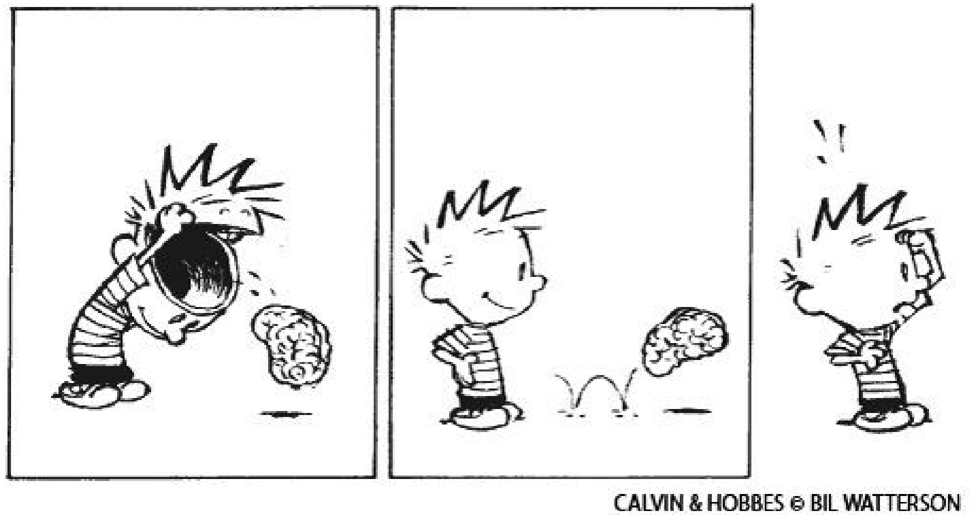
Java 25

Rémi Forax
Université Gustave Eiffel – September 2025



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 ...

What is Java ?

Java the language ?

Write Once Run Anywhere

Java the language ?

Write Once Run Anywhere ... forever

Java the language ?

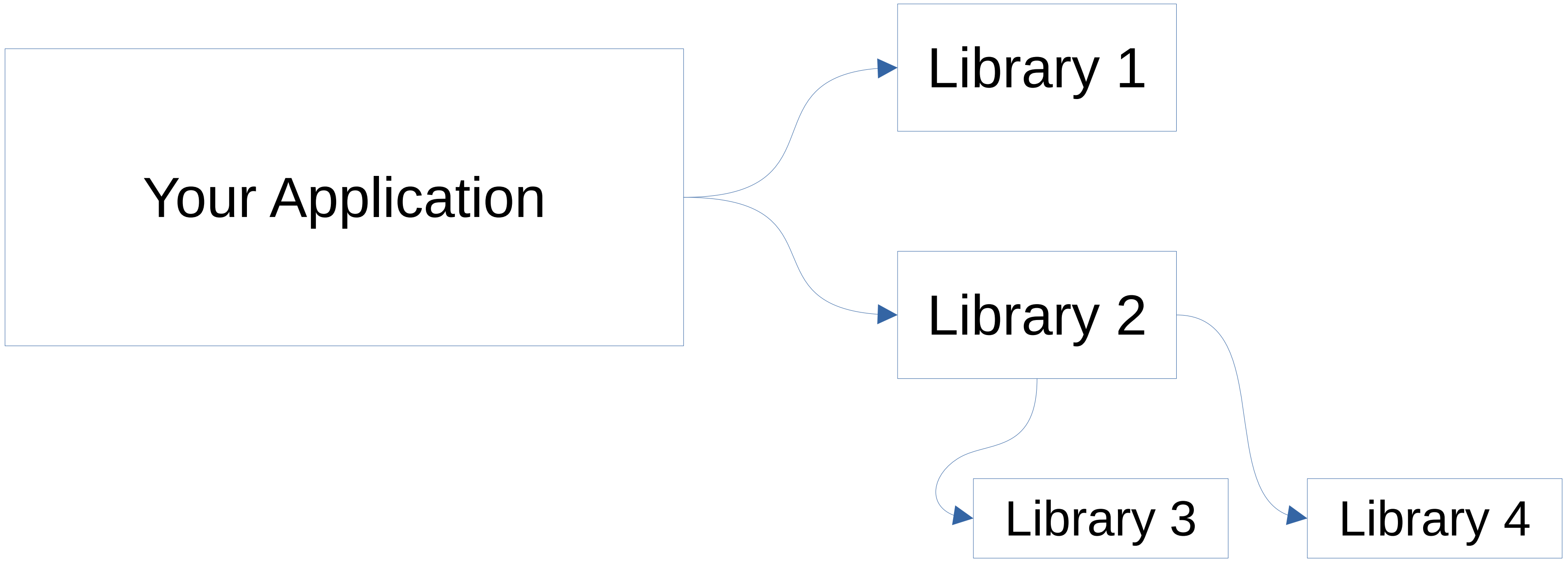
Write Once Run Anywhere ... forever

<==>

Composition at Scale

Composition at Scale

Fearless composition of libraries



API backward compatibility

Java provides binary backward compatibility

- Unlike C, C++, Rust, Python, or JS

Composition of libraries written with different versions of the language

- No: Python 2 vs Python 3

Java versions

Softwares change, Java too

- Java 1: OOP (1995)
- Java 5: Generics (2004)
- Java 8: Lambda (2014)
- Java 21: Record & Pattern Matching (2023)
- Java 2?: Value Type (202?)

Drift toward functional ...

Immutable atoms (String, Integer ...)

java.util.Date	→	java.time.*
Collections	→	List.of(), Set.of(), Map.of()
class	→	record
	→	value class
loops	→	streams
Inheritance Polymorphism	→	Sealed Interface Pattern Matching
MUTABLE WORLD		IMMUTABLE WORLD

Don’t of Java

Clash with the composition at scale

Cases of function coloring*

- Rust lifetime attributes (we have GCs)
- Async/Reactive methods (we have virtual threads)
- Use class/type for errors (we have exceptions)

  I know checked exceptions are bad :(

* https://journal.stuffwithstuff.com/2015/02/01/what-color-is-your-function/

What’s new in Java ?

Java 21 → Java 25

Language

- Compact classes, main() and java.lang.IO
- Import module
- Unnamed variable (‘_’) and the *any* pattern (released in 22)
- Flexible constructors

DEMO !

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

Java 21 → Java 25

APIs

- Scoped Values
- Structured Concurrency (new API, preview)
- Foreign Functions / Memory API (released in 22)
- Stream Gatherer (released in 24)
- Stable Value (preview)

Better Virtual Thread

Synchronized + IO calls

Until Java 24

-
  ```
  synchronized(foo) {
    channel.read(…);  // block the carrier thread
  }                    // aka virtual thread pinning
  ```

- Synchronized fast path (not contented) stores the lock inside the stack
  - The header of the object point to the lock on stack
  - To it can not be copied to heap :(

Synchronized + IO calls

In Java 24

- Synchronized is re-implemented
  - The fast lock is stored in a lock arena at the bottom of the stack
- Part of project Liliput
  - Move header from 96 bits to 64 bits
  - Enable with -XX:+UseCompactObjectHeaders
    - Should be default in Java 26

ScopeValue

ThreadLocal replacement

- ThreadLocal issues
  - Does not scale with many virtual threads
  - Not lightweight
  - Hard to optimize for the JIT (mutation)
- Need both a better API and a better implementation
  - ScopeValue

DEMO !

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

ScopeValue

- ScopeValue
  - Creation with ScopeValue.newInstance()
  - ScopeValue.where(key, value)
  - ScopeValue.where(…)
        .run(runnable)   // the value is available for the Runnable

- Values are cached on stack (vs heap)
- Can be optimized by the JIT (with inlining)

Gatherer

Stream operations

Relation

- 1 to 0 or 1: filter
- 1 to 1: map
- 1 to many: flatMap, mapMulti
- many to many ?

DEMO !

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

StableValue

Double Check Locking

The version that fails !

```
static BigObject BIG_OBJECT;

static BigObject getBigObject() {
  if (BIG_OBJECT != null) {
    return BIG_OBJECT;
  }
  synchronized(CurrentClass) {
    if (BIG_OBJECT != null) {
      return BIG_OBJECT;
    }
    return BIG_OBJECT = new BigObject();
  }
}
```

Double Check Locking (pseudo code)

The version that fails !

```
static BigObject BIG_OBJECT;

– static BigObject getBigObject() {
    if (BIG_OBJECT != null) {
      return BIG_OBJECT;
    }
    synchronized(CurrentClass) {
      if (BIG_OBJECT != null) {
        return BIG_OBJECT;
      }
      var tmp = new BigObject
      tmp.BigObject();                    // those two lines can be re-ordered
      BIG_OBJECT = tmp;
      return tmp;;
    }
  }
```

Double Check Locking

The Okay version

```
static volatile BigObject BIG_OBJECT;

static BigObject getBigObject() {
  if (BIG_OBJECT != null) {
    return BIG_OBJECT;
  }
  synchronized(CurrentClass) {
    if (BIG_OBJECT != null) {
      return BIG_OBJECT;
    }
    return BIG_OBJECT = new BigObject();
  }
}
```

DEMO !

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

Structured Concurrency

j.u.c.Executor flaws

- Does not cancel other tasks when one fails

- Java 19: Executor implements AutoCloseable

- Need a new API
  - The first API proposed has been modified in 25

DEMO !

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

Predefined Joiners

Predefined joiners

- awaitAllSuccessfulOrThrow
  - Stores firstException
- allSuccessfulOrThrow()
  - Stores firstException + List<SubTask>
- anySuccessfulOrThrow()
  - Stores Subtask
- allUntil(Predicate)
  - Stores Predicate

Summary

Executive Summary

- Java 25 is the LTS
- A lot of preview features are now released
  - Unnamed variable, unnamed pattern
    - Improve expressiveness
  - Compact class, better main, import module
    - Cool for demos and beginners
  - Flexible constructor (value class will land soon)
- Better runtime (Liliput, ZGC, Shenandoah, G1, …)
- Better APIs (ScopeValue, FFM, Gatherer)