JAVA 20

Java SE 20 was released March 21, 2023, so it should be generally available when this article is published. Most of the JEPs in this release have something to do with Pattern Matching or Virtual Threads and are resubmissions of already known Incubator and Preview features. This version doesn't contain any new main features. The most exciting innovation in this release is called 'Scoped Values' and is intended to widely replace thread-local variables.

In this article we will take a look at all the new and resubmitted Incubator and Preview features.

All code examples were tried out in a Docker container with OpenJDK 20 (Listing 1) [2].

L1
```
$ docker run -it --rm \
      -v $(pwd)/src:/src \
      openjdk:20-slim /bin/bash
$ cd /src/java20
```

{ PREVIEW AND INCUBATOR FEATURES }

429: Scoped Values (Incubator)

Just like Virtual Threads (see below), Scoped Values were developed as part of Project Loom [4].

Project Loom is intended to explore, incubate and deliver Java VM features, and APIs built on top of them for the purpose of supporting easy-to-use, high-throughput lightweight concurrency, and new programming models on the Java platform.

The Scoped Values feature in Java provides a way to define a value within a particular scope and ensures that it is used only within that scope. This feature makes it easier to manage data and reduces the risk of errors by limiting the scope of that data to only the areas where it is needed.

L2
```
package java20;
import jdk.incubator.concurrent.*;
public class JEP429 {
    private static final ScopedValue<String>
USERNAME = ScopedValue.newInstance();
    public static void main(String[] args) {
        ScopedValue.where(USERNAME, "Duke", () ->
System.out.println(USERNAME.get()));
        ScopedValue.where(USERNAME, "Java", () ->
System.out.println(USERNAME.get()));
    }
}
$ java --add-modules jdk.incubator.concurrent
--enable-preview --source 20 JEP429.java
WARNING: Using incubator modules: jdk.incubator.
concurrent
Duke
Java
```

Ivo Woltring is a Principal Expert and Codesmith at Ordina and likes to keep up with new developments in the software world.

432: Record Patterns (Second Preview)

This JEP aims to improve the expressiveness and readability of code that deals with records. A record pattern can be used with `instanceof` or `switch` to access the fields of a record without casting and calling accessor methods.

Type pattern matching was introduced in Java through JEP 394 in Java SE 17. The switch case statement was enhanced to work with pattern matching in JEP 406 and 420 in Java SE 18 [2,3].

L3
```
public class JEP432 {
    record Pair(Object x, Object y) { }
    record Point(int x, int y) {}
    enum Color { RED, GREEN, BLUE }
    record ColoredPoint(Point p, Color c) {}
    record Rectangle(ColoredPoint upperLeft,
ColoredPoint lowerRight) {}
    public static void noMatchExample() {
        Pair p = new Pair(42, 42);
        System.out.println("p instanceof
Pair(String s, String t) -> "
                + (p instanceof Pair(String s,
String t)));
    }
    static void printUpperLeftColors(Rectangle[]
```