it challenging to work with them in the IDE of our choice. For better ease, we can organize them into two parallel source folders, src/main/java and src/main/java-9.



Below we can see the compile process and packaging into MR-JAR:

```
compile code in `src/main/java` for Java 8 into `classes`
$ javac --release 8
    -d classes
    src/main/java/org/codefx/detect/*.java

compile code in `src/main/java-9` for Java 9 into `classes-9`
$ javac --release 9
    -d classes-9
    src/main/java-9/module-info.java
    src/main/java-9/org/codefx/detect/DetectVersion.java

$ jar --create
    --file target/detect.jar
    -C classes .
    --release 9
    -C classes-9 .
```

By running the resulting JAR on JVMs version 8 and 9, we can observe that, depending on the version, a different class is loaded.

5. Since applets have always been a source of security problems, Java 9 deprecated them. Code related to them was completely removed in Java 11.

Java applets are used **to provide interactive features to web applications** and can be executed by browsers for many platforms. They are small, portable Java programs embedded in HTML pages and can run automatically when the pages are viewed. Malware authors have used Java applets as a vehicle for attack.

6. The new heap profiler and low overhead flight recorder in Java 11 are examples of new low overhead flight recorders.

(JFR) is a tool for **collecting diagnostic and profiling data about a running Java application**. It is integrated into the Java Virtual Machine (JVM) and causes almost no performance overhead, so it can be used even in heavily loaded production environments.

7. It permitted the execution of a single Java source file.

you can execute a script containing Java code directly without compilation.

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

Instead of first compile it and then

```
$ javac HelloWorld.javaCopy
```

This would generate a class file that **we would have to run using the command:**

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

8. CORBA and Java EE modules have been removed from Java 11 as out-of-date and security risks.

The Java EE modules are a web services stack that have been available in the JDK since the release of Java SE 6 as a convenience for Java developers:

- **JAX-WS:** Java API for XML-Based Web Services (JSR-224)
- **JAXB:** Java Architecture for XML Binding (JSR-222)
- **JAF:** JavaBeans activation Framework (JSR-925)
- Common Annotations for the Java Platform (JSR-250)

It was determined that the JDK no longer needed to support these modules as they have evolved over the past 12 years and are readily available in third-party sites (such as Maven Central).

List of module-level APIs being removed in Java 11:

Name	Module	Description
Java API for XML Web Services (JAX-WS)	java.xml.ws	Defines the Java API for XML-Based Web Services (JAX-WS), and the Web Services Metadata API.
jdk.xml.ws	Tools for JAX-WS	
Java Architecture for XML Binding (JAXB)	java.xml.bind	Defines the Java Architecture for XML Binding (JAXB) API.

Name	Module	Description
jdk.xml.bind	Tools for JAXB	
JavaBeans Activation Framework (JAF)	java.activation	Defines the JavaBeans Activation Framework (JAF) API.
Common Annotations	java.xml.ws.annotation	Defines a subset of the Common Annotations API to support programs running on the Java SE Platform.
Common Object Request Broker Architecture (CORBA)	java.corba	Defines the Java binding of the OMG CORBA APIs, and the RMI-IIOP API.
Java Transaction API (JTA)	java.transaction	Defines a subset of the Java Transaction API (JTA) to support CORBA interoperation.
Aggregator module for the all modules above	java.se.ee	Defines the full API of the Java SE Platform.

Removing these modules is not without risks

https://www.dariawan.com/tutorials/java/java-11-remove-java-ee-and-corba-modules-jep-320/

9. The var keyword is also included in the language, making for a simpler development process.

By **only affecting local variables**, the Type Inference stops you repeating the same text over and over again.

10. New String methods

`repeat(int)` - Repeats the String as many times as provided by the `int` parameter

`lines()` - Uses a Spliterator to lazily provide lines from the source string

`isBlank()` - Indicates if the String is empty or contains only white space characters

`stripLeading()` - Removes the white space from the beginning

`stripTrailing()` - Removes the white space from the end

`strip()` - Removes the white space from both, beginning and the end of string

```
String example11 = "   Hello!   ";
example11.repeat(2) -->    Hello      Hello
example11.isBlank()-->false
example11.strip()-->Hello ( difference with trim() is that strip
considers \u0020 as whitespace )
```

```
example11.stripTrailing()-->   Hello

// lines
String example11   = " Hello \n Java \n 11 ";

// Generating stream of lines from string
Stream<String> lines = str.lines();

lines.forEach(System.out::println);
 Hello
 Java
 11
```

11. New HTTP Client API that implements HTTP/2 and Web Socket.

Problems With the Pre-Java 11 HTTP Client

The existing *HttpURLConnection* API and its implementation had numerous problems:

- URLConnection API was designed with multiple protocols that are now no longer functioning (FTP, gopher, etc.).
- The API predates HTTP/1.1 and is too abstract.
- It works in blocking mode only (i.e., one thread per request/response).
- It is very hard to maintain.

Unlike *HttpURLConnection*, HTTP Client provides synchronous and asynchronous request mechanisms.

The API consists of three core classes:

- *HttpRequest* represents the request to be sent via the *HttpClient*.
- *HttpClient* behaves as a container for configuration information common to multiple requests.
- *HttpResponse* represents the result of an *HttpRequest* call.

12. Modularity

A Java module is a packaging mechanism that enables you to package a Java application or Java API as a separate Java module. The latter is packaged as a modular JAR file.

- It creates structure inside large software systems by allowing them to be decomposed into smaller and more manageable pieces. The absence of modularity makes a system monolithic.
- It can also specify which of the Java packages it contains that should be visible to other Java modules which uses this module.
- A Java module must also specify which other Java modules are required to do its job.

Benefits of modularity

- Smaller Application Distributables via the Modular Java Platform: The benefit of splitting all the Java APIs up into modules is that you can now specify what modules of the Java platform your application requires.
- Encapsulation of Internal Packages: A Java module must explicitly tell which Java packages inside the module are to be exported (visible) to other Java modules using the module.
- Startup Detection of Missing Modules: From Java 9 and forward, Java applications must be packaged as Java modules too. Therefore an application module specifies what other modules it uses. Then the JVM can check the whole module dependency from the application module and forward, when the JVM starts up. If any required modules are not found at startup, the JVM reports the missing module and shuts down.

13. Improvements for Docker Containers

Prior to Java 10, memory and CPU constraints set on a container were not recognized by the JVM. In Java 8, for example, the JVM will default the maximum heap size to ¼ of the physical memory of the underlying host. Starting with Java 10, the JVM uses constraints set by container control groups (cgroups) to set memory and CPU limits. For example, the default maximum heap size is ¼ of the container's memory limit (e.g., 500MB for -m2G).

14. Oracle does not have any more updates for Java 8 and does not provide security updates for it anymore. However, Oracle still supports Java 11. All of these reasons make it desirable to move from Java 8 to 11.

We can still use Java 8 with support from Oracle but under a fee or use it without support which of course isn't recommended.