

Java 22+, Toward Java 25

Rémi Forax
Université Gustave Eiffel – June 2024



CALVIN & HOBBS © BIL WATTERSON

Don't believe what I'm saying !

Me, myself and I

Rémi Forax

- Assistant Prof at University Gustave Eiffel
- Expert for lambda, module, record, etc

Feel free to google me if you want to know more ...

Java 22/23 → Java 25

Language

- Implicit classes et `java.lang.IO` (preview)
- String templates ??? (in limbo)
- Unnamed variable ('_') and the *any* pattern (released 22)
- Flexible constructors (preview)

APIs

- Scoped Values (preview)
- Structured Concurrency (preview)
- Foreign Functions / Memory API (released 22)
 - Deprecate for removal `sun.misc.Unsafe` (java 23, warning in 24, off by default in 26+)
- Stream Gatherer (preview)

DEMO !

<https://github.com/forax/java-25-demo>

Great for teaching !

Implicit class

Great for scripting !

Class can be implicit

- No explicit constructor, no package

```
static void main(String[] args) { ... }
```

main / launcher protocol simplification

- The parameter is optional, static is optional
 - If not static, the default constructor is called
- ```
void main() { ... }
```

# Import module

*Great for scripting !*

Can import all exported packages of a module

- Equivalent to import \*

```
import module java.base;
```

```
<=> import java.lang.*, java.util.*, java.io.*, etc
```

Implicit classes import the module `java.base` by default

Great for teaching !

# java.io.IO

New class with 3 methods

- void **print/println**(Object)
- void **readln**(Object prompt)

Implicit classes import all static methods of  
java.io.IO

```
import static java.io.IO.*;
```



# String template

## String interpolation

- **var** name = ...  
STR."foo {name}"

## Generalized

- Can return any types
- SQL."select foo where id = {name}"

# String template (remaster)

## The proposed API

- String template arguments are typed `java.lang.Object`
- String template processor are not efficient
- Using a method is more powerful
  - `sql("select foo where id = \{name}\")`  
=> `sql("select foo where id = \{}", name)`

## Need to go back to the backboard

- No preview for Java 23

# Unnamed variable

Can be used anywhere a variable is not part of the API

```
for(var _ = ...)
```

```
try(var _ = ...)
```

```
catch(Throwable _)
```

```
(int x, int _) -> ...
```

```
class A {
```

```
 void m(String _) { } // error, public API !
}
```

# Any pattern

A pattern that recognizes any value

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

```
switch(point) {
 case Point(int _, int _) - > ... // unnamed variable
 case Point(_, _) - > ... // any pattern
}
```

# Flexible constructor

Align the Java semantics to the JVM semantics

- JVM: do not use “this” as a value before the call to the super constructor
- Java : do not use any statements before the call to the super constructor
  - But inner class outer instance is an exception

# Statements before super()

Preconditions can be validated **before** the call to super()

```
class Dog {
 final String name;

 Dog(String name) {
 Objects.requireNonNull(name); // ok !
 super();
 this.name = name;
 }
}
```

# Field inits before super()

Fields can be initialized before the call to super()

```
class Dog {
 final String name;

 Dog(String name) {
 Objects.requireNonNull(name);
 this.name = name; // ok !
 super();
 }
}
```

This is required for fields of value classes (Valhalla)

# Foreign Memory API

Safe/Fast Access to off-heap memory

- Off-heap memory == not managed by the GC
- Like ByteBuffer
  - But explicit deallocation, and index > 2G (long)

MemorySegment == a region of memory

Arena

- allocate() one or more MemorySegment
- close() deallocate all segments



# Example

Uses `Arena.global()` == no deallocation

- `Arena arena = Arena.global();`

Allocate 16 bytes of memory

- `MemorySegment segment = arena.allocate(16);`

Set a byte

- `segment.set(ValueLayout.JAVA_BYTE, 7, (byte) 12);`

Get a byte

- `segment.get(ValueLayout.JAVA_BYTE, 7)`

# Arenas

Apart `global()`, there are ~~three~~ two other arenas

- `Arena.ofConfined()`
  - Only the current thread can `allocate()`
- `Arena.ofShared()`
  - All threads can `allocate()`
  - `Close()` quite slow => thread handshakes
- `Arena.ofAuto()`
  - Deallocation by the GC, do not use explicitly !
    - `ByteBuffer` has been retrofitted to use `MemorySegment`

# Struct (1/2)

A MemoryLayout defines an offset for each fields

```
static final MemoryLayout POINT = MemoryLayout.structLayout(
 JAVA_INT.withName("x"),
 JAVA_INT.withName("y")
);
```

A VarHandle can access to a specific field

```
static final VarHandle X = POINT.varHandle(groupElement("x"));
```

-  
Warning: you need both **static** and **final** to get performance

# Struct (2/2)

A MemoryLayout defines an offset for each fields

- Allocate using a MemoryLayout

```
var segment = arena.allocate(POINT);
```

- Set a field value using the VarHandle

```
X.set(segment, 0L, 12);
```

- Get a field value using the VarHandle

```
(int) X.get(segment, 0L)
```

the cast is necessary otherwise the value is boxed !

Offset if the struct is inside a struct



# Array of Struct

Accessing a memory segment as an array

- Need an array element VarHandle

```
static final VarHandle ARRAY_X =
 POINT.arrayElementVarHandle(groupElement("x"));
```

- Allocate Using a MemorySegment and a size

```
var segment = arena.allocate(POINT, 16);
```

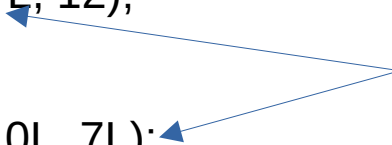
- Set a value at an index

```
ARRAY_X.set(segment, 0L, 7L, 12);
```

- Get a value at an index

```
(int) ARRAY_X.get(segment, 0L, 7L);
```

index of the array



# Memory Mapped File

The off-heap memory can be mapped to a file

- Create a FileChannel
  - **try**(**var** channel = FileChannel.open(path, READ, WRITE, CREATE)) {
- Declare an arena
  - **try**(**var** arena = Arena.ofConfined()) {
- Create the segment by mapping the file
  - **var** segment = channel.map(READ\_WRITE, 0L, 16L \* POINT.byteSize(), arena);

# MemorySegment based Stream

See a segment + MemoryLayout as a Stream

```
try(var arena = Arena.ofConfined()) {
 var segment = channel.map(READ_WRITE,
 0L, 16L * POINT.byteSize(), arena);
```

Using MemorySegment.elements(MemoryLayout)

```
var sum = segment.elements(POINT)
 .mapToInt(s -> (int) X.get(s, 0L))
 .sum();
```

The stream slices the big segments in small segments

# And with a parallel stream

Use `Arena.ofShared()` instead of `ofConfined()`

```
try(var arena = Arena.ofShared()) {
 var segment = channel.map(READ_WRITE,
 0L, 16L * POINT.byteSize(), arena);
```

Using `MemorySegment.elements(MemoryLayout)`

```
var sum = segment.elements(POINT)
 .parallel()
 .mapToInt(s -> (int) X.get(s, 0L))
 .sum();
```



# Executive Summary

# Executive Summary

- Scripting / Teaching is easier
- Valhalla integration with flexible constructors
- Stream gatherer (not specialized, no characteristics)
- Foreign API
  - Nice API to call C Methods (but unsafe)
  - Can access off heap memory in a safe/structured way
    - `sun.misc.Unsafe` requires a command line flag