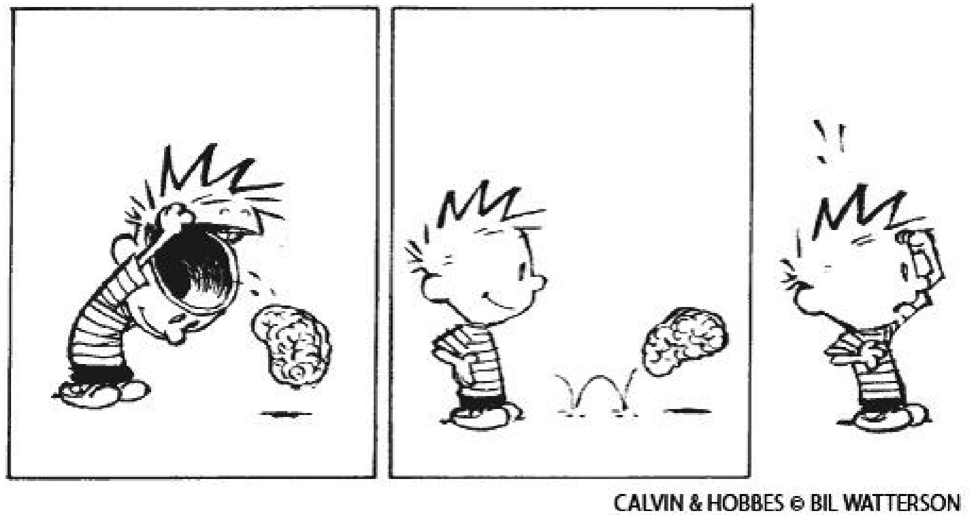
Java 21
add sparkle to your life

Rémi Forax
Université Gustave Eiffel – Sept 2023



Don’t believe what I’m saying !

Plan

Language Changes

- Unnamed class (preview)
- Pattern Matching : instanceof + switch + record pattern
- Template Processor (preview)

API Changes

- Sequenced Collection
- Virtual Threads
- Structured Concurrency (preview)

Platform integrity

Java enboarding

What if ?

Before

```
public class Main {
  public static void main(String[] args) {
    System.out.println("Java is cool !");
  }
}
```

After

```
void main() {
  System.out.println("Java is cooler !");
}
```

JEP 445: … instance main methods (preview)

Change the launcher protocol

- Look for
  - static void main(String[]) or static void main()
  - void main(String[]) or void main()
- If main() is an instance method, the constructor with no parameter is called first

JEP 445: unnamed classes … (preview)

Methods without an enclosing class are encapsulated into a top-level class with a name derived from the filename (minus .java)

```
record Message(String name) {}

void main() {
  System.out.println(new Message("hello"));
  getClass().isSynthetic();  // true
}
```

DEMO !

Pattern Matching

Open Types vs Closed Types

In libraries, we want the types to be **open** so users can implement them

In applications, we want types to be **closed** so developers knows all possible subtypes

This is almost always true

Open aka non-sealed Type

MilitaryUnit = Soldier | Carrier | ...

```
/*non-sealed*/ interface MilitaryUnit {
  int firepower();
}

record Soldier(String name, int firepower) implements MilitaryUnit {}

record Carrier(List<MilitaryUnit> units) implements MilitaryUnit {
  public int firepower() {
    return units.stream()
        .mapToInt(u -> u.firepower())
        .sum();
  }
}
```

Sealed Type

MilitaryUnit = Soldier | Carrier

```
sealed interface MilitaryUnit { }
record Soldier(String name, int firepower) implements MilitaryUnit {}
record Carrier(List<MilitaryUnit> units) implements MilitaryUnit {}
```

A good interface is an empty interface

Sealed Type operation

```
int firepower(MilitaryUnit unit) {
  if (unit instanceof Soldier soldier) {
    return soldier.firepower();
  }
  if (unit instanceof Carrier carrier) {
    return carrier.units().stream()
        .mapToInt(u -> firepower(u))
        .sum();
  }
  throw new MatchException("oops", null);
}
```

instanceof in Java 17

not typesafe !

new in Java 21 !

Pattern Matching / switch on type

Switch on type – new in Java 21

```
int firepower(MilitaryUnit unit) {
  return switch(unit) {
    case Soldier soldier -> soldier.firepower();
    case Carrier carrier -> carrier.units().stream()
        .mapToInt(u -> firepower(u))
        .sum();
  };
}
```

Will not compile if new subtype !

No default !

Record Patterns

```
int firepower(MilitaryUnit unit) {
  return switch(unit) {
    case Soldier(String name, int firepower) -> firepower;
    case Carrier(List<MilitaryUnit> units) -> units.stream()
        .mapToInt(u -> firepower(u))
        .sum();
  };
}
```

Will not compile if the data definition change !

Var Pattern

```
int firepower(MilitaryUnit unit) {
  return switch(unit) {
    case Soldier(var name, var firepower) -> firepower;
    case Carrier(var units) -> units.stream()
        .mapToInt(u -> firepower(u))
        .sum();
  };
}
```

Let the compiler infer the types

Unnamed Variable (preview)

```
int firepower(MilitaryUnit unit) {
  return switch(unit) {
    case Soldier(var _, var firepower) -> firepower;
    case Carrier(var units) -> units.stream()
        .mapToInt(u -> firepower(u))
        .sum();
  };
}
```

Use ‘_’ as a variable name (everywhere but in API)

Unnamed Pattern (preview)

```
int firepower(MilitaryUnit unit) {
  return switch(unit) {
    case Soldier(_, var firepower) -> firepower;
    case Carrier(var units) -> units.stream()
        .mapToInt(u -> firepower(u))
        .sum();
  };
}
```

Using ‘_’ as pattern

Data Oriented Programming

Sealed Types + Pattern matching enables DOP

Switch on types / instanceof

- Type pattern
- Record pattern
- Var pattern
- Unnamed pattern

Data definition is more important than code

String Template Processor

STR (preview)

```
var joe = new Soldier("Joe", 200);
var jane = new Soldier("Jane", 200);
var carrier = new Carrier(List.of(joe, jane));

System.out.println(STR."""
    Jane firepower:    \{ firepower(jane) }
    carrier firepower: \{ firepower(carrier) }
    """);
```

STR is auto-imported

String interpolation !

FMT (preview)

C-like formatting

```
var joe = new Soldier("Joe", 200);
var jane = new Soldier("Jane", 200);
var carrier = new Carrier(List.of(joe, jane));

System.out.println(FMT."""
    Jane firepower:    %04d\{ firepower(jane) }
    carrier firepower: %04d\{ firepower(carrier) }
    """);
```

FMT requires an import static java.util.FormatProcessor.FMT

My own TemplateProcessor (preview)

```
StringTemplate.Processor<String, RuntimeException> fireProcessor =
    (StringTemplate templatedString) -> {
        List<String> fragments = templatedString.fragments();
        List<Object> values = templatedString.values();
        System.out.println(STR."fragments:'\{ fragments }' values:'\{ values }'");
        ...
};

System.out.println(fireProcessor."""
    Jane firepower:    \{ jane }
    carrier firepower: \{ carrier }
    """);
```

A StringTemplate is a text separated by values

My own TemplateProcessor (2/2)

```
StringTemplate.Processor<String, RuntimeException> fireProcessor =
    (StringTemplate t) -> {
        return StringTemplate.interpolate(t.fragments(), t.values().stream()
            .map(value -> firepower((MilitaryUnit) value))
            .toList());
};

System.out.println(fireProcessor."""
    Jane firepower:    \{ jane }
      carrier firepower: \{ carrier }
      """);
```

No way to type the values :(

Performance :(

```
public String concat() {
  var message = "string template";
  return "hello " + message + " !";
}

public String with_STR() {
  var message = "string template";
  return STR."hello \{message} !";
}

static final StringTemplate.Processor<String, RuntimeException> STR_INTERPOLATE =
    StringTemplate::interpolate;

public String with_interpolate() {
  var message = "string template";
  return STR_INTERPOLATE."hello \{message} !";
}
```

	score ± error
concat	**5.042** ± 0.137 ns/op
with_STR	**5.037** ± 0.111 ns/op
with_interpolate	**12.509** ± 0.049 ns/op

Sequenced Collections

Goals

Add useful methods

- List.getFirst() / getLast()
- for(var item : list.reversed()) { … }
- LinkedHashSet.getFirst() / getLast()
- for(var item : linkedHashSet.reversed()) { … }

SequencedCollection

Collection with an order (insertion, sorted, access?)

Added methods :

- getFirst()/getLast()
- addFirst/addLast/removeFirst()/removeLast()
- SequencedCollection reversed()

This is a view !

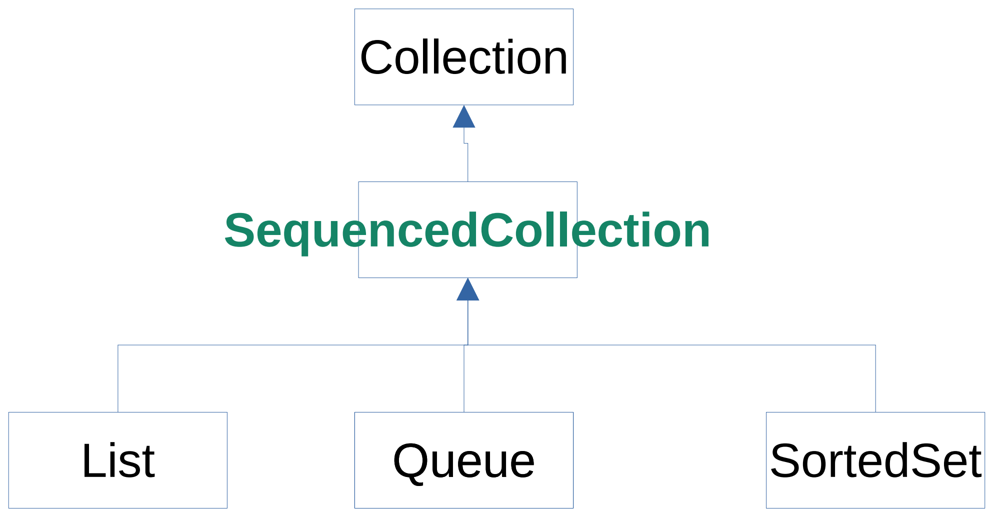
Example

```
var joe = new Soldier("Joe", 200);
var jane = new Soldier("Jane", 200);
var carrier = new Carrier(List.of(joe, jane));

System.out.println("first " + carrier.units().getFirst());
System.out.println("last " + carrier.units().getLast());

for (var unit: carrier.units().reversed()) {
   System.out.println("unit " + unit);
}
```

for loop in reverse order

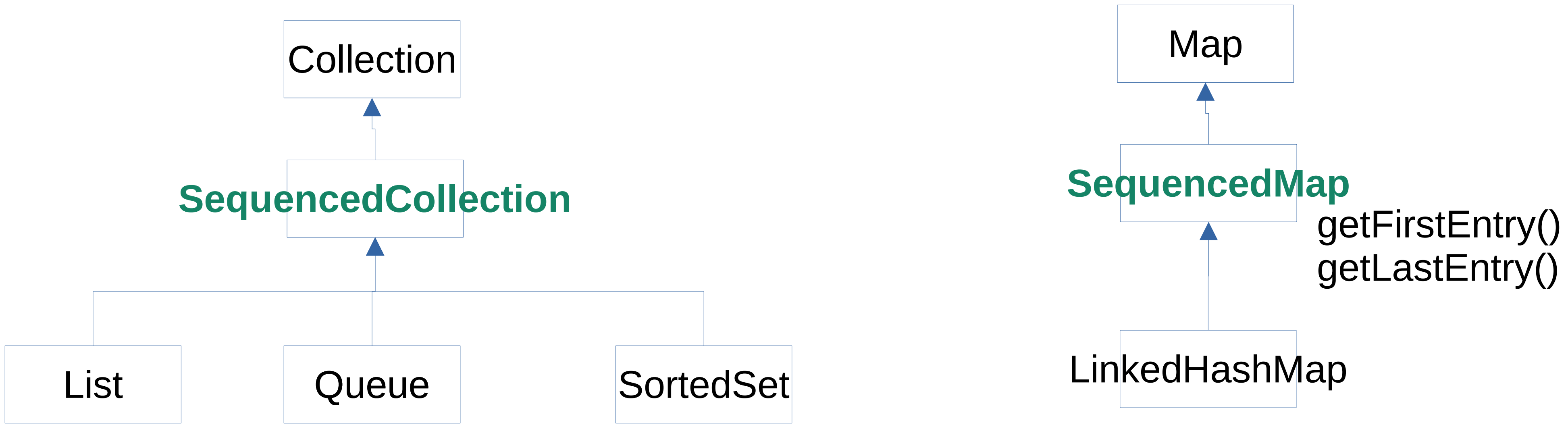
Hierarchy (v1)

SequencedCollection is a supertype of List, Queue and SortedSet



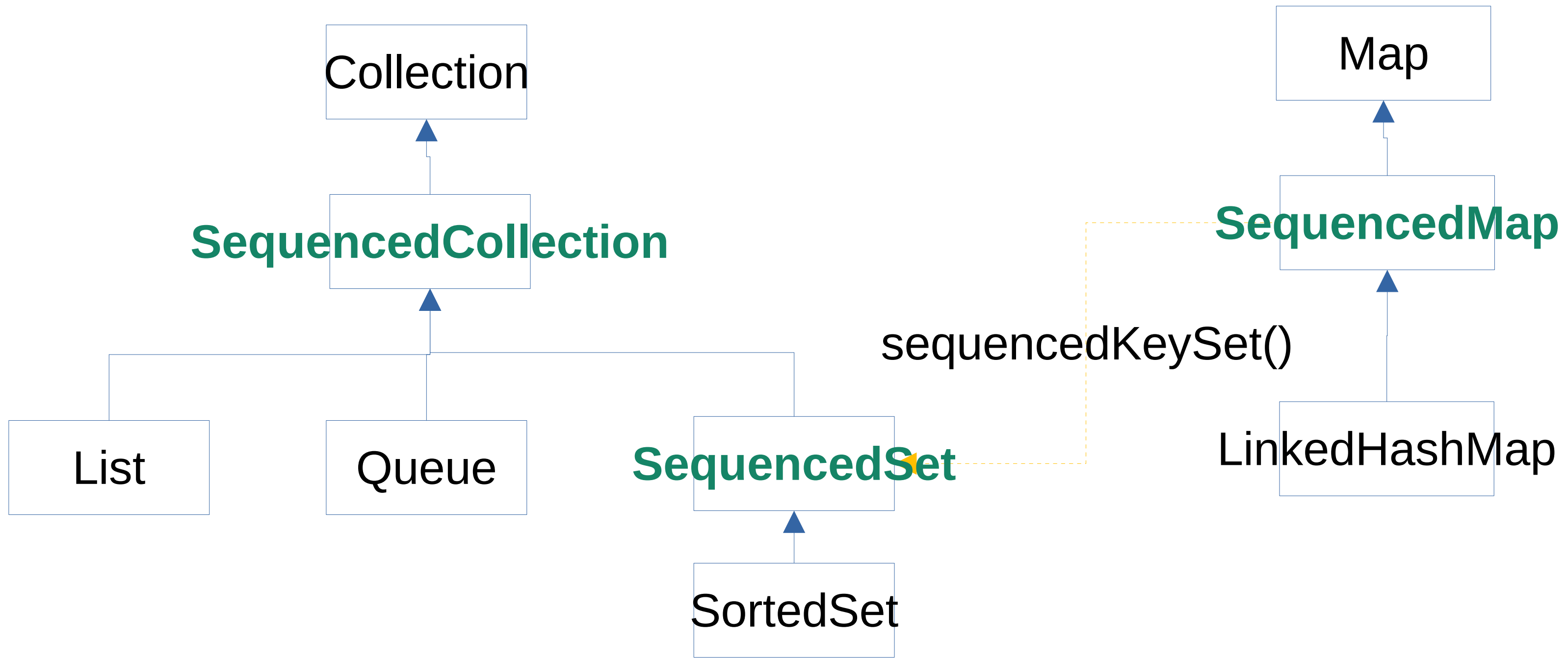
Hierarchy (v1) + Map

We want LinkedHashMap to be SequencedMap ?



Hierarchy (v2)

We need a SequencedSet to type SequencedMap.keySet()



Beware ! Here lies a dragon

Complexity

- List.addFirst()/removeFirst() are in O(n)

LinkedHashMap is weird

Ordered by access

```
var map = new LinkedHashMap<>(0, 0.75f, true);
map.put("foo", 3);
map.put("bar", 42);
System.out.println(map.get("bar"));  // 42
System.out.println(map.getFirstEntry());  // bar = 42
```

Virtual Threads

History

In C during the 80s

- OS processus + OS lock

In C during the 90s

- OS thread + mutex (application lock)

In Java during the 90s

- OS thread + synchronized (application lock)

Project Loom

Write synchronous code, execute asynchronously

Java 21: Application thread + application lock

Java can schedule millions of application threads (virtual threads) on top o few OS threads (# of cores)

How it works ?

When a method that should block is called

- The virtual thread is unmounted
  - The stack is copied to the heap
- A handler is registered on the blocking event

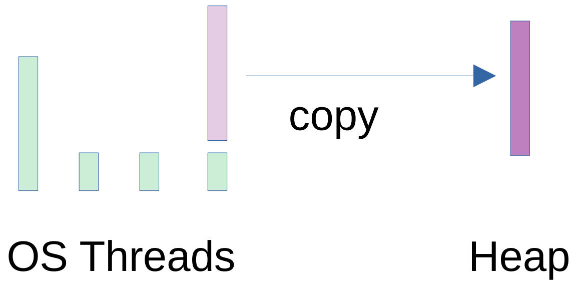
When the event handler is called

- The virtual thread can be scheduled (a fork/join pool in FIFO mode)
  - When scheduled, the virtual thread is mounted
    - The stack is copied back (incrementally)

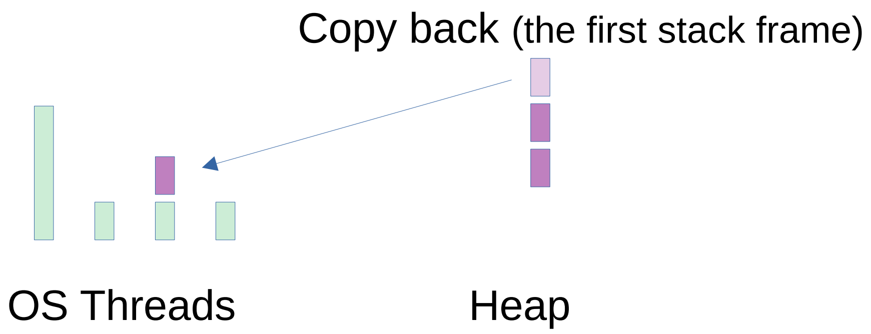
Mount and unmount

The VM copies parts of the stack back and forth

unmount



mount



jwebserver example

```
var path = Path.of("src/main/java").toAbsolutePath();
var handler = SimpleFileServer.createFileHandler(path);
var logger = SimpleFileServer.createOutputFilter(System.out, OutputLevel.INFO);
var server = HttpServer.create(new InetSocketAddress(8080), 10, "/", handler,
logger);

//var executor = Executors.newFixedThreadPool(5);
var executor = Executors.newVirtualThreadPerTaskExecutor();
server.setExecutor(executor);
server.start();
```

This executor does not pool the virtual threads

Command line
jwebserver [-b bind address] [-p port] [-d directory]

Can I use it now ?

Already supported by

- Spring 6, Micronaut 4,
- Quarkus (@RunOnVirtualThread),
- Helidon 4 (Mina) Q4 2023

Servers

- Tomcat, Jetty
- Netty has no support !?!

DB Drivers

- Postgres, H2, Oracle 21c

Structured Concurrency

Structured Concurrency (preview)

A better API than Executor / Future

```
try (var sts = new StructuredTaskScope<Integer>()) {
  var task1 = sts.fork(() - > ...);
  var task2 = sts.fork(() - > ...);

  sts.join();

  var result1 = task1.get();
  var result2 = task2.get();
}
```

No runaway threads anymore ?

Shutdown On Failure

Model serial groups of concurrent tasks

```
try (var sof = new StructuredTaskScope.ShutdownOnFailure<Integer>()) {
  var task1 = sof.fork(() - > ...);
  var task2 = sof.fork(() - > ...);          group 1
  sts.join();

  var task3 = sof.fork(() - > ...);          group 2
  sof.join();

  sof.throwIfFailed();
  .. task1.get() .. task2.get() .. task3.get()
}
```

Streamable (Java 22 ?)

Specify the business code on a stream of tasks

```
try (var sts = new StructuredTaskScope.Streamable<Integer>()) {
  sts.fork(() - > ...);
  sts.fork(() - > ...);

  List<Task<Integer>> list = sts.joinWhile(Stream::toList);

  System.out.println(list)
}
```

Streamable + limit + groupBy

If the stream is short-circuited, the remaining tasks are cancelled

```
try (var sts = new StructuredTaskScope.Streamable<Integer>()) {
  sts.fork(() - > ...);
  ...
  Map<State, Task<Integer>> map =
    sts.joinWhile(s - > s.limit(3).collect(groupingBy(Subtask::getState)));
  System.out.println(map.get(State.SUCCESS));
}
```

Platform Integrity

OpenJDK Integrity

Java 9 : Module enforces integrity

- No access to OpenJDK internals

    and setAccessible(true) is disable on OpenJDK code

Make Loom development faster

- reflection implementation changed
- Socket and Channel implementations changed

  etc ...

Integrity by default (1/2)

Extends the notion of integrity

https://openjdk.org/jeps/8305968

Dynamic agents, JNI and Foreign Function & Memory

Prepare to Disallow dynamic Loading of Agents (JEP 451)

- No problem if -javaagent or Launcher-Agent-Class
- Warning in Java 21
  - Error in the future, use -XX:+EnableDynamicAgentLoading

Integrity by default (2/2)

Foreign Function & Memory API (JEP 454)

- When calling a C function or allow unbounded access to native memory

  --enable-native-access=module or Enable-Native-Access

Prepare to restrict use of JNI

https://openjdk.org/jeps/8307341

also use --enable-native-access

I would like to hear your opinion ?

Executive Summary

Java 21

Java 21 focused on application developers

- Pattern Matching : instanceof + switch + record pattern
- Virtual Threads
- Sequenced Collection

with an eye to the future

- Unnamed class (preview)
- Template Processor (preview)
- Structured Concurrency (preview)