JSR 396: Java SE 21

Iris Clark
Specification Lead
iris.clark@oracle.com
August 8, 2023
JSR 396: Java SE 21

**Specification**
- Latest: [https://cr.openjdk.org/~iris/se/21/latestSpec](https://cr.openjdk.org/~iris/se/21/latestSpec) (DRAFT 34)
- Public Review ends 21 Aug

**Reference Implementation (RI) – JDK 21**
- Latest: [https://jdk.java.net/21](https://jdk.java.net/21) (build 34)
- Repository: [https://github.com/openjdk/jdk21](https://github.com/openjdk/jdk21)
- Rampdown Phase 2 (RDP2)
  - Feature set frozen
  - Development very strictly limited to selected bug fixes
- 12 Integrated SE JEPs; 129 approved SE CSRs
- General Availability (GA): 2023/09/19

**Technology Compatibility Took Kit (TCK) – JCK 21**
- Stabilization fork in Jul; Code freeze recently

**Schedule**
- 2022/12/07 Expert Group Formation
- 2023/07/18 – 2023/08/21 Public Review
- 2023/08/22 – 2023/08/28 Public Review – Final Approval Ballot
- 2023/09 Final Release
SE JEPs in Java SE 21

Language
- **440** Record Patterns
- **441** Pattern Matching for `switch`
- **430** String Templates (Preview)
- **443** Unnamed Patterns & Variables (Preview)
- **445** Unnamed Classes & Instance `main` Methods (Preview)

Virtual Machine
- **451** Prepare to Disallow the Dynamic Loading of Agents

Libraries
- **431** Sequenced Collections
- **444** Virtual Threads
- **453** Structured Concurrency (Preview)
- **446** Scoped Values (Preview)
- **442** Foreign Function & Memory API (Third Preview)

Security
- **452** Key Encapsulation Mechanism API
An Aside: JEP 12: Preview Features

- Preview features are fully specified, fully implemented, but subject to change.
- Code using a preview feature may not necessarily compile or run in another release.
- Must be enabled at compile time and run time:

  `javac --release 21 --enable-preview Main.java`

  `java --enable-preview Main`
  `java --source 21 --enable-preview Main.java // source code launcher`
  `jshell --enable-preview`

- All preview features in the current release must take one of the following actions in the next feature release
  - Remove
  - Re-preview
  - Standardize
JEP 440: Record Patterns

Extend pattern matching to de-structure instances of Record classes.

```java
static void printSum(Object obj) {
    if (obj instanceof Point(int x, int y)) {
        System.out.println(x+y);
    }
}
```

History
• First previewed in Java SE 19, re-previewed in Java SE 20

Why
• More sophisticated data queries
• Another step towards declarative, data-focused programming
JEP 441: Pattern Matching for switch

Enhance switch statements to support additional types and semantics.

```java
static String formatterPatternSwitch(Object obj) {
    return switch (obj) {
        case Integer i -> String.format("int %d", i);
        case String s  -> String.format("String %s", s);
        default        -> obj.toString();
    };
}
```

History
• First previewed in Java SE 17, re-previewed in Java SE 18, 19, and 20

Why
• Express complex data-oriented queries concisely and safely
JEP 430: String Templates (Preview)

Introduce string composition that couples literal text with embedded expressions and template processors.

String name = "Duke";
String info = STR."My name is \{name}";
assert info.equals("My name is Duke");   // true

Why
• Commonly used feature used in other popular programming languages
• Existing string composition techniques (String concatenation with ‘+’, StringBuilder, Formatter.format()) are verbose
• String composition that achieves the clarity of string interpolation without the inherent hazards (e.g. SQL injection attacks)
JEP 443: Unnamed Patterns & Variables (Preview)

Use the underscore character, '_', to identify unnecessary nested patterns and variables which must be declared but will not be used. Unnamed patterns may be used in record patterns.

```java
// before nested pattern
if (r instanceof ColoredPoint(Point(int x, int y), Color c)) {
    ... x ...
}

// after, using unnamed pattern
if (r instanceof ColoredPoint(Point(int x, _), _)) { ... x ... }
```

Why
• Improve readability of record patterns by eliding unnecessary patterns
• Improve maintainability by eliminating useless declarations
JEP 445: Unnamed Classes & Instance main Methods (Preview)

Reduce syntactic complexity of simple programs for novice users.

```java
void main() {
    System.out.println("Hello, World!");
}
```

**Why**
- Traditional “Hello, World” exposes too many concepts that may intimidate beginning programmers
- Reduce ceremony for simple programs such as scripts and command-line utilities
JEP 431: Sequenced Collections

Enhance the collections framework with new interfaces for sequenced collections which have a well-defined order.

<table>
<thead>
<tr>
<th>First element</th>
<th>Last element</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>List</strong></td>
<td>list.get(0)</td>
</tr>
<tr>
<td><strong>Deque</strong></td>
<td>deque.getFirst()</td>
</tr>
<tr>
<td><strong>SortedSet</strong></td>
<td>sortedSet.first()</td>
</tr>
<tr>
<td><strong>LinkedHashSet</strong></td>
<td>linkedHashSet.iterator.next()</td>
</tr>
<tr>
<td><strong>SequencedCollection</strong></td>
<td>c.getFirst()</td>
</tr>
<tr>
<td></td>
<td>list.get(list.size() - 1)</td>
</tr>
<tr>
<td></td>
<td>deque.getLast()</td>
</tr>
<tr>
<td></td>
<td>sortedSet.last()</td>
</tr>
<tr>
<td></td>
<td>// missing</td>
</tr>
<tr>
<td></td>
<td>c.getLast()</td>
</tr>
</tbody>
</table>

**Why**
- Simplifies code that depends only on sequence rather than class-specific behaviour
JEP 444: Virtual Threads

Introduce lightweight threads that dramatically reduce the effort of writing, maintaining, and observing high throughput concurrent applications.

```java
try (var executor = Executors.newVirtualThreadPerTaskExecutor()) {
    IntStream.range(0, 10_000).forEach(i -> {
        executor.submit(() -> {
            Thread.sleep(Duration.ofSeconds(1));
            return i;
        });
    });
} // executor.close() is called implicitly, and waits
```

**History**
- First previewed in Java SE 19, re-previewed in Java SE 20

**Why**
- Concurrency limited by the number of platform threads, implemented as OS threads
JEP 453: Structured Concurrency (Preview)

Introduce APIs to structure a task as a family of concurrent subtasks, and to coordinate them as a unit.

```java
Callable<String> task1 = ...;
Callable<Integer> task2 = ...;
try (var scope = new StructuredTaskScope<Object>()) {
    Subtask<String> subtask1 = scope.fork(task1);
    Subtask<Integer> subtask2 = scope.fork(task2);
    scope.join();
    ... process results/exceptions ...
} // close
```

History
• First incubated in Java SE 19, re-incubated in Java SE 20

Why
• Provide structure for large numbers of virtual threads
• Streamline error handling, improving reliability and enhancing observability
JEP 446: Scoped Values (Preview)

Introduce scoped values, which enable safe and efficient sharing of immutable data within and across threads.

```java
final static ScopedValue<...> NAME = ScopedValue.newInstance();

// In some method
ScopedValue.runWhere(NAME, “duke”, () -> { ... NAME.get() ... call methods ... });

// In a method called directly or indirectly from the lambda expression
... NAME.get() ...
```

**History**
- Incubated in Java SE 20

**Why**
- Alternative to thread-local variables and method arguments for sharing data across components
The API enables Java programs to call native libraries and process native data without the brittleness and danger of JNI.

```java
Linker linker = Linker.nativeLinker();
SymbolLookup stdlib = linker.defaultLookup();
MethodHandle strlen = linker.downcallHandle(
    stdlib.find("strlen").orElseThrow(),
    FunctionDescriptor.of(ValueLayout.JAVA_LONG, ValueLayout.ADDRESS)
);
try (Arena arena = Arena.ofConfined()) {
    MemorySegment cString = arena.allocateUtf8String("Hello");
    long len = (long)strlen.invokeExact(cString);
} // 5
```

**History**
- Incubated in Java SE 17 and 18. First previewed in Java SE 19, re-previewed in Java SE 20

**Why**
- Provide a safer alternative to JNI
JEP 451: Prepare to Disallow the Dynamic Loading of Agents

Introduces a warning when dynamic loading of agents is attempted

WARNING: A {Java,JVM TI} agent has been loaded dynamically (file:/u/duke/agent.jar)
WARNING: If a serviceability tool is in use, please run with -XX:+EnableDynamicAgentLoading to hide this warning
WARNING: If a serviceability tool is not in use, please run with -Djdk.instrument.traceUsage for more information
WARNING: Dynamic loading of agents will be disallowed by default in a future release

Command-line option -XX:+EnableDynamicAgentLoading will suppress this warning in Java SE 21 and will be required to enable dynamic agent loading in a future release.

History
• Originally proposed in 2017 circa Java SE 9 but deferred

Why
• Integrity by default
JEP 452: Key Encapsulation Mechanism API

Define APIs for key encapsulation mechanisms (KEMs) which use encryption to secure symmetric keys using public key cryptography.

- Example KEM algorithms include:
  - RSA-KEM
  - Elliptic Curve Integrated Encryption Scheme (ECIES)
  - Future NIST Post-quantum cryptography standard

Why

- Support current and future industry standards
- Defend against quantum attacks
openjdk.org/projects/jdk/21

---

**JDK 21**

This release will be the Reference Implementation of version 21 of the Java SE Platform, as specified by JSR 396 in the Java Community Process.

**Status**

JDK 21 is in Rampdown Phase Two. The overall feature set is frozen. No further JEPs will be targeted to this release.

The stabilization repository, jdk21, is open for select bug fixes, with approval, per the JEP Release Process (JEP 3). Late enhancements are still possible, with approval, but the bar is now extraordinarily high. Integrate most stabilization changes via backports from the main-line repository.

- RDP 2 candidate bugs
- Fix-Request Process
- Bug Deferral Process
- Late Enhancement Request Process

Early-access binaries under the GPL are available [here](https://openjdk.org/projects/jdk/21/).

**Schedule**

<table>
<thead>
<tr>
<th>Date</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>2023/06/08</td>
<td>Rampdown Phase One (fork from main line)</td>
</tr>
<tr>
<td>2023/07/20</td>
<td>Rampdown Phase Two</td>
</tr>
<tr>
<td>2023/08/10</td>
<td>Initial Release Candidate</td>
</tr>
<tr>
<td>2023/08/24</td>
<td>Final Release Candidate</td>
</tr>
<tr>
<td>2023/09/19</td>
<td>General Availability</td>
</tr>
</tbody>
</table>

**Features**

<table>
<thead>
<tr>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>
Other JEPs

Schedule

- 2023/06/08: Rampdown Phase One (fork from main line)
- 2023/07/20: Rampdown Phase Two
- 2023/08/10: Initial Release Candidate
- 2023/08/24: Final Release Candidate
- 2023/09/19: General Availability

Features

- 430: String Templates (Preview)
- 431: Sequenced Collections
- 439: Generational ZGC
- 440: Record Patterns
- 441: Pattern Matching for switch
- 442: Foreign Function & Memory API (Third Preview)
- 443: Unnamed Patterns and Variables (Preview)
- 444: Virtual Threads
- 445: Unnamed Classes and Instance Main Methods (Preview)
- 446: Scoped Values (Preview)
- 448: Vector API (Sixth Incubator)
- 449: Deprecate the Windows 32-bit x86 Port for Removal
- 451: Prepare to Disallow the Dynamic Loading of Agents
- 452: Key Encapsulation Mechanism API
- 453: Structured Concurrency (Preview)

Last update: 2023/7/20 15:05 UTC
Other notable changes in Java SE 21

129 Compatibility & Specification Review (CSR) Requests
https://bugs.openjdk.org/issues/?filter=43361

1 JSR Maintenance Release
269: Pluggable Annotations Processing API [MR15]

3 Removed APIs
java.lang.Compiler (9)
java.lang.ThreadGroup
  .allowThreadSuspension(boolean) (14)
javax.management.remote.rmi
  .RMIIOPServerImpl (9)

2 Terminally Deprecated APIs Added
javax.management.remote.JMXConnector
  .getMBeanServerConnection()
javax.swing.plaf.synth.SynthLookAndFeel
  .load()
Resources

- https://openjdk.org/projects/jdk/21/spec/
  - JEPs: https://bugs.openjdk.org/secure/Dashboard.jspa?selectPagId=21418
  - CSRs: https://bugs.openjdk.org/secure/Dashboard.jspa?selectPagId=21419
  - https://mail.openjdk.org/mailman/listinfo/java-se-spec-experts
  - https://jdk.java.net/21/

- https://openjdk.org/projects/jdk/22/spec/
- https://mail.openjdk.org
- https://github.com/openjdk