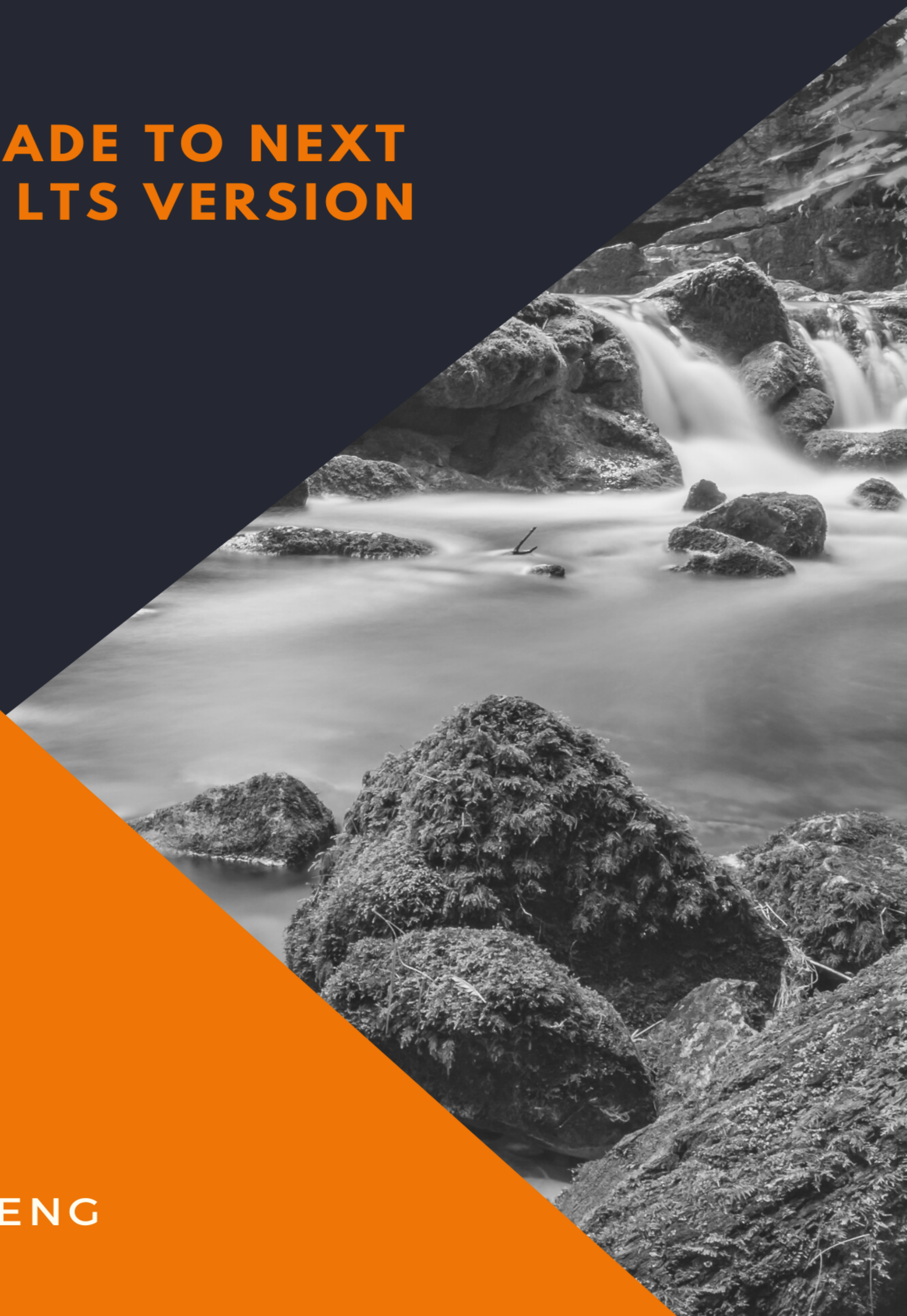
FROM JAVA 11 TO JAVA 17

UPGRADE TO NEXT JAVA LTS VERSION



FU CHENG

From Java 11 to Java 17

Upgrade to Java 17 LTS from Java 11 LTS

Fu Cheng

This book is available at https://leanpub.com/java11to17

This version was published on 2025-12-08

Leanpub

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Also By Fu Cheng

Exploring Java 25
Text-to-SQL, Spring AI Implementation with RAG
Understanding Java Virtual Threads
From Java 21 to Java 25
Build AI Applications with Spring AI
From Java 17 to Java 21
Build Native Java Apps with GraalVM
ES6 Generators
A Practical Guide for Java 8 Lambdas and Streams
Lodash 4 Cookbook
JUnit 5 Cookbook

Contents

Introduction

Java 17 was released on September 14, 2021. Java 17 is the next LTS version after Java 11. It's expected that users of Java 11 will gradually migrate to Java 17. This book summaries all major changes from Java 11 to Java 17, so you can easily migrate from Java 11 to Java 17.

Spring Framework 6 is released with requirements of JDK 17+, so you have to upgrade to JDK 17 to use Spring Framework 6.

If you are currently using Java 8, then you may need to know how to upgrade from Java 8 to Java 11 first. You can check out my book **Exploring Java 9** for changes in Java 9.

If you want to upgrade from Java 17 to Java 21, you can check out my book **From Java 17 to Java 21**.

Java Releases Schedule After Java 9

After Java 9, the Java community has adopted a new release mode. There will be a new Java release every six months, which means two releases per year. One version will be released in March, while the other version will be released in September. For example, Java 16 was released in March 2021, while Java 17 was released in September 2021.

For each release, there will be only six months support, until the next version is released. During these six months, there will be two quarterly security updates. For example, Java 15 only has the initially released version `15.0.0` and two security updates `15.0.1` and `15.0.2`. After Java 16 is released, there will be no more updates for Java 15. You have to upgrade to Java 16 to get updates.

For most projects, it's not practical to frequently update Java versions. Java also has LTS (Long-term support) versions. When using LTS versions, you can get updates for a long time. Java 8 and Java 11 are both LTS versions. Java 17 will be the next LTS version. For Java 11, you can get free public updates until at least 2024. Some companies offer free public updates for an extended period. So for most projects, LTS versions should be used.

JEPs in Different Java Releases

Java Enhancement Proposal (JEP) is used to organize changes to OpenJDK. A Java release may contain a list of JEPs. Below are lists of JEPs added in each Java release from Java 12 to Java 17.

Java 12

Java 12 has 8 JEPs, see the table below.

Figure 1. JEPs in Java 12

Number	Name
189	Shenandoah: A Low-Pause-Time Garbage Collector (Experimental)
230	Microbenchmark Suite
325	Switch Expressions (Preview)
334	JVM Constants API
340	One AArch64 Port, Not Two
341	Default CDS Archives
344	Abortable Mixed Collections for G1
346	Promptly Return Unused Committed Memory from G1

Java 13

Java 13 has 5 JEPs, see the table below.

Figure 2. JEPs in Java 13

Number	Name
350	Dynamic CDS Archives
351	ZGC: Uncommit Unused Memory
353	Reimplement the Legacy Socket API
354	Switch Expressions (Preview)
355	Text Blocks (Preview)

Java 14

Java 14 has 16 JEPs, see the table below.

Figure 3. JEPs in Java 14

Number	Name
305	Pattern Matching for instanceof (Preview)
343	Packaging Tool (Incubator)
345	NUMA-Aware Memory Allocation for G1
349	JFR Event Streaming
352	Non-Volatile Mapped Byte Buffers
358	Helpful NullPointerExceptions
359	Records (Preview)
361	Switch Expressions (Standard)
362	Deprecate the Solaris and SPARC Ports
363	Remove the Concurrent Mark Sweep (CMS) Garbage Collector
364	ZGC on macOS
365	ZGC on Windows
366	Deprecate the ParallelScavenge + SerialOld GC Combination
367	Remove the Pack200 Tools and API
368	Text Blocks (Second Preview)
370	Foreign-Memory Access API (Incubator)

Java 15

Java 15 has 14 JEPs, see the table below.

Figure 4. JEPs in Java 15

Number	Name
339	Edwards-Curve Digital Signature Algorithm (EdDSA)
360	Sealed Classes (Preview)
371	Hidden Classes
372	Remove the Nashorn JavaScript Engine
373	Reimplement the Legacy DatagramSocket API
374	Disable and Deprecate Biased Locking
375	Pattern Matching for instanceof (Second Preview)
377	ZGC: A Scalable Low-Latency Garbage Collector
378	Text Blocks
379	Shenandoah: A Low-Pause-Time Garbage Collector
381	Remove the Solaris and SPARC Ports
383	Foreign-Memory Access API (Second Incubator)
384	Records (Second Preview)
385	Deprecate RMI Activation for Removal

Java 16

Java 16 has 17 JEPs, see the table below.

Figure 5. JEPs in Java 16

Number	Name
338	Vector API (Incubator)
347	Enable C++14 Language Features
357	Migrate from Mercurial to Git
369	Migrate to GitHub
376	ZGC: Concurrent Thread-Stack Processing
380	Unix-Domain Socket Channels
386	Alpine Linux Port
387	Elastic Metaspace
388	Windows/AArch64 Port
389	Foreign Linker API (Incubator)
390	Warnings for Value-Based Classes
392	Packaging Tool
393	Foreign-Memory Access API (Third Incubator)
394	Pattern Matching for instanceof
395	Records
396	Strongly Encapsulate JDK Internals by Default
397	Sealed Classes (Second Preview)

Java 17

Java 17 has 14 JEPs, see the table below.

Figure 6. JEPs in Java 17

Number	Name
306	Restore Always-Strict Floating-Point Semantics
356	Enhanced Pseudo-Random Number Generators
382	New macOS Rendering Pipeline
391	macOS/AArch64 Port
398	Deprecate the Applet API for Removal
403	Strongly Encapsulate JDK Internals
406	Pattern Matching for switch (Preview)
407	Remove RMI Activation
409	Sealed Classes

Number	Name
410	Remove the Experimental AOT and JIT Compiler
411	Deprecate the Security Manager for Removal
412	Foreign Function & Memory API (Incubator)
414	Vector API (Second Incubator)
415	Context-Specific Deserialization Filters

Install Java 17

You can get Java 17 binaries from several different places. Since Java 17 is a LTS version, many vendors will provide Java 17 releases. For most users, Eclipse Temurin is the best choice as it's not vendor-specific.

- Download OpenJDK 17 build from jdk.java.net.
- Download Eclipse Temurin from Adoptium.
- Use SDKMAN to install, e.g. `sdk install java 17.0.16-amzn` or `sdk install java 17.0.16-tem`.
- Use IDE to download. IntelliJ IDEA can download JDK.

Below is the output of `java -version` for Eclipse Temurin 17.

Figure 7. Output of java -version

```
openjdk version "17.0.13" 2024-10-15
OpenJDK Runtime Environment Temurin-17.0.13+11 (build 17.0.13+11)
OpenJDK 64-Bit Server VM Temurin-17.0.13+11 (build 17.0.13+11, mixed mode, sh\
aring)
```

Build Tools

When using Java 17 with build tools, Maven or Gradle, you may need to add the `--enable-preview` argument to enable preview features. The following code shows an example of the configuration of Maven compiler plugin. `jdk.incubator.foreign` is an incubating module, so it needs to be added explicitly using `--add-modules`.

Figure 8. Maven compiler plugin configuration

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>3.13.0</version>
    <configuration>
        <source>17</source>
        <target>17</target>
        <compilerArgs>
            <arg>--enable-preview</arg>
            <arg>--add-modules</arg>
            <arg>jdk.incubator.foreign</arg>
        </compilerArgs>
    </configuration>
</plugin>
```

Source Code

Source code of this book can be found on GitHub.

Records

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Value Objects

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Alternatives

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Lombok

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Kotlin Data Class

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Scala Case Class

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Use Records

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Constructors

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Nested Records

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Local Records

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Record class

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Sealed Classes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Inheritance in Java

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Sealed Classes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Sealed Interfaces

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Sealed and Record Classes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Reflection API

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Class File Changes

Switch Expressions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Problems with switch Statement

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Switch Expression

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Arrow Labels

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Pattern Matching

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Pattern Matching for `instanceof`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Pattern Matching for `switch`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Text Blocks

Text Blocks were introduced in Java 12 as a preview feature and became a standard feature in Java 14. The following table shows the brief history of text blocks in Java.

Figure 9. History of text blocks

Java Release	Status	JEP
Java 12	Preview	JEP 355
Java 13	Second Preview	JEP 368
Java 14	Standard	JEP 378

It's a common requirement to embed multi-line string literals in Java source code. A typical example is to generate JSON/YAML/XML content based on string templates. Before Java 12, we had to use string concatenation to write multi-line strings in Java.

The following code shows how to use string concatenation to generate XML strings. Multi-line strings like this are hard to maintain. If you want to add double quotes in the string, these double quotes need to be escaped.

Figure 10. String templates using concatenation

```
import io.vividcode.java11to17.textblock.User;

public class StringTemplate {

  public String generate(User user) {
    return "<user>\n"
        + "  <id>" + user.id() + "</id>\n"
        + "  <name>" + user.name() + "</name>\n"
        + "  <email>" + user.email() + "</email>\n"
        + "</user>";
  }
}
```

Text Blocks

Text blocks are multi-line string literals delimited by three double quotes. The following code shows two examples of text blocks. Double quotes don't need to be escaped. Line terminators are kept.

Figure 11. String templates using text blocks

```
public class SimpleTextBlocks {

  private static final String BLOCK1 = """
      <user>
        <id>001</id>
        <name>alex</name>
        <email>alex@example.com</email>
      </user>
      """;

  private static final String BLOCK2 = """
      {
        "id": "001",
        "name": "alex",
        "email": "alex@example.com"
      }
      """;
}
```

Text blocks are processed by the Java compiler. In the runtime, text blocks are indistinguishable from normal single-line strings. Java compiler processes a text block in three steps:

1) Translate line terminators in the content to `LF`.
2) Remove incidental white space surrounding the content.
3) Interpret escape sequences in the content.

The first step is straight-forward. Java compiler normalizes `CR` and `CRLF` to `LF`.

Re-indentation

White spaces may play an important role in multi-line strings. For XML and JSON content, white spaces can improve readability. For YAML content, white spaces are significant for the data structure. When text blocks are embedded in the source code, their indentation needs to match Java source code's format.

In the code below, the first and fifth line of the text block have an indentation of 6 spaces, while other lines have an indentation of 8 spaces. For all lines, the indentation of 6 spaces is introduced by Java source code.

Figure 12. Indentation in source code

```
public class Indentation {

  private static final String BLOCK = """
      <user>
        <id>001</id>
        <name>alex</name>
        <email>alex@example.com</email>
      </user>
      """;
}
```

The actual content should be as shown below. This means the 6 spaces introduced by Java source code should be removed. This process is called re-indentation.

Figure 13. Actual XML content

```
<user>
  <id>001</id>
  <name>alex</name>
  <email>alex@example.com</email>
</user>
```

A line in a text block may contain both leading and trailing white spaces. Trailing white spaces are removed automatically by the Java compiler. For leading white spaces, they may be used for indentation. Java compiler applies a *re-indentation algorithm* to remove extra leading white spaces.

To generate content with correct indentation, Java compiler removes the **same number** of white spaces from each line. The number of white spaces to remove is the minimal number of white spaces in all lines. In the code above, the number of white spaces to remove is 6. After removing 6 white spaces in each line, we can get the desired output.

Special attention should be paid to the trailing blank line. The position of the closing delimiter in the trailing blank line can affect the number of white spaces to remove.

In the code below, the trailing blank line has an indentation of 6 spaces, which is the minimal number of white spaces in all lines. So only 6 white spaces will be removed. In the result content, the first line will have an indentation of 4 spaces.

Figure 14. Indentation with trailing blank line

```
public class Indentation2 {

  private static final String BLOCK = """
          <user>
            <id>001</id>
            <name>alex</name>
            <email>alex@example.com</email>
          </user>
      """;
}
```

Escape Sequences

Text blocks support all of the escape sequences supported in string literals. We can use `"` and `""` freely in a text block. However, if you want to add more than two double quotes in a text block, some of these double quotes need to be escaped. More specifically, if you want to add `n` double quotes, at least `Math.floorDiv(n, 3)` of them need to be escaped. For example, to add 7 double quotes in a text block, at least 2 double quotes need to be escaped, which can be written as `""\"""\""`.

Two new escape sequences are added.

1. `\<line-terminator>` explicitly suppresses the insertion of a newline character. It can only be used in text blocks.
2. `\s` translates to a single space. It can be in text blocks, traditional string literals, and character literals.

Sometimes we may want to add very long strings in the code. Instead of using string concatenations, we can text blocks. However, text blocks will add line terminators when the long string is broken into multiple lines. We can use `\<line-terminator>` to suppress this behavior.

In the code below, the result string `LONG_STRING` won't have line terminators.

Figure 15. Long string without line terminators

```
public class LongString {

  private static final String LONG_STRING = """
      This a very long string. \
      This string continues. \
      The last part of this string. \
      """;
}
```

`\s` escape sequence has a special usage in text blocks. Trailing white spaces are removed by the Java compiler. If we want to keep some trailing white spaces for formatting purposes, we can add `\s` to the end of the line. All white spaces before `\s` will be kept, while other white spaces after `\s` will be removed.

In the code below, all lines will have exactly 11 characters.

Figure 16. Keep white spaces

```
public class KeepWhiteSpace {

  private static final String BLOCK = """
      alex:    1\s
      bob:    12\s
      david: 100\s
      """;
}
```

New String Methods

New methods are added to `String` related to text blocks.

- `stripIndent()` removes incidental white spaces from a string. This method actually implements the re-indentation algorithm performed by Java compiler to text blocks.
- `translateEscapes()` translates escape sequences in a string.
- `formatted(Object... args)` formats using current string as the format string with provided arguments.

The `formatted` method is very convenient when using text blocks to format strings.

The code below shows an example of using text blocks to format strings.

Figure 17. Use text block to format strings

```
public class FormatStrings {

  public String format(User user) {
    return """
        <user>
          <id>%s</id>
          <name>%s</name>
          <email>%s</email>
        </user>
        """.formatted(user.id(), user.name(), user.email());
  }
}
```

Low-level APIs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

JFR Event Streaming

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Event Stream

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Event

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Example

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

JVM Constants API

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Loadable Constants

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Constants API

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Constable

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

ConstantDesc

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Hidden Classes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Names of Hidden Classes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Create Hidden Classes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Class Loading of Hidden Classes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Unloading of Hidden Classes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Stack Traces

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Nest-based Access Control

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Standard Libraries

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Unix-Domain Socket Channels

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Reimplement the Legacy Socket API

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Reimplement the Legacy `DatagramSocket` API

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Enhanced Pseudo-Random Number Generators

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

New macOS Rendering Pipeline

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Edwards-Curve Digital Signature Algorithm (EdDSA)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Context-Specific Deserialization Filters

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

JVM

This chapter discusses changes related to JVM.

Class-Data Sharing

Class-Data Sharing (CDS) is introduced in JDK 5. CDS allows a set of classes to be pre-processed into a shared archive file that can then be memory-mapped at runtime to reduce startup time. It can also reduce dynamic memory footprint when multiple JVMs share the same archive file.

In Java 12 (JEP 341), a CDS archive is generated automatically from the default class list. This archive file is put into the `lib/server` directory. Since the `-Xshare:auto` option was enabled by default for the server VM in Java 11, users can benefit from this CDS change automatically. If you don't want this feature, it can be disabled via `-Xshare:off`.

In Java 10 (JEP 310), CDS was enhanced to allow application classes to be placed in the shared archive. CDS can be enabled for the system class loader, the platform class loader, and user-defined class loaders by specifying the `-XX:+UseAppCDS` option.

To actually use CDS for application classes, there are typically three steps:

First, we need to determine the classes to archive. This is done by running the application with `-Xshare:off` option, and `-XX:DumpLoadedClassList` option to record the classes that are loaded. For example:

Figure 18. Determine the classes to archive

```
$ java -Xshare:off -XX:+UseAppCDS \
    -XX:DumpLoadedClassList=app.lst -cp app.jar MyApp
```

Then, we can create the AppCDS archive. This is done by using the `-Xshare:dump -XX:+UseAppCDS` options, with `-XX:SharedClassListFile` option to specify the list of classes generated in the step above. For example:

Figure 19. Create AppCDS archive

```
$ java -Xshare:dump -XX:+UseAppCDS \
    -XX:SharedClassListFile=app.lst \
    -XX:SharedArchiveFile=app.jsa -cp app.jar
```

Finally, we can use the AppCDS archive. This is done by using the `-Xshare:on -XX:+UseAppCDS` options. For example:

Figure 20. Use AppCDS archive

```
$ java -Xshare:on -XX:+UseAppCDS \
    -XX:SharedArchiveFile=app.jsa \
    -cp app.jar MyApp
```

Steps above make AppCDS inconvenient to use, because a trial run is required to generate the class list. In Java 13 (JEP 350), we can enable dynamic archiving to eliminate the first step.

The first step is to create a shared archive dynamically when an application exits. All we need to do is to specify the `-XX:ArchiveClassesAtExit` option.

Figure 21. Create a shared archive when an application exits

```
$ java -XX:ArchiveClassesAtExit=app.jsa -cp app.jar MyApp
```

Then we can run the application again using the dynamic archive.

Figure 22. Run the application using the archive

```
$ java -XX:SharedArchiveFile=app.jsa -cp app.jar MyApp
```

The dynamic archive requires the default CDS archive to be used as its base archive.

> If the dynamically generated archive doesn't meet your requirement, you can still use the three steps above to generate an archive.

Helpful NullPointExceptions

`NullPointerException` is very common in Java programs. However, the message of this exception is not very helpful to identify the actual cause, especially for long reference paths. The message is just `null`. We can only use the line number to locate the place that causes this exception.

For long reference paths, it's hard to pinpoint the actual `null` object. For example, if a `NullPointerException` is thrown when an access path like `a.b.c.d`, it could be `a`, `b`, or `c` is `null`. It's hard to tell by checking the line numbers.

In Java 14 (JEP 358), a new option `-XX:+ShowCodeDetailsInExceptionMessages` can be specified to enable detailed messages of `NullPointerException`. The message is generated by analyzing bytecode instructions by the JVM.

The code below throws a `NullPointException` when accessing `b.a`.

Figure 23. Code with NPE

```
1  public class NPE {
2  
3    private static class A {
4  
5      void doSomething() {
6      }
7    }
8  
9    private static class B {
10 
11     A a;
12   }
13 
14   public static void main(String[] args) {
15     B b = new B();
16     b.a.doSomething();
17   }
18 }
```

Without the detailed message, the exception message looks like below. We can only know that the exception happens at line 18.

Figure 24. NullPointerException without detailed message

```
Exception in thread "main" java.lang.NullPointerException
    at io.vividcode.java11to17.jvm.NPE.main(NPE.java:18)
```

With the detailed messages enabled, the exception message looks like below. From the message, we can know that `b.a` is `null`.

Figure 25. NullPointerException with detailed message

```
Exception in thread "main" java.lang.NullPointerException: Cannot invoke "io.\
vividcode.java11to17.jvm.NPE$A.doSomething()" because "b.a" is null
    at io.vividcode.java11to17.jvm.NPE.main(NPE.java:18)
```

As of Java 17, the option `ShowCodeDetailsInExceptionMessages` has been enabled by default. If you don't like it, it can be disabled by specifying the `-XX:-ShowCodeDetailsInExceptionMessages` option.

Elastic Metaspace

In JVM, metaspace memory is managed in per-class-loader arenas. For applications that use many small class loaders, this may cause high metaspace usage. In Java 16 (JEP 387), the metaspace memory allocator has been replaced with a buddy-based allocation scheme, which can reduce class loader overhead.

The memory is committed from the operating system to arenas on demand, which can reduce footprint for certain class loaders.

The metaspace memory is arranged into uniformly-sized *granules* which can be committed and uncommitted independently of each other.

Deprecate and Disable Biased Locking

Biased locking is an optimization technique used in the HotSpot Virtual Machine to reduce the overhead of uncontended locking. Biased locking doesn't bring much performance gains in nowadays applications. However, it makes code hard to maintain.

In Java 15 (JEP 374), biased locking is disabled by default. It can still be enabled using the `-XX:+UseBiasedLocking` option. The option `UseBiasedLocking` and all options related to the configuration and use of biased locking are deprecated.

Packaging Tool

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Use jpackage

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Custom Runtime Image

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Use jdeps

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Use jlink

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Create Package

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Garbage Collectors

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

ZGC

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Shenandoah

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Deprecated GC

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Removed GC

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Updates to G1

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Abortable Mixed Collections for G1

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Promptly Return Unused Committed Memory from G1

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

NUMA-Aware Memory Allocation for G1

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Foreign Function & Memory API

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Foreign Memory

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Memory Segments

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Create Memory Segments

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Memory Addresses

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Dereferencing Memory Segments

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Memory Layouts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Resource Scopes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Thread confinement

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Cleaner

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Foreign Functions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Downcall - Call from Java to Native

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Upcall - Call From Native To Java

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/java11to17.

Deprecated & Removed Features

Many features have been marked as deprecated or removed since Java 11. You need to pay special attention to these features if your application still uses them.

Deprecated Features

Below are deprecated features in Java 17.

Applet API

Applet API is deprecated for removal in Java 17 (JEP 398). Since all browser vendors have either removed support for Java browser plug-in or plan to do so, removing this API is completely OK.

Applet API was previously deprecated in Java 9 (JEP 289).

Security Manager

Security Manager is deprecated for removal in Java 17 (JEP 411). `java.lang.SecurityManager` and other related classes and methods have been annotated with `@Deprecated(forRemoval=true)`.

A warning message is issued when Security Manager is enabled at startup or installed dynamically at runtime.

In Java 12, the value `disallow` can be specified for the system property `java.security.manager` to disable Security Manager. In this case, no security manager is set at startup and cannot be set dynamically at runtime. Calling `System::setSecurityManager` always throws an `UnsupportedOperationException`.

More changes to security manager will come in Java 18 and later versions. In Java 18, `disallow` will become the default value of `java.security.manager`.

Security manager has been in Java since 1.0, but it's rarely used to secure Java code in both client-side and server-side. If your application uses security manager, it's time to plan the migration without it.

Removed Features

Below are features removed in Java 17.

Experimental AOT and JIT Compiler

The Graal compiler was incorporated into JDK as an experimental ahead-of-time (AOT) and just-in-time (JIT) compiler.

- Ahead-of-time compilation in JDK 9 (JEP 295)
- JIT compiler in JDK 10 (JEP 317).

The Graal compiler was removed in Java 17. Below is a list of what's removed:

- Module `jdk.aot`
- Module `jdk.internal.vm.compiler`
- Module `jdk.internal.vm.compiler.management`
- Tool `jaotc`

The experimental Java-level JVM compiler interface (JVMCI) is preserved, so you can still use externally-built versions of the Graal compiler for JIT compilation.

If you want to use the Graal compiler for either AOT or JIT compilation, you should use GraalVM directly.

RMI Activation

RMI Activation was deprecated for removal in Java 15 by JEP 385 and removed in Java 17 by JEP 407. The rest of RMI is preserved.

The package `java.rmi.activation` is removed.

Pack200 Tools

Pack200 is a compression scheme for JAR files. It's been used to compress JDK and client applications since Java SE 5.0. JDK 8 was the last release compressed using Pack200. Starting from JDK 9, new compression schemes were used.

Pack200 tools and related API were deprecated for removal in Java 11 (JEP 336) and removed in Java 14 (JEP 367).

Below is a list of what' removed:

- Tools `pack200` and `unpack200`
- Classes `java.util.jar.Pack200`, `java.util.jar.Pack200.Packer` and `java.util.jar.Pack200.Unpacker`
- Module `jdk.pack`

Nashorn JavaScript Engine

The Nashorn JavaScript engine was introduced in JDK 8 (JEP 174) to replace the old Rhino scripting engine. With the rapid development of ECMAScript language specification, it becomes challenging to manage the Nashorn engine to keep up with the specification. The engine was deprecated in Java 11 (JEP 335) and removed in Java 15 (JEP 372).

Below is a list of what' removed:

- Module `jdk.scripting.nashorn`
- Module `jdk.scripting.nashorn.shell`
- Tool `jjs`

Solaris and SPARC Ports

Solaris/SPARC, Solaris/x64, and Linux/SPARC ports were deprecated for removal in Java 14 (JEP 362) and removed in Java 15 (JEP 381).

Misc.

This chapter discusses miscellaneous changes.

Restore Always-Strict Floating-Point Semantics

Starting from Java SE 1.2, there are two floating-point modes in Java: the default and strict floating-point modes. To use strict floating-point mode, `strictfp` should be used.

In Java 17 (JEP 306), the default floating-point mode is removed. Now strict floating-point mode is always used. `strictfp` is no longer required.

macOS/AArch64 Port

> This is introduced in Java 17 by JEP 391.

JDK has been ported to macOS on AArch64.

Alpine Linux Port

> This is introduced in Java 16 by JEP 386.

JDK has been ported to Alpine Linux, and other Linux distributions that use `musl` as their primary C library, on both the x64 and AArch64 architectures.

Strongly Encapsulate JDK Internals

With the introduction of Java Platform Module System in Java 9, JDK internals are supposed to be encapsulated. However, many libraries that rely on JDK internal APIs may stop working if those APIs are encapsulated. To facilitate the migration, JDK 6 to JDK 15 allows access to those APIs with warnings printed in the console.

In Java 16 (JEP 396), the strong encapsulation has been enabled by default by setting the default value of `--illegal-access` option to `deny`. However, it's still possible to override this behavior by specifying the `permit` as the value of `--illegal-access` option.

In Java 17 (JEP 403), it's no longer possible to relax the strong encapsulation of internal elements using `--illegal-access` option.

Critical APIs in `sun.misc` and `sun.reflect` packages are still open.

Migrate to GitHub

OpenJDK's source code has been migrated to GitHub.