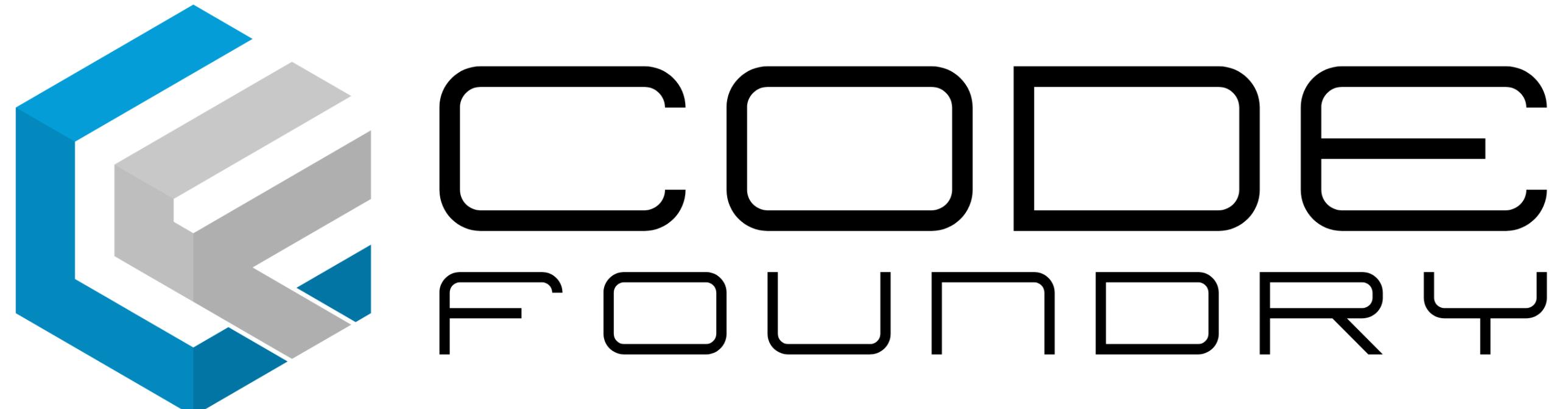


Java 18+

features

Adolfo Benedetti ~ 13 jan 2022 ~ Code Foundry



Agenda

#Java 18 Features: (Mar, 2022)

- UTF-8 by Default
- Simple Web Server
- Code Snippets in Java API Doc
- Reimplement Core Reflection with Method Handles
- Vector API
- Internet-Address Resolution SPI
- Foreign Function & Memory API
- Pattern Matching for switch (preview 2)
- Deprecated method finalize

Agenda

#Java 19 Features: (Sep, 2022)

- Record Patterns (preview)
- Linux/RISC-V Port
- Foreign Function & Memory API
- Virtual Threads
- Vector API (Fourth Incubator)
- Pattern Matching for the switch (preview 3)
- Structured Concurrency (Incubator)

Agenda

#Java 20 features: (21th of March 2023)

- Scoped values (incubator)
- Record pattern (preview 2)
- Pattern matching for switch (preview 4)
- Foreign Function & memory API (preview 2)
- Virtual Threads (preview 2)
- Structured concurrency (second incubator)

UTF-8 by Default

Java 18

- UTF-8 zal de standaard encoding zijn voor alle besturingssystemen, waardoor de prestaties worden verbeterd en fouten veroorzaakt door onverenigbare tekenreeksen worden verminderd.
- Om te controleren, gebruikt: **java -XshowSettings:properties -version 2>&1 | grep file.encoding**
- Om te wijzigen, gebruikt: **-Dfile.encoding=UTF-16**
- Beïnvloede klassen:
 - Stream Reader and Writer
 - File Reader and Writer
 - **Formatter** and **Scanner**
 - **URLEncoder** and **URLDecoder**
- **System.out** en **System.err** zullen dezelfde encoding gebruiken als de terminal
- Controleer de terminal encoding met: **Console.charset()**
- Deze functie beïnvloedt bestaande Java-applicaties, eventueel moeten plannen om de code aan te passen.

Simple Web Server

Java 18

- Java 18's Simple Web Server, JEP 408, is een nieuwe tool die is toegevoegd in JDK 18, in de **jdk.httpserver** module.
- Deze web server is een minimale HTTP statische bestandsserver en ondersteunt alleen de **GET** en **HEAD** request methoden.
- Het is een out-of-the-box tool met gemakkelijke opzet en minimale functionaliteit die je snel aan de slag kunt laten gaan en je op de taak kunt concentreren.
- Het is niet geschikt voor productie, maar juist handig voor prototyping, ad-hoc coding en testing.

Simple Web Server

Java 18

Code Snippets in Java API Doc

Java 18

```
class SomeAppenderVersion {
    /**
     * The <code>@snippet</code> will render something like this:
     *
     * <pre>
     * // sum two numbers
     * int sum = MathOperationBuilder.<strong>builder</strong>()
     *     .withA(...)
     *     .withB(...)
     *     .<strong>sum</strong>();
     * {@link System#out System.out}.println(sum);
     * </pre>
     */
    no usages
    public void methodWithSimpleDoc() {
```

Code Snippets in Java API Doc

Java 18

```
/**  
 * Builder to help execute math operation =).  
 * Usage:  
 * {@snippet :  
 * // sum two numbers  
 * int sum = MathOperationBuilder.builder() // @highlight substring="builder"  
 * .withA(1) // @replace regex="\d+" replacement="..."  
 * .withB(2) // @replace regex="\d+" replacement="..."  
 * .sum(); // @highlight substring="sum"  
 * System.out.println(sum); // @link substring="System.out"  
 * target="System#out"  
 *}  
 */  
no usages  
public MathOperationBuilder methodWithSnippetDoc() {
```

Code Snippets in Java API Doc

Java 18

methodWithSimpleDoc

```
public void methodWithSimpleDoc()
```

The @snippet will render something like this:

```
// sum two numbers
int sum = MathOperationBuilder.builder()
    .withA(...)
    .withB(...)
    .sum();
System.out.println(sum);
```

methodWithSnippetDoc

```
public CodeSnippetInJavaDoc.MathOperationBuilder methodWithSnippetDoc()
```

Builder to help execute math operation =). Usage:

```
// sum two numbers
int sum = MathOperationBuilder.builder()
    .withA(...)
    .withB(...)
    .sum();
System.out.println(sum);
```

Code Snippets in Java API Doc

Java 18

Reimplement Core Reflection with Method Handles

Java 18

- JEP 416 heeft als doel om **java.lang.reflect**. Method, Constructor en Field te herimplementeren op basis van **java.lang.invoke** method handles.
- De doelstelling is om de onderhouds- en ontwikkelingskosten te verminderen.
- De nieuwe implementatie zal een directe uitvoering van de method handles voor specifieke reflectieve objecten bieden.

Vector API *Panama*

Java 18

- Vector API Developer Program voor Java is een set van methoden voor data-parallele operaties op gesizede vector-types voor programmeren in Java direct, zonder kennis van de onderliggende CPU.
- Het maakt efficiënt gebruik van Single Instruction, Multiple Data (SIMD) versnelling voor compute-intensieve applicaties, machine learning en AI programmeren in Java.
- Vector API verbetert de prestaties in BigData applicaties, zoals Apache Flink, Apache Spark Machine Learning libraries, Intel Big DL en deep learning training workloads.

Vector API

Java 18

- Straks bij Java 19

Internet-Address Resolution SPI

Java 18

- Java 18 zal een nieuwe **service provider interface (SPI)** introduceren voor *hostnaam* en internet adresresolutie.
- Doel is om Java-applicaties meer controle te geven over internet-adressering.
- De huidige **java.net.InetAddress** API gebruikt de native resolver van het besturingssysteem.
- De motivatie voor deze *SPI* is om:
 - Project Loom te ondersteunen, door blokkerende calls naar het besturingssysteem te vermijden.
 - Nieuwe netwerkprotocollen te integreren, zoals *DNS* over *QUIC*, *TLS* of *HTTPS*.
 - Bestaande bibliotheken aan te passen met een custom resolver.
 - Testen te vergemakkelijken door controle te krijgen over hostnaam en adresresolutie.
 - De InetAddress API zal gebruik maken van een service loader om de resolver provider te vinden, als deze niet gevonden wordt, dan zal de oorspronkelijke implementatie gebruikt worden.
- Nieuwe classes in de **java.net.spi** package zijn **InetAddressResolverProvider** en **InetAddressResolver**.

Foreign Function & Memory API

Java 18

- Het biedt een manier voor Java code om te communiceren met code en gegevens buiten de Java runtime, specifiek op dezelfde machine.
- Fix de beperkingen van de Java Native Interface (JNI) op, zoals het tijdrovende proces voor elke native library, beperkingen voor het werken met andere talen dan C en C++, en onvermogen om het Java type systeem te integreren met het C type systeem.
- Het biedt functionaliteiten zoals de mogelijkheid om buitenlandse geheugen te alloceren, buitenlandse gestructureerde geheugen te manipuleren en te benaderen, de levensduur van buitenlandse resources te beheren en buitenlandse functies aan te roepen.
- Functionaliteiten van de API:
 - Allocateren van foreign memory (**MemorySegment**, **MemoryAddress**, and **SegmentAllocator**)
 - Manipulatie en toegang tot gestructureerd foreign memory (**MemoryLayout**, **VarHandle**)
 - Beheer van de levensduur van foreign resources (**ResourceScope**)
 - Aanroepen van foreign functions (**SymbolLookup**, **CLinker**, and **NativeSymbol**)

Foreign Function & Memory API

Java 18

- Straks bij Java 19

Pattern Matching for switch (preview 2) Amber

Java 18

- Pattern matching for switch expressions en statements is een nieuwe feature in Java 18 die het mogelijk maakt om switch statements of expressies te gebruiken met patroon matching ipv alleen constanten.

```
public static double getPerimeter(Shape shape) throws IllegalArgumentException {  
    return switch (shape) {  
        case Rectangle r → 2 * r.length() + 2 * r.width();  
        case Circle c → 2 * c.radius() * Math.PI;  
        default → throw new IllegalArgumentException("Unrecognized shape");  
    };  
}
```

Pattern Matching for switch (preview 2) Amber

Java 18

- Je kan ook gebruik maken van pattern matching voor de selector expression van de switch statement. Hier een voorbeeld waarbij het type van obj bepaald wordt met patroon matching

```
static void typeTester(Object obj) {  
    switch (obj) {  
        case null → System.out.println("null");  
        case String s → System.out.println("String");  
        case Color c → System.out.println("Color with " + c.values().length + " values");  
        case Point p → System.out.println("Record class: " + p.toString());  
    }  
}
```

Pattern Matching for switch (preview 2) Amber

Java 18

Deprecated method finalize

Java 18

```
public class FileReader {
    private BufferedReader reader;

    public FileReader() {
        InputStream input = this.getClass().getClassLoader().getResourceAsStream
            ("sample.txt");
        this.reader = new BufferedReader(new InputStreamReader(input));
    }

    public String readFirstLine() throws IOException {
        String firstLine = reader.readLine();
        return firstLine;
    }

    @Override
    public void finalize() {
        try {
            reader.close();
            System.out.println("BufferedReader close in the finalize method");
        } catch (IOException e) {

        }
    }
}
```

Record Patterns (preview)

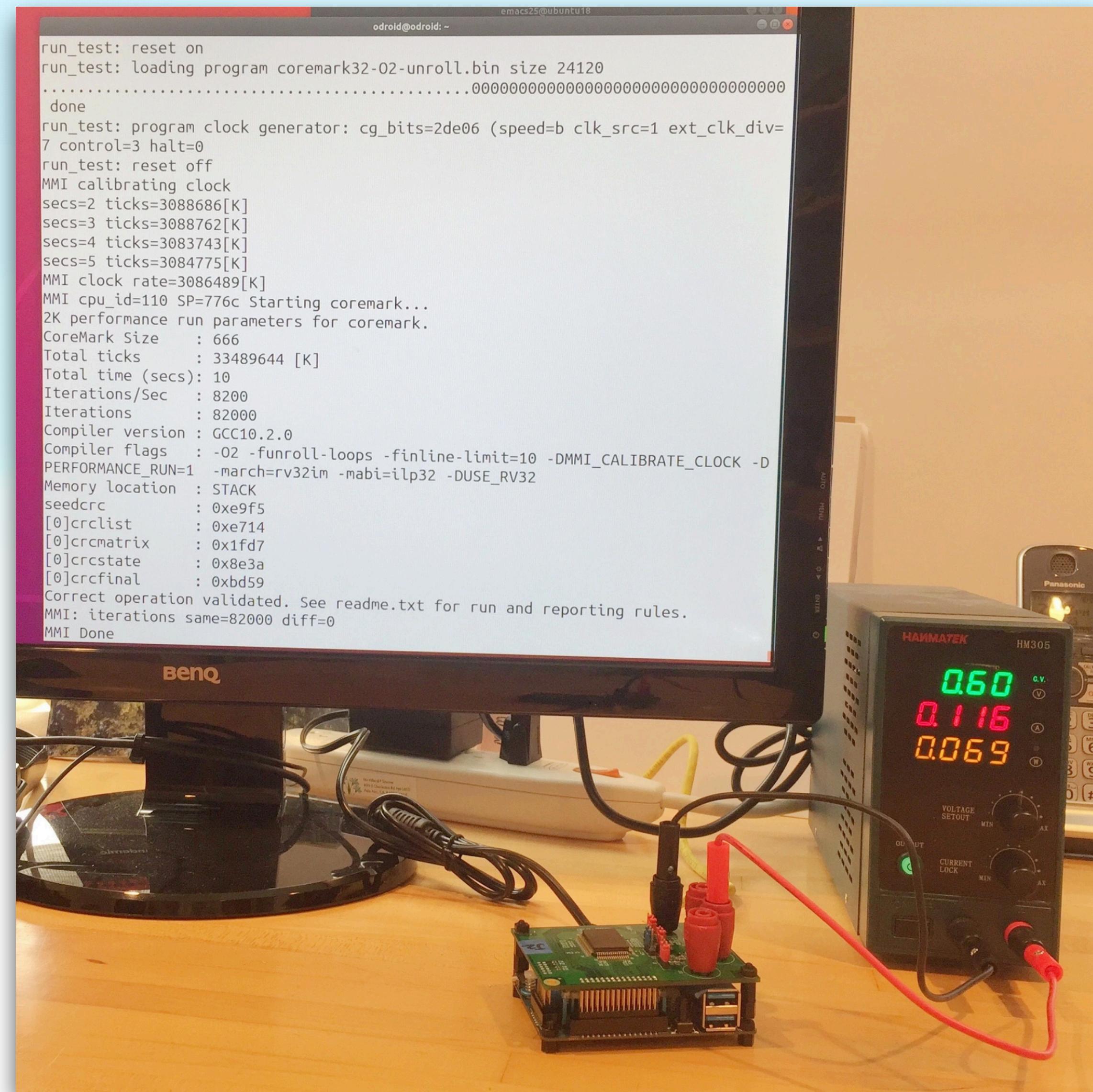
Java 19

- Nu is het mogelijk om de waarden van een 'record' uit elkaar te halen in een switch statement of expressie.
- Dit wordt gedaan door gebruik te maken van een specifieke syntax voor een 'record pattern', bijvoorbeeld: **Point(int x, int y)**.
- Hierbij worden de variabelen **int x** en **int y** geinitialiseerd met de waarden van de 'accessor method' van de record, in dit geval **Point.x()** en **Point.y()**. De namen van de variabelen hoeven niet overeen te komen met de namen van de componenten van de record.
- Daarnaast is het nu ook mogelijk om type patterns en record patterns te gebruiken in combinatie met de **instanceof** operator. Hierdoor is het bijvoorbeeld mogelijk om te checken of een object een specifieke record is en direct de componenten hiervan te extraheren, bijvoorbeeld: **o instanceof Point(int x, int y)**.
- Let wel op, dat een '*null*' waarde geen match is voor een 'record pattern'.

Record Patterns (preview)

Java 19

Linux/RISC-V Port



Foreign Function & Memory API

Java 19

```
jextract --output classes -t org.unix of met de source jextract --source --output src -t org.unix
Run: `java --enable-preview --source 19 QsortMain.java`

public class QsortMain {

    public static void main(String[] args) {
        int[] unsortedArray = new int[]{0, 9, 3, 4, 6, 5, 1, 8, 2, 7};

        try (MemorySession session = MemorySession.openConfined()) {
            // Allocate off-heap memory and store unsortedArray in it
            MemorySegment array = session.allocateArray(
                ValueLayout.JAVA_INT,
                unsortedArray);
        }
    }
}
```

Foreign Function & Memory API

Java 19

Virtual Threads Loom

Java 19

- Virtual threads zijn gewoon threads, maar het aanmaken en blokkeren ervan is **goedkoop**: ze worden beheerd door de Java runtime en zijn **niet één-op-één** wrappers van **OS threads**; ze zijn geïmplementeerd in user-space in de JDK.
- De Thread API wordt behouden en de-emphasized, waardoor virtual threads gewoon Threads zijn en elke bibliotheek die Thread al kent, ook virtual threads kent.
- Virtual threads kunnen worden geblokkeerd wanneer ze native code (JNI) aanroepen of binnen een gesynchroniseerde block of methode zijn, het is aanbevolen om in deze gevallen **ReentrantLock** te gebruiken.
- Virtual threads zijn compatibel met alles wat gebouwd is op de JDK.
- Virtual threads kunnen worden getest met een early access build en door VM opties zoals **-Djdk.defaultScheduler.parallelism=N**.
- Virtual threads kunnen worden aangemaakt met **Thread.startVirtualThread()** of **Thread.ofVirtual()**.
- Java VisualVM kan worden gebruikt om het gebruik van virtual threads te observeren.

Vector API (Fourth Incubator) Panama

Java 19

- De Vector API biedt een mechanisme voor het schrijven van cross-platform data-parallelle algoritmen in Java, zoals complexe wiskundige en array-gebaseerde operaties. De Vector API biedt een draagbare API voor het uitdrukken van vectormathematische berekeningen. De eerste iteratie van de API werd voorgesteld door JEP 338 en geïntegreerd in Java 16. De tweede incubator, JEP 414, maakt deel uit van Java 17. Een derde incubator is in uitvoering en is momenteel gericht op Java 18 als JEP 417.
- Dit werk is onderdeel van Java's Project Panama, dat vele van de verbindingen tussen de JVM en niet-Java-API's (ook wel buitenlandse API's genoemd) versterkt, waaronder het maken van vectormathematische intrinsieken onderdeel van de HotSpot JVM (en dus onderdeel van core Java).
- De Vector API heeft als doel om de situatie te verbeteren door een manier te bieden om complexe vectoralgoritmen in Java te schrijven, met behulp van de bestaande HotSpot auto-vectorizer maar met een gebruikersmodel dat vectorisatie veel voorspelbaarder en robuuster maakt. Handgemaakte vectorlussen kunnen hoogwaardige algoritmen uitdrukken, zoals vectoriale hashCode of gespecialiseerde arrayvergelijkingen, die een auto-vectorizer misschien nooit optimaliseert. Tal van domeinen kunnen profiteren van deze expliciete vector API, waaronder machine learning, lineaire algebra, cryptografie, financiën en code binnen de JDK zelf.

Vector API (Fourth Incubator) *Panama*

Java 19

```
    /**
     * Hier is een eenvoudige scalaire berekening over elementen van arrays,
     * aangenomen dat de array argumenten dezelfde lengte hebben.
     */
    void scalarComputation ( float[] a, float[] b, float[] c){
        for (int i = 0; i < a.length; i++) {
            c[i] = (a[i] * a[i] + b[i] * b[i]) * -1.0f;
        }
    }
```

Vector API (Fourth Incubator) *Panama*

Java 19

```
    /**
     * Hier is een equivalente vectorberekening, met behulp van de Vector API, die
     * parallel zal worden uitgevoerd met behulp van de SIMD-capaciteiten van de CPU
     * (indien aanwezig in de hardware).
     */
    static final VectorSpecies<Float> SPECIES = FloatVector.SPECIES_PREFERRED;

    void vectorComputation ( float[] a, float[] b, float[] c){
        int i = 0;
        int upperBound = SPECIES.loopBound(a.length);
        for ( ; i < upperBound; i += SPECIES.length()) {
            // FloatVector va, vb, vc;
            var va = FloatVector.fromArray(SPECIES, a, i);
            var vb = FloatVector.fromArray(SPECIES, b, i);
            var vc = va.mul(va)
                      .add(vb.mul(vb))
                      .neg();
            vc.intoArray(c, i);
        }
        for ( ; i < a.length; i++) {
            c[i] = (a[i] * a[i] + b[i] * b[i]) * -1.0f;
        }
    }
}
```

Pattern Matching for the switch (preview 3)

Java 19

- Kleine verbeteringen vanuit Java 18:
- Guarded pattern van `&&` na `when` keyword
 - definition: guard is the boolean expression, guarded pattern is the case with guard
 - guarded pattern: `case Hero h when h.getCity() == NEW_YORK`
 - guard: `h.getCity() == NEW_YORK`

Pattern Matching for the switch (preview 3)

Java 19

Structured Concurrency (Incubator)

Java 19

- Structured concurrency is een manier om multithreaded code eenvoudiger te maken om te schrijven, te lezen en onderhouden door meerdere taken die in verschillende threads uitgevoerd worden te groeperen als één eenheid van werk. Kort gezegd gaat het erom de eenvoud van single-threaded code te behouden bij multi-threaded workflows waar mogelijk.
- Structured concurrency werkt perfect samen met virtuele threads, waar virtuele threads een overvloed aan threads bieden en structured concurrency ervoor zorgt dat deze correct en robuust gecoördineerd worden.
- Naast de eenvoud van de code, is hier het echt krachtige de vereenvoudigde manier om foutscenario's van verschillende uitvoeringen die in volledig verschillende (potentieel virtuele) threads worden uitgevoerd, te behandelen.
- VM-flags die nodig zijn om virtuele threads en structured concurrency: **--enable-preview --add-modules jdk.incubator.concurrent**

Structured Concurrency (Incubator)

Java 19

Scoped values (incubator)

Java 20

- Java 20 introduceert Scoped Values, een incubator feature voor het delen van onveranderlijke data tussen threads. Dit is een alternatief voor Thread Local Variables en is speciaal ontworpen om te werken met Virtual Threads.
- Scoped Values biedt een programmamodel om data te delen binnen een thread en met child threads, met een duidelijke levensduur van de data en alleen toegang voor legitieme callees. Dit maakt data-sharing veilig en efficiënt tussen componenten.
- Je kan een **ScopedValue** creëren door gebruik te maken van de methode **ScopedValue.newInstance()**, en vervolgens gebruik maken van **ScopedValue.where** om data toe te wijzen en run of call om de data te binden aan de huidige thread.
- Voor het delen van data tussen een thread en zijn child thread, kan men gebruik maken van de Structured Concurrency API en de klasse **StructuredTaskScope**.

Scoped values (incubator)

Java 20

Record pattern (preview 2)

Java 20

- Toegevoegd ondersteuning voor het afleiden van type argumenten van generieke record patterns
 - Nu kan het generieke type inferred worden
 - Gegeven `'record Decorator<T>(T t) {}'` en de variabele `'Decorator<Decorator<String>> wr'` kan het generieke type van het record pattern inferred worden in `'w insteadof Decorator(Decorator(var s))'`
- Toegevoegd ondersteuning voor record patterns in de header van een enhanced for statement
 - `'for (Point(var x, var y) : shapePoints)'`
- Verwijderd ondersteuning voor genoemde record partners.

Record pattern (preview 2)

Java 20

Pattern matching for switch (preview 4)

Java 20

- Toegevoegd ondersteuning voor inference of type arguments voor generic record pattern

Pattern matching for switch (preview 4)

Java 20

Foreign Function & memory API (preview 2)

Java 20

- ...

Virtual Threads (preview 2)

Java 20

- API's aangepast JEP 425:
- `Thread.join(Durantion)`
- `Thread.sleep(Duration)`
- `Thread.currentThreadId()`
- `Future.resultNow()`
- `Future.exceptionNow()`
- `Future.state()`
- `ExecutorService` extends `AutoClosable`

Structured concurrency (second incubator)

Java 20

- ...

Bonus Valhalla Project

Java 20

- Project Valhalla is een ontwikkelingsproject van Java dat zich richt op het aanpassen van de Java taal en runtime aan moderne hardware.
- De belangrijkste motivatie voor Project Valhalla is om de prestaties van Java te verbeteren door indirections die leiden tot pointer-fetches te verminderen.
- Value types zijn een belangrijk aspect van Project Valhalla. Het idee hierachter is dat ze pure data-aggregaten zijn zonder object-identiteit. Dit leidt tot snellere prestaties doordat er geen indirections meer nodig zijn.
- Value types kunnen methoden en velden hebben, interfaces implementeren en als generieke types worden gebruikt. Ze kunnen worden gezien als snellere objecten of als gedefinieerde primitieven.
- Project Valhalla biedt ook de mogelijkheid om generieke types te gebruiken zonder dat er boxed hoeft te worden.



Zijn er nog vragen?