JDK 22 new features

Tomaž Cerar, GlobalID

JDK 22 basics

- Non LTS release
- Small but important changeset
  - 12 JDK Enhancements Proposals (JEPs)
  - 8 preview or incubator modules
- Released 2024-03-19

JDK 21 basics

- Long-term support (LTS) release
  - Oracle switched from 3 years between LTS to 2 years
- Feature rich
  - 15 JDK Enhancements Proposals (JEPs)
  - 7 preview or incubator modules
- Released 2023-09-19
- Next LTS is expected to be 25
  - Release September 2025

JDK 18 JEPs

400:	UTF-8 by Default
408:	Simple Web Server
413:	Code Snippets in Java API Documentation
416:	Reimplement Core Reflection with Method Handles
417:	Vector API (Third Incubator)
418:	Internet-Address Resolution SPI
419:	Foreign Function & Memory API (Second Incubator)
420:	Pattern Matching for switch (Second Preview)
421:	Deprecate Finalization for Removal

JDK 19 JEPs

405: Record Patterns (Preview)

422: Linux/RISC-V Port

424: Foreign Function & Memory API (Preview)

425: Virtual Threads (Preview)

426: Vector API (Fourth Incubator)

427: Pattern Matching for switch (Third Preview)

428: Structured Concurrency (Incubator)

JDK 20 JEPs

429: Scoped Values (Incubator)

432: Record Patterns (Second Preview)

433: Pattern Matching for switch (Fourth Preview)

434: Foreign Function & Memory API (Second Preview)

436: Virtual Threads (Second Preview)

437: Structured Concurrency (Second Incubator)

438: Vector API (Fifth Incubator)

JDK 21 JEPs

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)

JDK 22 JEPs

423:	**Region Pinning for G1**
447:	Statements before super(...) (Preview)
454:	**Foreign Function & Memory API**
456:	**Unnamed Variables & Patterns**
457:	Class-File API (Preview)
458:	**Launch Multi-File Source-Code Programs**
459:	String Templates (Second Preview)
460:	Vector API (Seventh Incubator)
461:	Stream Gatherers (Preview)
462:	Structured Concurrency (Second Preview)
463:	Implicitly Declared Classes and Instance Main Methods (Second Preview)
464:	Scoped Values (Second Preview)

Additions

Language

441: Pattern Matching for switch (21)

440: Record Patterns (21)

456: Unnamed Variables & Patterns (22)

Libraries

444: Virtual Threads (21)

454: Foreign Function & Memory API (22)

431: Sequenced Collections (21)

452: Key Encapsulation Mechanism API (21)

400: UTF-8 by Default (18)

Networking

418: Internet-Address Resolution SPI (18)

408: Simple Web Server (18)

Other

439: Generational ZGC (21)

423: Region Pinning for G1 (22)

416: Reimplement Core Reflection with Method Handles (18)

422: Linux/RISC-V Port (19)

451: Prepare to Disallow the Dynamic Loading of Agents (21)

413: Code Snippets in Java API Documentation (18)

458: Launch Multi-File Source-Code Programs(22)

Preview & Incubating

447: Statements before super(...) (Preview)

457: Class-File API (Preview)

459: String Templates (Second Preview)

460: Vector API (Seventh Incubator)

461: Stream Gatherers (Preview)

462: Structured Concurrency (Second Preview)

463: Implicitly Declared Classes and Instance Main Methods (Second Preview)

464: Scoped Values (Second Preview)

Deprecations

HotSpot JVM

449: Deprecate the Windows 32-bit x86 Port for Removal (21)

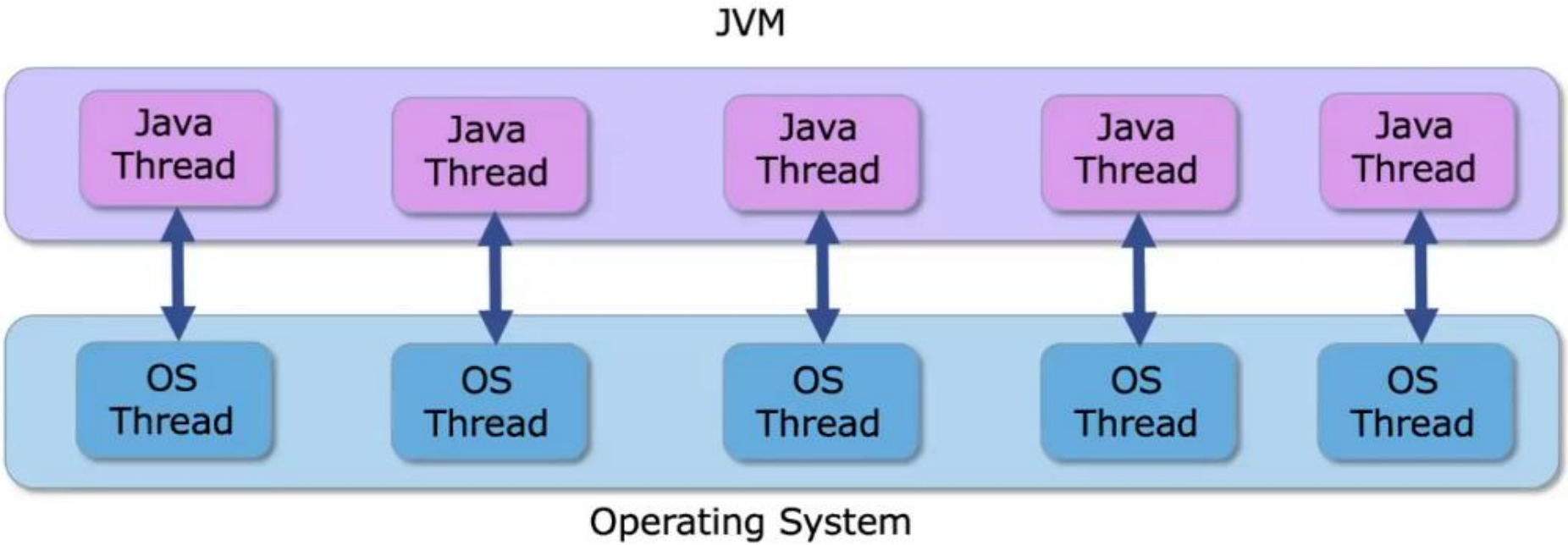
Libraries

421: Deprecate Finalization for Removal (18)

Threads in Java

Fundamental part of Java that makes it simple to write concurrent code

- More than one thing happening in parallel

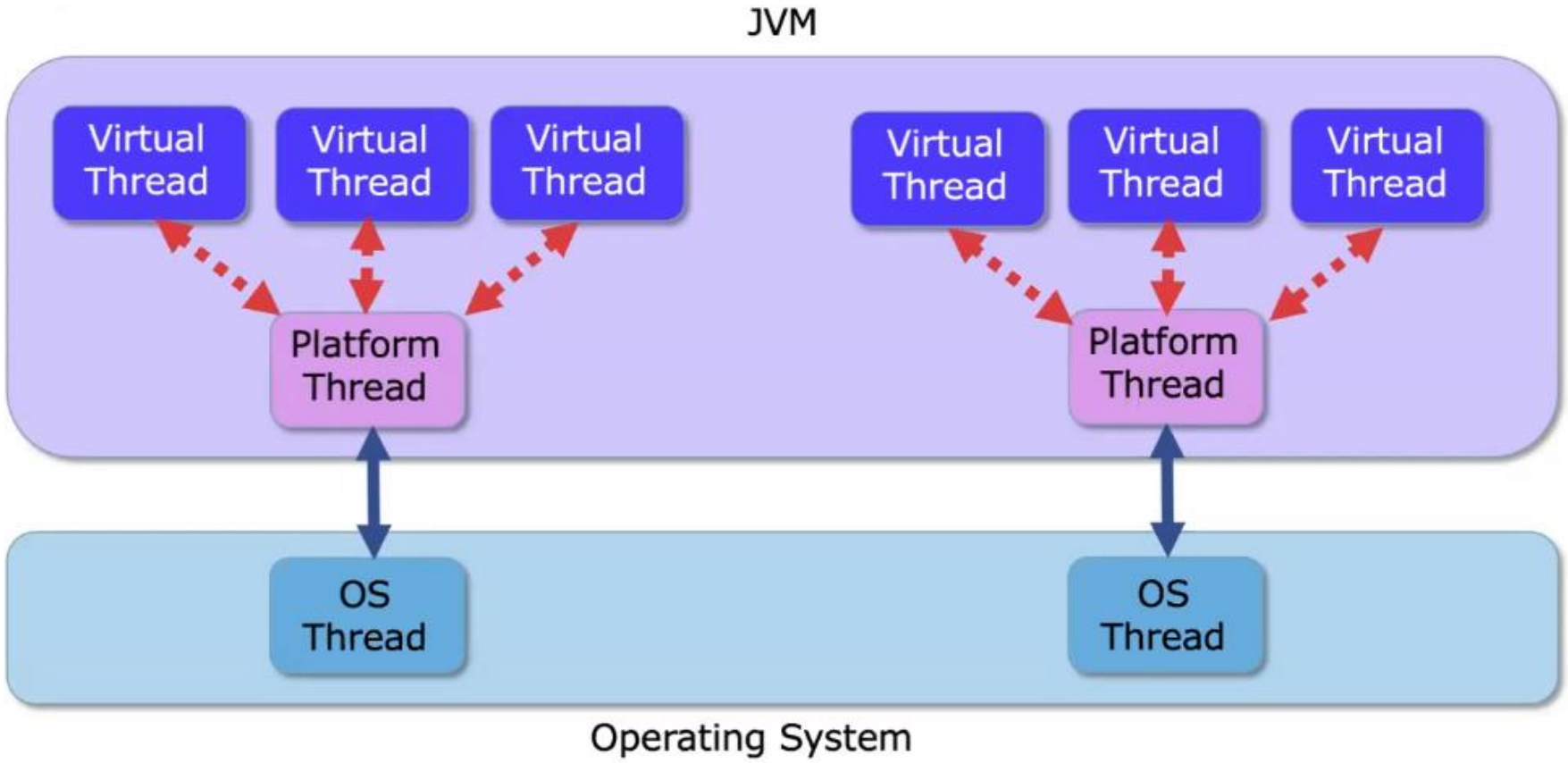


Virtual threads

- Threads enable simple concurrency in Java
- Creating a Thread object allows us to use an operating system thread, which is good
- The problem is scalability
  - Trying to start thousands of threads won't work
  - Just the memory (stack) for a thread is 1MB
- For many web applications we use thread-per-request as a programming model
- Each request often spends most of its time waiting for IO (database, network, file, etc.)

- Virtual threads allows one operating system (platform) threads to be shared by many Java threads
  - Context switching is transparent to developer
  - Each virtual threads is only few KB
  - Massively more scalable

Virtual threads



Virtual threads

- Platform threads and virtual threads both use Thread
  - The difference is how the threads are created

```
var vt = Thread.ofVirtual(); // or .ofPlatform()

var t = vt.name("MyThread").start(() -> System.out.println("Virtual World"));

var t = vt.name("MyThread").unstarted(() -> System.out.println("Virtual World"));

var tf = vt.factory();
var vt2 = tf.newThread (() -> System.out.println("Virtual World"));
```

Virtual threads

- They are not a magic bullet
  - Switching from platform threads to virtual threads will not guarantee better performance
  - It may deliver better scalability

- Considerations for virtual threads
  - Don't use them for CPU intensive tasks (virtual threads are not time sliced or preempted by the JVM)
  - Don't use synchronized blocks or methods with virtual threads
  - Don't call Thread.yield() from a virtual thread

PREVIEW

String Templates (JEP 459)

- There are many ways to build strings in Java where you want to insert values
  - String concatenation
  - StringBuilder
  - String: :format and String: : formatted
  - java.text.MessageFormat
- All have disadvantages

String Templates

- Template expressions (new to Java)
- Uses STR, a template processor
  - STR is a public static final field imported automatically into every Java source file

```
String firstName = "Tomaz";
String lastName = "Cerar";
String fullName = STR."\{firstName} \{lastName}"; // "Tomaz Cerar"
String sortName = STR."\{lastName}, \{firstName}"; // "Cerar, Tomaz
```

* There might be some bigger changes coming

String Templates

More powerful than just string insertion

- They can perform arithmetic

```
int x = 10, y = 20;
String s = STR."\{x} + \{y} = \{x + y}"; // "10 + 20 = 30"
```

They can call methods

```
String s = STR."We are here \{
// Let's use the new B pattern in DateTimeFormatter
        DateTimeFormatter
                .ofPattern("B")
                .format(LocalTime.now())
        } learning Java"; // "We are here in the afternoon learning Java"
```

Pattern matching instanceof And Records

- Pattern matching a record

```
record Point(double x, double y) { }

public void pythagoras(Object o) {
    if (o instanceof Point p) {
        double x = p.x();
        double y = p.y();
        System.out.println("Hypotenuse = " + Math.sqrt((x * x) + (y * y));
    }
}
```

Record patterns

- Record patterns are a deconstruction pattern
- Deconstruction patterns only work with records

```
public void pythagoras(Object o) {
    if (o instanceof Point(double x, double y))
        System.out.println("Hypotenuse = " + Math.sqrt((x*x) + (y*y)));
}
```

Patterns are Composable

```
interface Rectangle {};
record Point(double x, double y) {};
enum Colour { RED, GREEN, BLUE };

record ColourPoint(Point p, Colour c) {};
record ColourRectangle(ColourPoint topLeft,
                       ColourPoint bottomRight) implements Rectangle { ... };

public void printColour(Rectangle r) {
   if (r instanceof ColourRectangle(ColourPoint topleft, ColourPoint bottomRight))
      System.out.println(topleft.c());

}
```

Patterns are Composable

```
public void printColour(Rectangle r) {
    if (r instanceof ColourRectangle(ColourPoint(Point p, Colour c), ColourPoint br))
        System.out.println(c);
}
```

```
public void printTopLeftX(Rectangle r) {
    if (r instanceof ColourRectangle(ColourPoint(Point(double x, double y), Colour c), ColourPoint br))
        System.out.println(x);
}
```

Unnamed Variables and Patterns (22)

```
public void printTopLeftX(Rectangle r) {
    if (r instanceof ColourRectangle(ColourPoint(Point(var x, var y), var c), var br))
        System.out.println(x);
}
```

We don't care about y, c or br so why specify them at all?

- The compiler needs to know they're there to determine the correct record
- Now, we can use a single underscore instead

```
public void printTopLeftX(Rectangle r) {
    if (r instanceof ColourRectangle(ColourPoint(Point(var x, _), _), _))
        System.out.println(x);
}
```

Pattern Matching For switch (JEP 441)

- Switch is limited on what types you can use (Integral values, Strings, enumerations)
- Expanded to allow type patterns to be matched

```
void typeTester(Object o) {
    switch (o) {
        case null -> System.out.println("Null type");
        case String s -> System.out.println("String: " + s);
        case Color c -> System.out.println("Color with RGB: " + c.getRGB());
        case int[] ia -> System.out.println("Array of ints, length" + ia.length);
        default -> System.out.println(o.toString());

    }
}
```

Pattern Matching For switch

- Null is special
- case null can be used in all switch statements & expressions
  - If not included, it will be added by compiler

```
void typeTester(Object o) {
    switch (o) {
        case String s -> System.out.println("String: " + s);
        case Color c -> System.out.println("Color with RGB: " + c.getRGB());
        case int[] ia -> System.out.println("Array of ints, length" + ia.length);
        default -> System.out.println("Bad input!");
    }
}
```

Pattern Matching For switch

- Null is special
- case null can be used in all switch statements & expressions
  - If not included, it will be added by compiler

```
void typeTester(Object o) {
    switch (o) {
        case null -> throw new NullPointerException(); // Added by compiler
        case String s -> System.out.println("String: " + s);
        case Color c -> System.out.println("Color with RGB: " + c.getRGB());
        case int[] ia -> System.out.println("Array of ints, length" + ia.length);
        default -> System.out.println("Bad input!");
    }
}
```

Pattern Matching For switch

- Null is special
- case null can be used in all switch statements & expressions
  - If not included, it will be added by compiler

```
void typeTester(Object o) {
    switch (o) {
        case String s -> System.out.println("String: " + s);
        case Color c -> System.out.println("Color with RGB: " + c.getRGB());
        case int[] ia -> System.out.println("Array of ints, length" + ia.length);
        case null, default -> System.out.println("Bad input!");
    }
}
```

Pattern Matching for switch (completeness)

- Pattern switch statements (and expressions) must be exhaustive
  - All possible values must be handled

```
void typeTester(Object o) {
   switch (o) {
      case String s -> System.out.println("String: " + s);
      case Integer i -> System.out.println("Integer with value " + i.intValue());
   }
}
```

// results in: 'switch' statement does not cover all possible input values

Pattern Matching for switch (completeness)

```
public sealed class Shape permits Triangle, Square, Pentagon { ... }

void typeTester(Shape shape) {
   switch (shape) {
      case Triangle t -> System.out.println("It's a triangle");
      case Square s -> System.out.println("It's a square");
      case Pentagon p -> System.out.println("It's a pentagon");
      case Shape s -> System.out.println("It's a shape");
   }
}
```

Guarded Patterns

```
void shapeTester(Shape shape) { // Using previous sealed class example
   switch (shape) {
      case Triangle t when t.area() > 25 -> System.out.println("It's a big triangle");
      case Triangle t -> System.out.println("It's a small triangle");
      case Square s -> System.out.println("It's a square");
      case Pentagon p -> System.out.println("It's a pentagon");
      case Shape s -> System.out.println("It's a shape");
   }
}
```

GuardedPattern:

```
   PrimaryPattern when ConditionalAndExpression
```

// Only works for patterns, no case "Foo" when somethingElse() -> ...

Pattern Dominance

Less specific cases must not hide more specific cases

```
void typeTester(Shape shape) {
    switch (shape) {
        case Shape s -> System.out.println("It's a shape");
        case Triangle t -> System.out.println("It's a triangle");
        case Square s -> System.out.println("It's a square");
        case Pentagon p -> System.out.println("It's a pentagon");
    }
}
```

Pattern Dominance

Less specific cases must not hide more specific cases

```
void typeTester(Shape shape) {
    switch (shape) {
        case Shape s -> System.out.println("It's a shape");
        case Triangle t -> System.out.println("It's a triangle");
        case Square s -> System.out.println("It's a square");
        case Pentagon p -> System.out.println("It's a pentagon");
    }
}
```

Pattern Dominance

```
void shapeTester(Shape shape) {
    switch (shape) {
        case Triangle t -> System.out.println("It's a small triangle");
        case Triangle t when t.area() > 25 -> System.out.println("It's a big triangle");
        case Square s -> System.out.println("It's a square");
        case Pentagon p -> System.out.println("It's a pentagon");
        case Shape s -> System.out.println("It's a shape");
    }
}
```

PREVIEW

Implicitly Declared Classes and Instance Main Methods (JEP463)

- Writing your first Java program requires lots of boilerplate code (and understanding)
- There are many concepts embedded in this simple piece of code

```
public class MyFirstClass {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

Implicitly Declared Classes and Instance Main Methods

- Let's make it easier
- Firstly, introduce instance main method

```
class MyFirstClass {
    void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

- Secondly add unnamed classes

```
void main(String[] args) {
    System.out.println("Hello World!");
}
```

Launch Multi-File Source-Code Programs (22)

- Expands on JDK11 launch single source-code programs
- Allows to write simple scripts without compiling
- JBang could be even better
- Still limitations with loading dependencies

Libraries

Sequenced Collections (JEP 431)

- Some collections have a defined order
  - `List, Deque, SortedSet`
- `List` and `Deque` implement Collection that has no defined order
- Similarly, `SortedSet` inherits from `Set`, which is not ordered by definition
- There is no common supertype that defines an order. Until now

```
interface SequencedCollection<E> extends Collection<E> {
   SequencedCollection<E> reversed(); // new method
   // methods promoted from Deque
   void addFirst(E);
   void addLast(E);
   E getFirst();
   E getLast();
   E removeFirst();
   E removeLast();
}
```

Foreign Function and Memory API (JEP 442)

- Part of project Panama
  - A replacement for JNI
- Allows simplified interaction with libraries not written in Java (or bytecode compiled languages)
- Also allows interaction with native memory
  - Without resorting to internal APIs like sun.misc.Unsafe

- Interestingly, jextract is not included in the JDK

Foreign Function Interface API

- Provides statically-typed, pure-Java access to native code
  - Works in conjunction with the Foreign Memory Access API
  - Initially targeted at C native code. C++ should follow
- More powerful when combined with Project Panama jextract command

```
public static void main(String[] args) throws Throwable {
        var linker = CLinker.getInstance();
        var lookup = LibraryLookup.ofDefault();
        // get a native method handle for 'getpid' function
        var getpid = linker.downcallHandle(lookup.lookup("getpid").get(),
                MethodType.methodType(int.class),
                FunctionDescriptor.of(CLinker.C_INT));

        System.out.println((int)getpid.invokeExact());
}
```

Foreign Memory API

- Accessing memory outside the heap

```
String[] javaStrings = {"mouse", "cat", "dog", "car"};
try (Arena offHeap = Arena.ofConfined()) {
   MemorySegment pointers = offHeap.allocateArray(ValueLayout.ADDRESS,
javaStrings.length);
   for (int i = 0; i < javaStrings.length; i++) {
      MemorySegment cString = offHeap.allocateUtf8String(javaStrings[i]);
      pointers.setAtIndex(ValueLayout.ADDRESS, i, cString);

   }
   // Use Foreign Function API to manipulate the strings in some way
   for (int i = 0; i < javaStrings.length; i++) {
      MemorySegment cString = pointers.getAtIndex(ValueLayout.ADDRESS, i);
      javaStrings[i] = cString.getUtf8String(0);
   }
}
```

PREVIEW

Scoped Values (JEP 464)

- An alternative to thread-local variables
- Values (i.e. immutable) rather than variable
- Each thread can have its own value
- Not necessary to copy a scoped value for each child thread
  - Unlike thread-local variables they cannot be changed so don't need their own copy

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

ScopedValue.runWhere(NAME, "Simon", () -> useTheString());
```

Unstructured Concurrency

```
Response handle() throws ExecutionException, InterruptedException {
        Future<String> user = executorSvc.submit(() -> findUser());
        Future<Integer> order = executorSvc.submit(() -> fetchOrder());
        String theUser = user.get(); // Join findUser
        int theOrder = order.get(); // Join fetchOrder
        return new Response(theUser, theOrder);
}
```

- Each subtask can succeed or fail independently
- What happens if
  - findUser throws exception?
  - the handle method gets interrupted
  - findUser takes a long time to complete but fetchOrder is fast

PREVIEW

Structured Concurrency (JEP 462)

```
Response handle() throws ExecutionException, InterruptedException {
    try (var scope = new StructuredTaskScope.ShutdownOnFailure()) {
          Supplier<String> user = scope.fork(() -> findUser());
          Supplier<Integer> order = scope.fork(() -> fetchOrder());
          scope.join() // Join both subtasks
            .throwIfFailed(); // ... and propagate errors

          // Here, both subtasks have succeeded, so compose their results
          return new Response(user.get(), order.get());
}
```

- Error-handling with short-circuiting
- Cancellation propagation
- Clarity
- Observability

PREVIEW

Stream Gatherers (JEP 461)

- Enhances the Stream API to support custom intermediate operations
  - stream pipelines can transform data in ways not possible before
- Beter flexibility for custom stream operations
  - Improved accumulation

PREVIEW

Vector API (JEP 460)

- Not to be confused with the Vector collection class
- API to express vector computations
  - Compile at runtime to optimal hardware instructions
  - Deliver superior performance to equivalent scalar operations
  - Incubator module

- Ideally, this would not be necessary
  - Compiler should identify where vector operations can be used

Vector API

```
void vectorAdd(int[] a, int[] b, int[c]) {
   VectorSpecies<Integer> species = IntVector.SPECIES_256; // Or SPECIES_PREFERRED

   // Assume all arrays are of the same size
   for (int i = 0; i < a.length; i += SPECIES.length()) {
      VectorMask<Integer> m = SPECIES.indexInRange(i, a.length);
      IntVector va = IntVector.fromArray(SPECIES, a, i, m);
      IntVector vb = IntVector.fromArray(SPECIES, b, i, m);
      IntVector vc = va.mul(vb);
      vc.intoArray(c, i, m);
   }
}
```

Small & fun stuff

Unicode Emoji Properties

- `isEmoji(int codePoint)`
- `isEmojiPresentation(int codePoint)`
- `isEmojiModifier(int codePoint)`
- `isEmojiModifierBase(int codePoint)`
- `isEmojiComponent(int codePoint)`
- `isExtendedPictographic(int codePoint)`

The java.net.http.HttpClient Is Now AutoCloseable

The following methods have been added to the API:

- `void close()`
- `void shutdown()`
- `void shutdownNow()`
- `boolean awaitTermination(Duration duration)`
- `boolean isTerminated()`

Unix domain socket

```
UnixDomainSocketAddress address = UnixDomainSocketAddress.of("/tmp/socket.sock");

UnixDomainSocketChannel channel = UnixDomainSocketChannel.open(address);
```

Miscellaneous

- JEP 449: Deprecate the Windows 32-bit x86 port for removal
- JEP 451: Prepare to disallow the dynamic loading of agents
- JEP 452: Key encapsulation mechanism
- JEP 439: Generational ZGC

Conclusions

- JDK 21 is the latest LTS release
- JDK 22 is a nice “Service Pack”
- Includes plenty of interesting new features
  - + few in preview
- Virtual threads is an appealing feature for both developers and DevOps
- Adoption will most likely be faster than JDK 17
- Anyone on JDK 8 or 11 will probably skip JDK 17 and move directly to JDK 21

Q & A