*Record Patterns (Preview) – We can allows apply a so-called "record pattern".*

*Ex -* public record Position(int x, int y) {}

*Before - private void print(Object object) {*

*if (****object instanceof Position position****) {*

*System.out.println("object is a position, x = " + position.x()*

*+ ", y = " + position.y());*

*}*

*// else ...*

*}*

*Now - private void print(Object object) {*

*if (****object instanceof Position(int x, int y)****) {*

*System.out.println("object is a position, x = " + x + ", y = " + y);*

*}*

*// else ...*

*}*

***Record Pattern with switch –***

*private void print(Object object) {*

*switch (object) {*

*case* ***Position(int x, int y)***

*-> System.out.println("object is a position, x = " + x + ", y = " + y);*

*// other cases ...*

*}*

*}*

***Nested Record Patterns*** *- It is also possible to match nested records.*

public record Path(Position from, Position to) {}

private void print(Object object) {

if (**object instanceof Path(Position(int x1, int y1), Position(int x2, int y2))**) {

System.out.println("object is a path, x1 = " + x1 + ", y1 = " + y1

+ ", x2 = " + x2 + ", y2 = " + y2);

}

// else ...

}

private void print(Object object) {

switch (object) {

case Path(**Position(int x1, int y1), Position(int x2, int y2))**

-> System.out.println("object is a path, x1 = " + x1 + ", y1 = " + y1

+ ", x2 = " + x2 + ", y2 = " + y2);

// other cases ...

}

}

*Record patterns thus provide us with an elegant way to access the record's elements after a type check.*

***Virtual Threads (Preview) – JEP 425***

*The most exciting innovation in Java 19 is "Virtual Threads". Virtual threads have been developed in Project Loom for several years and could only be tested with a self-compiled JDK so far.*

***Structured Concurrency (Incubator) – JEP 428***

*Also developed in Project Loom and initially released as an incubator feature in Java 19 with JDK Enhancement Proposal 428 is the so-called "Structured Concurrency.*

*When a task consists of several subtasks that can be processed in parallel, Structured Concurrency allows us to implement this in a particularly readable and maintainable way.*

***Foreign Function & Memory API (Preview) – JEP 424***

*In Project Panama, a replacement for the cumbersome, error-prone, and slow Java Native Interface (JNI) has been in the works for a long time.*

*The "Foreign Memory Access API" and the "Foreign Linker API" were already introduced in Java 14 and Java 16 – both initially individually in the incubator stage. In Java 17, these APIs were combined to form the "Foreign Function & Memory API" (FFM API), which remained in the incubator stage until Java 18.*

*In Java 19, JDK Enhancement Proposal 424 finally promoted the new API to the preview stage, which means that only minor changes and bug fixes will be made. So it's time to introduce the new API!*

*The Foreign Function & Memory API enables access to native memory (i.e., memory outside the Java heap) and access to native code (e.g., C libraries) directly from Java.*

***Vector API (Fourth Incubator) – JEP 426 -*** *The Vector API has been part of the JDK since Java 16 as an incubator and was further developed in Java 17 and Java 18.*

*With JDK Enhancement Proposal 426, Java 19 delivers the fourth iteration in which the API has been extended to include new vector operations – as well as the ability to store vectors in and read them from memory segments.*

***Deprecation of Locale class constructors***

*In Java 19, the public constructors of the Locale class were marked as "deprecated".*

*Instead, we should use the new static factory method Locale.of(). This ensures that there is only one instance per Locale configuration.*

Locale german1 = new Locale("de"); // deprecated

Locale germany1 = new Locale("de", "DE"); // deprecated

Locale german2 = Locale.of("de");

Locale germany2 = Locale.of("de", "DE");

System.out.println("german1 == Locale.GERMAN = " + (german1 == Locale.GERMAN));

System.out.println("germany1 == Locale.GERMANY = " + (germany1 == Locale.GERMANY));

System.out.println("german2 == Locale.GERMAN = " + (german2 == Locale.GERMAN));

System.out.println("germany2 == Locale.GERMANY = " + (germany2 == Locale.GERMANY));

***java.lang.ThreadGroup is degraded -***

*In Java 14 and Java 16, several Thread and ThreadGroup methods were marked as "deprecated for removal". The reasons are explained in the linked sections.*

*The following of these methods have been decommissioned in Java 19:*

*ThreadGroup.destroy() – invocations of this method will be ignored.*

*ThreadGroup.isDestroyed() – always returns false.*

*ThreadGroup.setDaemon() – sets the daemon flag, but this has no effect anymore.*

*ThreadGroup.getDaemon() – returns the value of the unused daemon flags.*

*ThreadGroup.suspend(), resume(), and stop() throw an UnsupportedOperationException.*

*Automatic Generation of the CDS Archive - Application CDS makes it possible to load the classes of an application into the memory once when operating several JVMs on one machine and to share this memory area with all JVMs. This saves memory and time for loading the .jar and .class files and converting them into a platform-specific binary format.*

*With Java 19, the configuration of AppCDS has been simplified once again. You can now specify the following VM parameter to automatically create or update a CDS archive.*

java -XX:+AutoCreateSharedArchive -XX:SharedArchiveFile=helloworld.jsa \

-cp target/helloworld.jar eu.happycoders.appcds.Main

***Linux/RISC-V Port – JEP 422 -*** *Due to the increasing use of RISC-V hardware, a port for the corresponding architecture was made available with JEP 422.*

***Additional Date-Time Formats -*** *We can use the DateTimeFormatter.ofLocalizedDate(…), ofLocalizedTime(…), and ofLocalizedDateTime(…) methods and the subsequent call to withLocale(…) to generate a date/time formatter. We control the exact format using the FormatStyle enum, which can take the values FULL, LONG, MEDIUM, and SHORT.*

*In Java 19, the method ofLocalizedPattern(String requestedTemplate) was added, with which we can also define flexible formats. Here is an example:*

*LocalDate now = LocalDate.now();*

*DateTimeFormatter formatter = DateTimeFormatter.ofLocalizedPattern("yMMM");*

*System.out.println("US: " + formatter.withLocale(Locale.US).format(now));*

*System.out.println("Germany: " + formatter.withLocale(Locale.GERMANY).format(now));*

*System.out.println("Japan: " + formatter.withLocale(Locale.JAPAN).format(now));*

*The code outputs the following:*

*US: Jan 2024*

*Germany: Jan. 2024*

*Japan: 2024年1月*

***New System Properties for System.out and System.err*** *- Since version 18, Java automatically uses the character encoding of the console or terminal for printing to System.out and System.err. On Linux, this is usually UTF-8 and on Windows, code page 437.*