Java 20

Java 20, released on the 21st of March 2023, is the latest short-term incremental release built on Java 19 as of today. It consists of the seven important JDK Enhancement Proposals (JEPs) mentioned in JEP 2.0. The JEP process is used to evaluate proposals for enhancements to the JDK. The majority of the updates in Java 20 are improvements or enhancements to the features introduced in earlier releases. Moreover, Oracle JDK 20 isn't a long-term support release. Therefore, it will receive updates until the release of Java 21. In this article, we'll explore some of these new features.

1.. Scoped Values (JEP 429)

A large number of Java applications have components or modules that need to share data among themselves. Often, these modules are thread based, so we must protect the data they share from any change. We've been using variables of the type ThreadLocal to allow components to share data. But it has some consequences:

- A ThreadLocal variable is mutable. The ThreadLocal API allows access to both get() and set() methods of its variable type.
- We may face memory leak issues since the value of the ThreadLocal variables is retained until we explicitly call the remove() method on it or the thread exits. Thus, there's no binding to the lifetime of these per-thread variables.
- They may lead to excessive memory footprints in case of using large numbers of threads. This is because the child threads can inherit the ThreadLocal variables of parent threads, thus allocating memory for every ThreadLocal variable.

To overcome these problems of ThreadLocal variables, Java 20 has introduced scoped values for sharing data within and across threads. Scoped values provide a simple, immutable, and inheritable data-sharing option, specifically in situations where we're working with a large number of threads. A Scoped Value is an immutable value that is available for reading for a bounded period of execution by a thread. Since it's immutable, it allows safe and easy data-sharing for a limited period of thread execution. Also, we need not pass the values as method arguments. We can use the where() method of the class Scoped Value to set the value of a variable for the bounded period of thread execution. Moreover, once we get the data using the get() method, we cannot access it again.

Once the run() method of a thread finishes execution, the scoped value reverts to the unbound state. We can use the get() method to read the value of a scoped-value variable inside a thread.

2.. Record Patterns (JEP 432)

JDK 19 had already introduced Record Patterns as a preview feature. Java 20 delivers an improved and refined version of record patterns. Let's see some of the improvements in this release:

- Added support for type inference of arguments of generic record patterns.
- Added support for record patterns to be usable in the header of an enhanced for loop.
- Removed support for named record patterns, where we could provide an optional identifier to the record patterns that we can use to refer to the record pattern.

This release essentially aims to express more improved, composable data queries with pattern matching while not changing the syntax or semantics of type patterns.

3.. Pattern Matching for Switch (JEP 433)

Java 20 provides a refined version of pattern matching for switch expressions and statements, specifically about the grammar used in switch expressions. It was first delivered in Java 17, followed by some improvements in Java 18 and 19, thus expanding the usability and applicability of switch statements and expressions.

The main changes in this release include:

- Using a switch expression or a pattern over an enum class now throws a MatchException. Earlier, we used to get an IncompatibleClassChangeError if no switch label was applied at run time.
- There are improvements in the grammar for switch labels.
- They have added support for type-inference of arguments for generic record patterns in switch expressions and statements, along with the other constructs that support patterns.

4.. Foreign Function and Memory API (JEP 434)

Java 20 incorporates the improvements and refinements to the Foreign Function and Memory (FFM) API that were introduced in the previous Java versions. This is a second preview API. The Foreign Function and Memory API allow Java developers to access code from outside the JVM and manage memory out of the heap (i.e., memory not managed by the JVM). The FFM API aims to provide a safe, improved, and pure Java-based substitute to Java Native Interface (JNI).

It includes the following refinements:

- The MemorySegment and MemoryAddress abstractions are unified. This means that we can actually determine the range of memory addresses associated with a segment from its spatial bounds.
- They've also facilitated the use of pattern matching in switch expressions and statements by enhancing the sealed MemoryLayout hierarchy.
- Moreover, they've split the Memory Session into Arena and Segment Scope to facilitate sharing segments across maintenance boundaries.

5.. Virtual Threads (JEP 436)

Virtual threads were first proposed as a preview feature in JEP 425 and delivered in Java 19. Java 20 proposes the second preview aiming to gather more feedback and improvement suggestions after the use. Virtual threads are lightweight threads that reduce the effort of writing, maintaining, and observing high-throughput concurrent applications. Thus, it makes it easy to debug and troubleshoot the server applications with existing JDK tools. This could be useful in the scaling of server applications. However, we should note that the traditional Thread implementation still exists, and it doesn't aim to replace the basic concurrency model of Java. Below are a few of the minor changes since the first preview:

- They made the previously introduced methods in Thread – join(Duration), sleep(Duration), and threadId() – permanent in Java 20.
- Similarly, newly introduced methods in Future to examine the task state and result – state() and resultNow() – were made permanent in Java 20.
- Additionally, ExecutorService now extends AutoCloseable.

- The degradations to ThreadGroup, the legacy API for grouping threads, described in JEP 425, were made permanent in JDK 19. The ThreadGroup isn't suitable for grouping virtual threads.

6.. Structured Concurrency (JEP 437)

Structured Concurrency was proposed by JEP 428 and delivered in JDK 19 as an incubating API. This JEP proposes to re-incubate the API without much changes in JDK 20. Thus, it allows time for more feedback. The goal is to simplify multithreaded programming by introducing an API for structured concurrency. It improves reliability by grouping multiple threads doing similar tasks into a single unit of work. As a result, there's improved error handling and thread cancellations. Additionally, it promotes an improved way of concurrent programming aiming to protect from common risks arising from thread cancellation. However, there's one change in this re-incubated API. We get an updated StructuredTaskScope that supports the inheritance of scoped values by threads created in a task scope. Thus, we can now conveniently share immutable data across multiple threads.

7.. Vector API (JEP 438)

The Vector API was first proposed by JEP 338 and integrated into JDK 16 as an incubating API. This release (fifth incubator) is a follow-up of multiple rounds of incubations and integrations into all the previous Java versions. The Vector API is used to express vector computations within Java on supported CPU architectures. A vector computation is actually a sequence of operations on vectors. The Vector API aims to provide a means of vector computations that are more optimal than traditional scalar computations. This release doesn't introduce any change to the API compared to Java 19. However, it includes a few bug fixes and performance enhancements. Finally, development in the Vector API is closely aligned with Project Valhalla since it aims to adapt Project Valhalla's enhancements in the future.