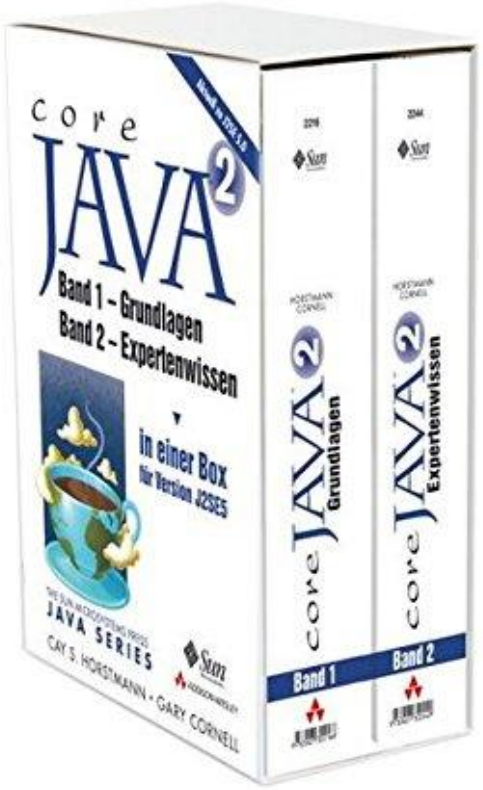
Java Evolution



- Cay Horstmann
- Author of Core Java (11 editions since 1996), Java for the Impatient, etc.

Why Care about Evolution?

Joker<?>

- “There are two kinds of programming languages: the ones people complain about, and the ones that nobody has heard of.”—Bjarne Stroustrup
- Java was designed as a “blue-collar” language
  - Easy to learn
  - Unsurprising
- Java has a vast standard library
  - Mostly of high quality
- Java is incredibly backwards-compatible
- For many years, evolution was managed by a deliberative and formal process
  - Java Community Process
  - Java Specification Requests
- The process has changed
  - How well is it working?
  - What can/should developers do?

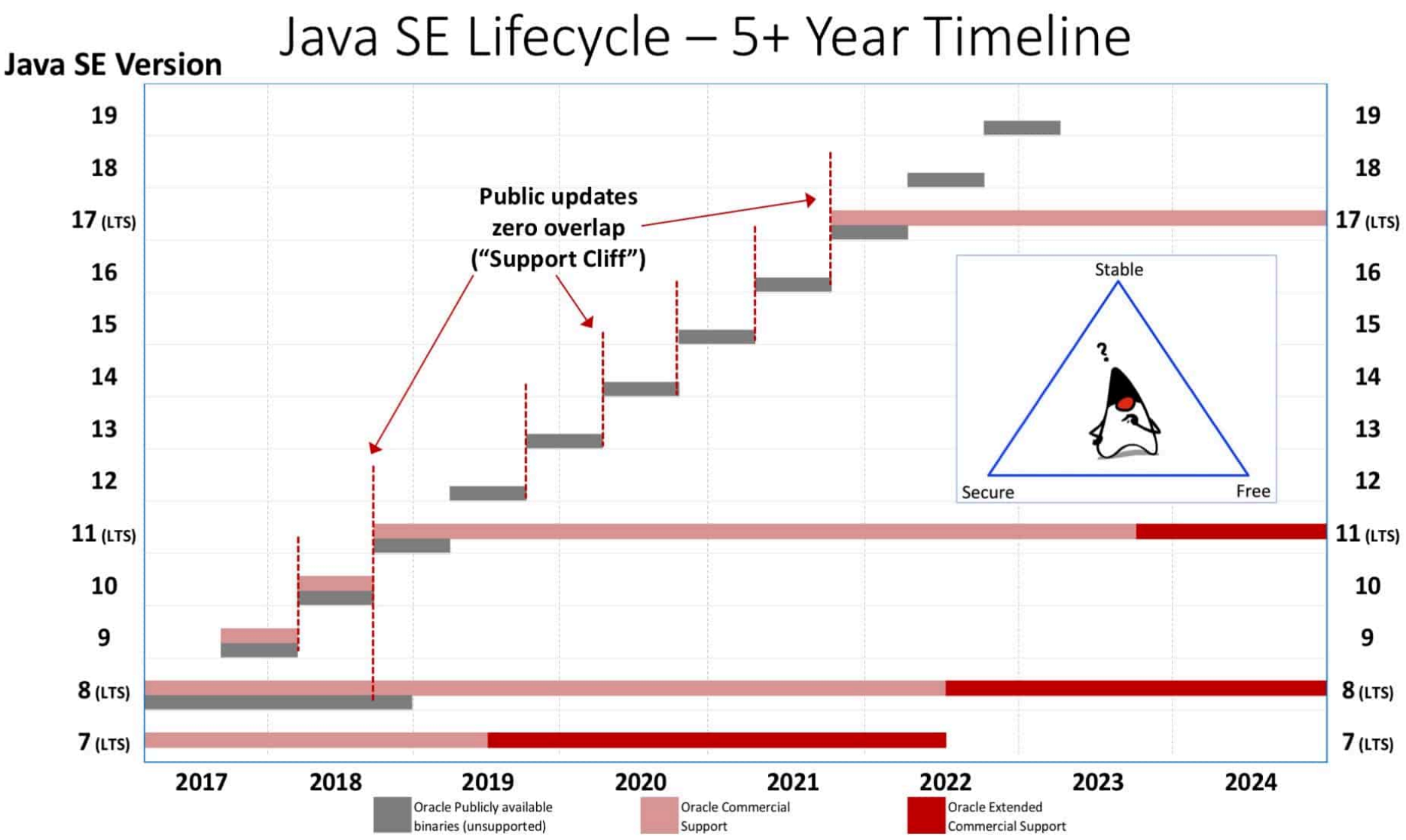
A Brief History of Java

Joker<?>

Version	Year	New Language Features	Number of Classes and Interfaces
1.0	1996	The language itself	200
1.1	1997	Inner classes	418
1.2	1998	The `strictfp` modifier	1,588
1.3	2000	Nothing at all	1,883
1.4	2002	Assertions	2,696
5.0	2004	Generic classes, “for each” loop, varargs, autoboxing, metadata, enumerations, static import	3,551
6	2006	Nothing at all	4,069
7	2011	Switch with strings, diamond operator, binary literals, exception handling enhancements	4,509
8	2014	Lambda expressions, interfaces with default methods	4,733
9	2017	Modules	6,603
10	2018	`var`	6,599 (!)
11	2018	Nothing at all	4,910 (!!)—Java FX, JNLP, Java EE overlap, CORBA removed
12	2019	`switch` expression preview	4,935
13	2019	Text blocks preview	4,904

The Release Cadence

Joker<?>



Projects, JEPs, Incubation, Preview

Joker<?>

- Projects
  - Amber: Small productivity-oriented language features
  - Valhalla: Value types and generic specialization
  - Loom: Fibers and continuations
  - Many others: Graal, Panama, Kulla, Sumatra, ...
- JEP = Java Enhancement Proposal
  - De facto managed by Oracle
  - Participants must sign Oracle Contributor Agreement
  - Tracked on JDK bug system
  - Discussed on openjdk mailing lists
- Incubator module (JEP 11): Non-final API
- Preview feature (JEP 12): Non-final language/VM feature

File API Evolution

Joker<?>

- Bug report JDK-8183743—Umbrella: add overloads that take a Charset parameter
- Java 7 introduces `StandardCharsets` subclass of `Charset`
- `StandardCharsets.UTF_8` can be autocompleted, don't have to ponder `"UTF-8"` vs `"UTF8"` vs `"UTF_8"`
- Java 7 makes use of it: `Files.newBufferedReader(Path, Charset)`
- Bug JDK-8183743 says “what about the rest of the API?”
- Java 10 adds constructors `Scanner(..., Charset charset)` and `PrintWriter(..., Charset charset)`
- Java 11 does the same for `FileReader`, `FileWriter`
- Hooray!
- For similar enhancements, search for CSR in https://bugs.openjdk.java.net

HttpClient

Joker<?>

- 2012 “Http Client API for JDK 8” announced on `openjdk-net-dev` mailing list
  - Replacing crufty `HttpURLConnection`
- 2014 JEP 110 “HTTP/2 Client (Incubator)” expands scope to HTTP/2 and WebSocket
- 2015 API review on mailing list, first prototype (HTTP/1.1 only) available
- 2016 Minor API revisions
  - Build a client:

```
HttpClient client = HttpClient.newBuilder()
    .followRedirects(HttpClient.Redirect.ALWAYS)
    .build();
```

  - Build a request:

```
HttpRequest request = HttpRequest.newBuilder()
    .uri(new URI("http://horstmann.com"))
    .GET()
    .build();
```

  - Get and handle response:

```
HttpResponse<String> response = client.send(request, HttpResponse.BodyHandler.asString());
```

HttpClient

Joker<?>

- Asynchronous processing:

```
client.sendAsync(request, HttpResponse.BodyHandler.asString())
   .completeOnTimeout(errorResponse, 10, TimeUnit.SECONDS)
   .thenAccept(response -> ...);
```

- 2017 HttpClient released with JDK 9 in module `jdk.incubator.httpclient`
- 2017 JEP 321: HTTP Client (Standard)
- 2018 HttpClient moved to module `java.net.http`, with very minor API changes
  - Ex. `HttpRequest.BodyProcessor.fromString(data)` → `HttpRequest.BodyPublishers.ofString(data)`

HttpClient

Joker<?>

- So, it should be perfect now, right?
- No body publishers for form data, file upload
  - It's 2019, and we still need to mess with `URLEncloder`, MIME multipart
- No JSON body publisher, processor
  - JSON not part of the core platform
- No automatic compression handling
- No way of controlling high-volume concurrent requests (Bug JDK-8183743)
- Which executor runs the `thenAccept` lambda?


```
ExecutorService pool = Executors.newFixedThreadPool(NTHREADS);
HttpClient.newBuilder()
    .followRedirects(HttpClient.Redirect.NORMAL)
    .executor(pool)
    .build()
    .sendAsync(request, HttpResponse.BodyHandlers.ofString())
    .thenAccept(response -> . . .);
```

  - The docs say `pool`
  - In fact, it's the standard fork-join pool (Bug JDK-8204339)
- Takeaway: Don't expect miracles from the incubation process

Raw String Literals

Joker<?>

- Jan. 2018 JEP 326
- Goals:
  - Foreign strings without escapes
  - Multiline strings
- Non-goals:
  - String interpolation
  - Postprocessing

- Simple and ingenious: One more backtick than the contents

````
String markdown = ````Writing about JavaScript
```
alert("JavaScript");
```
in Markdown````
````

- Teensy problem: The string cannot *start* with a backtick.
  - Why not start literal with ```` followed by newline?

````
String markdown = ````
```
alert("JavaScript");
```
````
````
````

Off the Deep End

Joker<?>

- Not blending in:

```
   ...
   String myNameInABox = `
+-----+
| Cay |
+-----+`;
```

- Remedy: Stripping common whitespace prefix:

```
String myNameInABox = `
                       +-----+
                       | Cay |
                       +-----+
                      `;
```

- What about tabs?
  - Started out with: Every prefix has to be the same sequence of tabs/spaces
  - Later changed to: Every Unicode white space character counts as with 1
- Discussion went back and forth over hundreds of messages, including this beauty: “You are correct. I thought I caught all the stray cases, but... The examples should have indentations that are multiples of 4. The issue with pasting from IDEs to mailers. ”

Text Blocks

Joker<?>

- JEP 326 withdrawn before JDK 12 shipped
  - `` could be confused with empty string
  - Can't have raw string start with `
  - Waste of backtick character, ` hard to see, hard to type
  - “Any number of quotes” confusing for IDEs
- Next up: JEP 355 Text Blocks
  - Multiline strings
  - Not raw, but medium rare—not too many escape sequences

```
String myFaceInASCII = """
                       ""\"""\"""
                       | o  o |
                       |  ==  |
                       \\------/
                       """;
```

- Newline after initial """ required
- Must escape backslash, at least one " in """
- Prefix and trailing spaces stripped
  - Want more control over stripping? CSR JDK-8227870 proposes *newline* and \s
- Ships with JDK 13

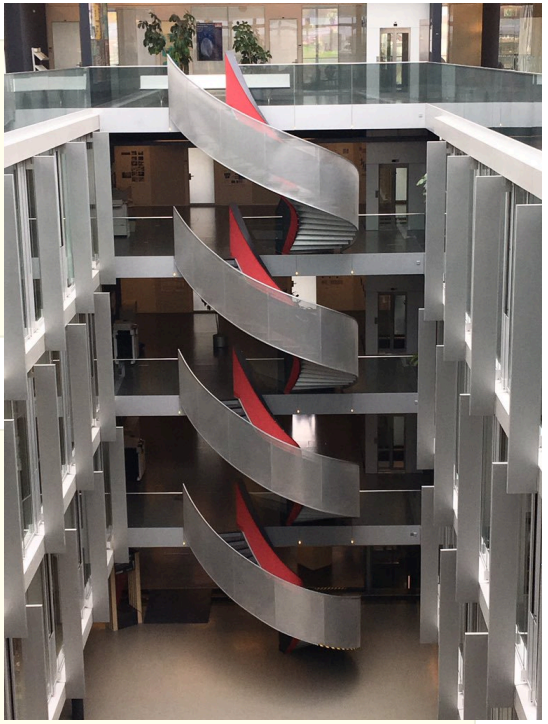
Multiline	Raw	Interpolation	Postprocessing
Yes	No	No	No

Scala Pattern Matching

Joker<?>

- A better `switch`

```
str(i) match {
  case '+' | '-' => sign = 44 - ch
  case ch if Character.isDigit(ch) => digit = Character.digit(ch, 10)
  case SPACE => -1 // Uppercase for constants
  case _ => sign = 0
}
```



- Type patterns

```
obj match {
  case x: Int => x
  case s: String => Integer.parseInt(s)
  case _: BigInt => Int.MaxValue // Caution: not case BigInt
  case _ => 0
}
```

- Extraction

```
arr match {
  case Array() => 0
  case Array(x, 0) => x
  case Array(x, rest @ _*) => rest.min
}
```

- Also works with variable declarations, for comprehensions

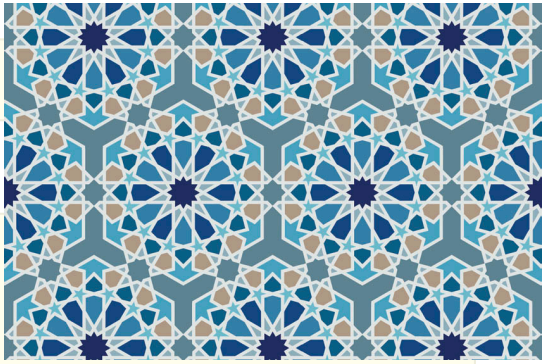
```
val Array(first, second, rest @ _*) = arr
for ((k, "") <- System.getProperties())
```

Java Pattern Matching

Joker<?>

- JEP 305:

```
ch = obj instanceof String s && !s.isEmpty() ? s.charAt(0) : ' '
```



- Musings about

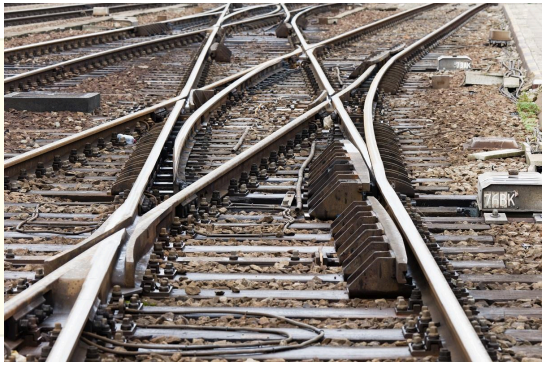
```
case String s
case Point(var x, var y)
case Point(var x, 0)
case Point(var x, _)
case String s && !s.isEmpty()
```

Switch expressions

Joker<?>

- JEP 325:

```
int numLetters = switch (day) {
    case MONDAY, FRIDAY, SUNDAY -> 6;
    case TUESDAY                -> 7;
    case THURSDAY, SATURDAY     -> 8;
    case WEDNESDAY              -> 9;
};
```



- An expression, not a statement
  - Like conditional expression `? :`
- Expressions to the right of `->`
- No “fall through”
- Must be exhaustive
  - Can have `default ->`
- “These changes will simplify everyday coding, *and prepare the way for the use of pattern matching*”
- Delivered as preview feature in JDK 12
- And with a minor change as preview feature in JDK 13 (JEP 354)
- Now targeted as a standard feature of JDK 14 (JEP 361)

Give Me a Break

Joker<?>

- What if you need to log something to the right of ->?

```
case TUESDAY -> { logger.info("Tuesday"); 7 }
```

- That's not Java—no block expression
- Lambdas suffer from the same problem

```
x -> x * x
x -> { logger.info("Dare to be square"); return x * x }
```

- What to do about a problem like block expressions?
  - `case TUESDAY -> { logger.info("Tuesday"); return 7; }`
  - `case TUESDAY -> { logger.info("Tuesday"); break 7; } // JDK 12`
- What about

```
case ... -> {
   for (int i = 0; i < a.length; i++) {
      if (a[i] == x) break i; // Error
   }
   break -1;
}
```

- What about

```
case ... -> { ...; break out; } // A labeled break?
```

- In JDK 13, it's `yield`

Frankenswitch

Joker<?>

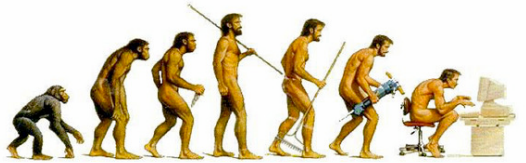
- What about poor old toxic statement switch?

```
switch (day) {
    case MONDAY, FRIDAY, SUNDAY -> // Multiple case labels
        numLetters = 6; // No fallthrough
    case TUESDAY -> {
        logger.info("Tuesday");
        numLetters = 7;
    }
    case THURSDAY, SATURDAY ->
        numLetters = 8;
    default ->
        numLetters = 9;
}
```

Evolution is Not a One-Way Street

Joker<?>

- What if expression switch has fall-through envy?



```
int numLetters = switch(day) {
   case MONDAY, FRIDAY, SUNDAY:
      yield 6;
   case TUESDAY:
      logger.info("Tuesday");
      yield 7;
   case THURSDAY:
      logger.info("Thursday"); // Yay! Fallthrough
   case SATURDAY:
      yield 8;
   default:
      yield 9;
};
```

The Matrix (JDK 12)

Joker<?>

	Expression	Statement

No fallthrough	`int numLetters = switch (day) {`
	`  case MONDAY, FRIDAY, SUNDAY -> 6;`
	`  case TUESDAY -> 7;`
	`  case THURSDAY, SATURDAY -> 8;`
	`  default -> 9;`
	`};`
		`switch (day) {`
		`  case MONDAY, FRIDAY, SUNDAY ->`
		`  numLetters = 6;`
		`  case TUESDAY -> {`
		`  logger.info("Tuesday");`
		`  numLetters = 7;`
		`  }`
		`  case THURSDAY, SATURDAY ->`
		`  numLetters = 8;`
		`  default ->`
		`  numLetters = 9;`
		`}`

Fallthrough	`int numLetters = switch(day) {`
	`  case MONDAY, FRIDAY, SUNDAY:`
	`  break 6;`
	`  case TUESDAY:`
	`  logger.info("Tuesday");`
	`  break 7;`
	`  case THURSDAY:`
	`  logger.info("Thursday");`
	`  case SATURDAY:`
	`  break 8;`
	`  default:`
	`  break 9;`
	`};`
		`switch(day) {`
		`  case MONDAY, FRIDAY, SUNDAY:`
		`  numLetters = 6;`
		`  break;`
		`  case TUESDAY:`
		`  logger.info("Tuesday");`
		`  numLetters = 7;`
		`  break;`
		`  case THURSDAY:`
		`  logger.info("Thursday");`
		`  case SATURDAY:`
		`  numLetters = 8;`
		`  break;`
		`  default:`
		`  numLetters = 9;`
		`}`

The Original Sin

Joker<?>

- “It seemed more desirable to tease the desired benefits (expression-ness, better control flow, saner scoping) into orthogonal features, so that **switch expressions** and **switch statements** could have more in common. The greater the divergence between **switch expressions** and **switch statements**, the more complex the language is to learn, and the more sharp edges there are for developers to cut themselves on.”
- Another matrix

	Statement	Expression
Two-way branch	`if/else`	`? :`
Multi-way branch	`switch`	`switch`

- Why not multiway expressions?

```
int numLetters = day ??
   MONDAY, FRIDAY, SUNDAY -> 6 :
   TUESDAY -> 7 :
   THURSDAY, SATURDAY -> 8 :
    9;
```

- And what does any of this have to do with *pattern matching*?
  - `switch`: Jump table with binary search
  - Patterns are sequential, not constant

Records

Joker<?>

- Just immutable data

```
record Point(int x, int y) {
   double distance(Point other) { ... }
}
```

- Automatic constructor, private final fields, accessors `x()`, `y()`, `equals`, `hashCode`, `toString`
- No other instance fields
- Eventually, some kind of deconstructor for pattern matching
- `final` and not `abstract`, supertype `java.lang.Record`
- Serializable
- Tuples with names
- Actually some data
  - Judging from Google `@AutoValue` usage, likely to be about as popular as `enum`
- Non-goal: To solve all Java boilerplate problems
- To appear “soon”
- Some people want methods to update fields:.

```
Point projection = myPoint.y(0)
```

Loom

Joker<?>

- 1995: Java language has thread support
- 1997: Java Web Server runs each web request in a new thread
  - Amazing: thousands of concurrent requests
- Threads are expensive
  - But what can you do when one blocks?
- Asynchronous programming
  - Callback hell
  - Futures
  - Async/await
- Loom: What if blocking wasn't expensive?
  - Millions of concurrent *fibers*
  - Each thread runs many fibers
  - Creating, switching between fibers cheap
  - Blocking is virtually free
  - VM, API park, unpark blocking fibers

“Make Concurrency Easy Again”

Joker<?>

- Not so fast...
- More than one reason for concurrency
- User interfaces: UI components not threadsafe
  - Single UI thread serializes operations
  - Fibers won't help
  - Keep using `AsyncTask`/`SwingWorker`
- What about parallel streams—the previous promise to “make concurrency easy again”?
  - Works great for non-blocking workloads...
  - ...on splittable data structures
  - With fibers, ok to block in tasks...
  - ...on splittable data structures
- Fibers don't add value for computationally-intensive tasks
- Sweet spot: Many more tasks than threads, tasks not compute-bound

Kick the Tires

Joker<?>

- Easy to build:

```
git clone https://github.com/openjdk/loom
cd loom
git checkout fibers
sh configure
make images
```

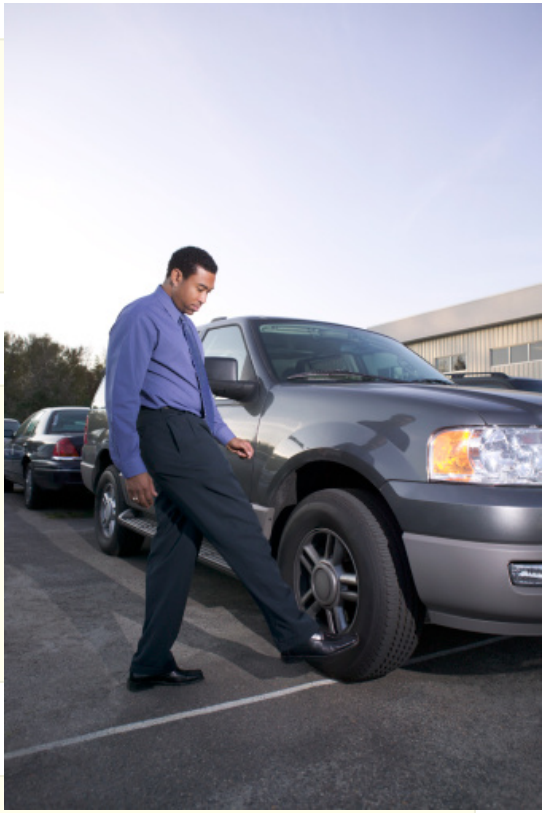
- Run a million fibers:

```
FiberScope scope = FiberScope.open();
for (int i = 1; i <= NTASKS; i++) {
   int n = i;
   scope.schedule(() -> run(n));
}
scope.close();
```

- Now `Thread.sleep` makes the current fiber sleep:

```
public static void run(int n) {
   try {
      Thread.sleep(1000);
   } catch (InterruptedException ex) {
      ex.printStackTrace();
   }
   System.out.println(n);
}
```

- Try it with threads
  - On my laptop, out of memory after about 10,000 threads

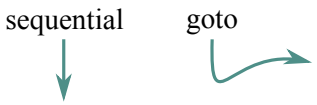


Structured Concurrency

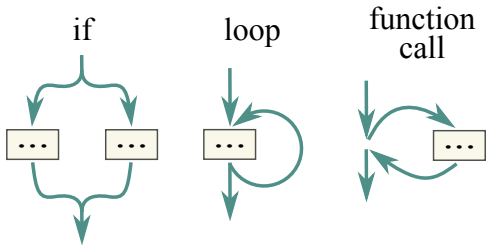
Joker<?>

- Nathaniel Smith: “Start and forget” is like `goto`

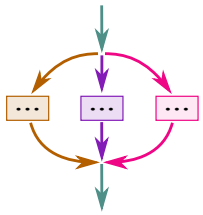
```
go myfunc();
new Thread(this::myfunc).start();
```



- 1960s: Structured programming replaces `goto` with branches, loops, functions



- Structured concurrency: Should do the same with concurrent tasks



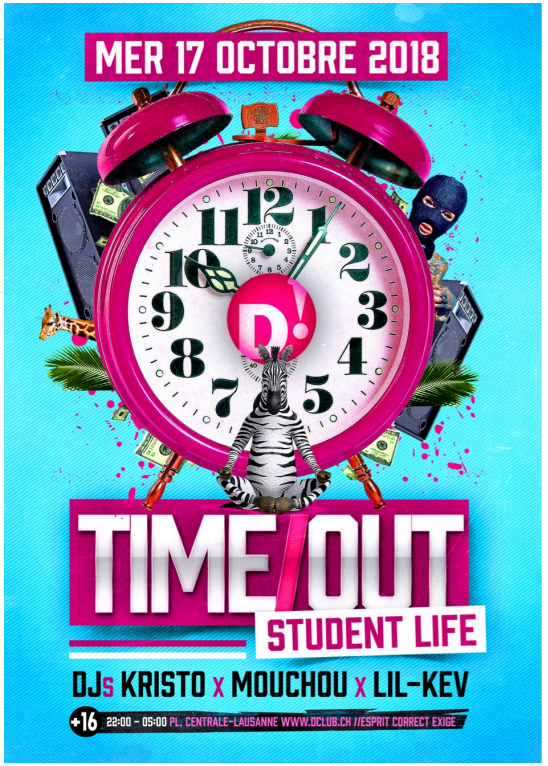
- Control over resource deallocation, cancelation

Timeout

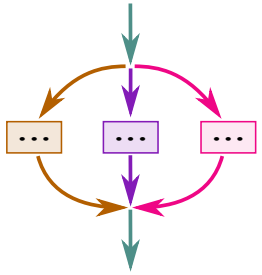
Joker<?>

- Concurrent tasks in a scope:

```
FiberScope scope = FiberScope.open();
for (int i = 0; i < NTASKS; i++) {
   scope.schedule(this::myfunc);
}
scope.close(); // Blocks until all fibers finish
```



- Can use `try`-with-resources
- What if some of them don't finish?



- Open scope with deadline (`Instant`) or timeout (`Duration`):

```
try (var scope = FiberScope.open(Instant.now().plusSeconds(30))) {
   for (...)
      scope.schedule(...);
} // All fibers that haven't finished canceled after deadline
```

- Deadlines compose better

Cancellation

Joker<?>

- Long-standing problem for Java concurrency
- Call `fiber.cancel()` to set cancel status
- Unparks parked fiber, throws `InterruptedException/IOException`
- Cooperative cancellation: Busy fiber code should periodically call `Fiber.cancelled()`
- When scope times out, its fibers get cancelled
- Unlike interrupted status, cancel status can never be reset
- Cancelation can be controlled by options in `FiberScope` constructor
  - `CANCEL_AT_CLOSE`: Closing scope cancels all scheduled fibers instead of blocking
  - `PROPAGATE_CANCEL`: If owning fiber is canceled, any newly scheduled fibers automatically canceled
  - `IGNORE_CANCEL`: Scheduled fibers can't be canceled
- Top-level default: all options unset
  - `PROPAGATE_CANCEL`, `IGNORE_CANCEL` inherited from parent scope

Bigger Example

Joker<?>

- At JCrete 2019, Heinz Kabutz gave a puzzler with a program that loaded thousands of Dilbert cartoon images, one per day
- For each image,
  - Load page such as `https://dilbert.com/strip/2011-06-05`
  - Find image URL in strip
  - Load image from that URL
  - Display or save image

- It was a mess of completable futures, somewhat like:

```
CompletableFuture
  .completedFuture(getUrlForDate(date))
  .thenComposeAsync(this::readPage, executor)
  .thenApply(this::getImageUrl)
  .thenComposeAsync(this::readPage)
  .thenAccept(this::process);
```

- With Fibers:

```
try (var scope = FiberScope.open()) {
   for (int i = 0; i < NUMBER_TO_SHOW; i++) {
      LocalDate date = ...;
      scope.schedule(() -> {
         String page = new String(readPage(getUrlForDate(date)));
         byte[] image = readPage(getImageUrl(page));
         process(image);
      });
   }
}
```

API Under Construction

Joker<?>

- Is a fiber a kind of thread?
- `java.lang.Thread` has accreted a good amount of cruft
  - Priorities
  - Thread groups
  - Thread locals
  - Context class loader
  - `stop`, `suspend`, `resume`
- Loom team investigating different approaches
  - `Fiber extends Thread`
  - `Fiber extends Strand`, `Thread extends Strand`
  - No relationship between `Fiber` and `Thread`
  - Fiber is a `Thread` with a “lightweight” attribute
- Pain points
  - Footprint
  - Compatibility
  - Serviceability

API Under Construction

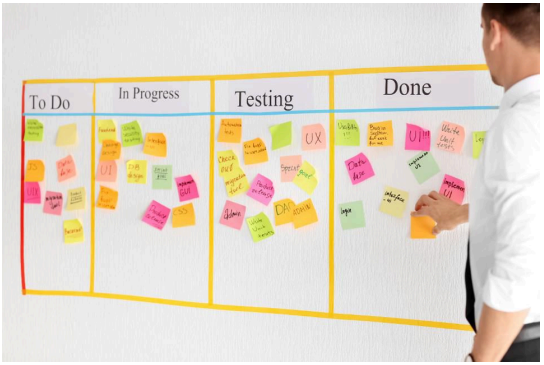
Joker<?>

- What about `Thread.currentThread()`
  - Currently returns “shadow `Thread` object”
  - The actual carrier threads are never exposed
- Don't really want fiber locals
  - Task locals?
  - Processor locals?
  - Scope locals?
- Scopes and scope locals

State of the Project

Joker<?>

- API implementations are being made fiber friendly
  - `Thread.sleep`
  - `j.u.c` Locks
  - NIO, sockets
  - JSSE implementation of TLS
- Reimplementations already in JDK 11, 12, 13
- Can't yet block on monitors (`ReentrantLock` is ok)
- Working on debugger, monitoring support
- Lots of “instabilities”
- Performance nowhere near where it needs to be
- Need help with testing
- Now is a good time to get involved



The End

Joker<?>

- Java is not done
- Major structural changes get a lot of attention by very smart people
- So do features of lesser importance
- But the interests of application programmers are not always represented
- They need you!
- Read those JEPs
- Read the Amber, Valhalla, Loom, Panama, etc. project pages
  - And the documents they reference
- Try early builds
- Give feedback
  - Mailing lists
  - Blogs
- Data is gold