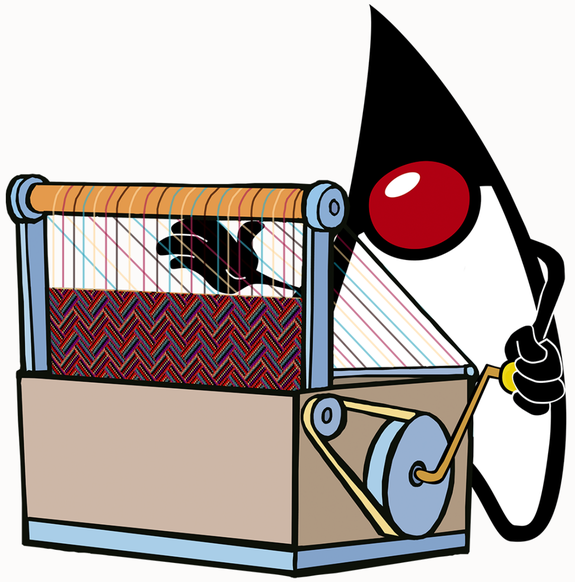
Structured Concurrency



Alan Bateman

Java Platform Group, Oracle

March 21, 2025

Project Loom

An upgrade to Java's concurrency model

- Enables *high-scale* server applications

- Adds *virtual threads*, lightweight threads where each thread runs a single task
  - Think "thread per request", "thread per task", "thread per connection"
  - 1M+ concurrent network connections with a thread per connection on commodity hardware
  - Uses the familiar `java.lang.Thread` API, but more a more scalable implementation
  - Readable, sequential code with understandable control flow
  - Obviates the need for contorted code that we see with async or "reactive" frameworks
  - Support better observability / debuggability at scale

Virtual Threads

- Permanent feature since JDK 21
- Well received by developers and eco system
- Significant interest in avoiding async/reactive

  Go back to simpler synchronous/blocking code instead!
- Frameworks learning how to expose virtual threads to developers
- Performance is good, reliability is good
- Main pain point has been pinning with object monitors, addressed in JDK 24

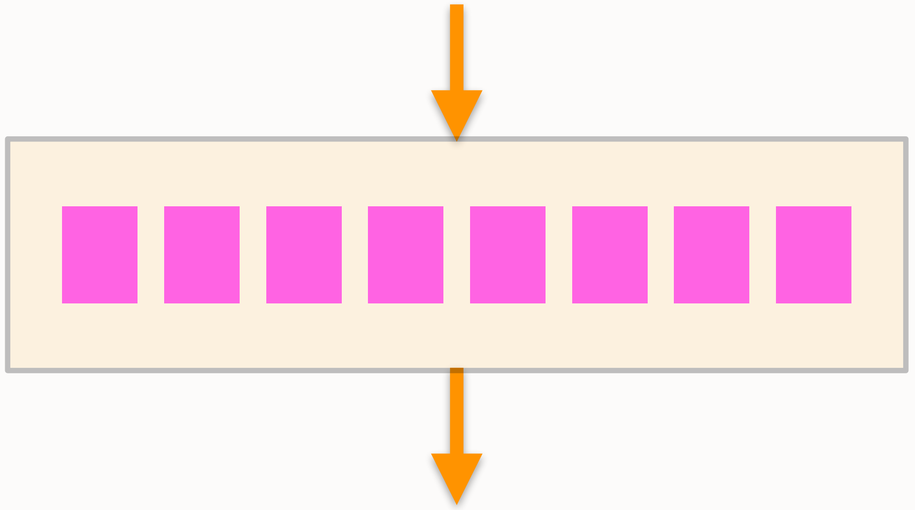
Other features

- Grouping Threads
  - Structured Concurrency, currently in preview

- Better alternatives to Thread Locals
  - Scoped Values, currently in preview

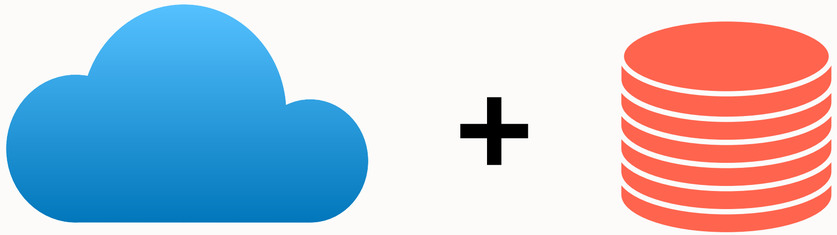
What is Structured Concurrency?

- An approach to concurrent programming that:
  - Simplifies cases where a main tasks splits into several subtasks
  - Preserves the natural relationship between tasks and subtasks
  - Reduces bugs that arise from cancellation and shutdown
  - A great match for virtual threads



Motivating example

- Combine the result from some web service with data fetched from a database



Sequential implementation

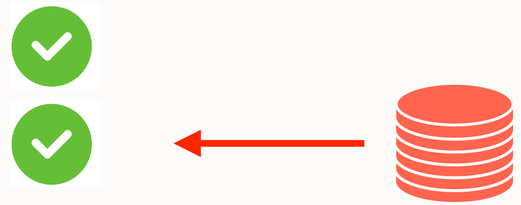
```
String user = fetchUser();

int order = fetchOrder();

var result = new Result(user, order);
```

Sequential implementation

```
String user = fetchUser();

int order = fetchOrder();

var result = new Result(user, order);
```

Sequential implementation

```
String user = fetchUser();
int order = fetchOrder();
var result = new Result(user, order);
```



Sequential implementation

```
String user = fetchUser();
int order = fetchOrder();
var result = new Result(user, order);
```

Sequential implementation

```
String user = fetchUser();

int order = fetchOrder();

var result = new Result(user, order);
```

Sequential implementation

```
String user = fetchUser();
```

```
Exception in thread "main" java.lang.ArithmeticException
        at Test.fetchUser(Test.java:35)
        at Test.main(Test.java:26)
```

Sequential implementation

```
String user = fetchUser();
```

```
int order = fetchOrder();
```

```
Exception in thread "main" java.lang.ArithmeticException
        at Test.fetchOrder(Test.java:35)
        at Test.main(Test.java:26)
```

Concurrent implementation

```
private static final ExecutorService THREAD_POOL = Executors.newFixedThreadPool(8);
```

Concurrent implementation

```
Future<String> task1 = THREAD_POOL.submit(() -> fetchUser());

Future<Integer> task2 = THREAD_POOL.submit(() -> fetchOrder());


var result = new Result(task1.get(), task2.get());
```

Concurrent implementation

Take 1

```
Future<String> task1 = THREAD_POOL.submit(() -> fetchUser());
Future<Integer> task2 = THREAD_POOL.submit(() -> fetchOrder());

var result = new Result(task1.get(), task2.get());
```

Concurrent implementation

```
Future<String> task1 = THREAD_POOL.submit(() -> fetchUser());

Future<Integer> task2 = THREAD_POOL.submit(() -> fetchOrder());


var result = new Result(task1.get(), task2.get());
```

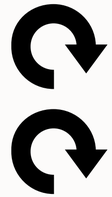
Concurrent implementation

```
Future<String> task1 = THREAD_POOL.submit(() -> fetchUser());
Future<Integer> task2 = THREAD_POOL.submit(() -> fetchOrder());

var result = new Result(task1.get(), task2.get());
```


Concurrent implementation

Take 2

```
Future<String> task1 = THREAD_POOL.submit(() -> fetchUser());
Future<Integer> task2 = THREAD_POOL.submit(() -> fetchOrder());

var result = new Result(task1.get(), task2.get());
```

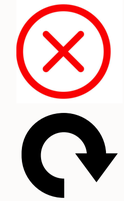


Concurrent implementation

```
Future<String> task1 = THREAD_POOL.submit(() -> fetchUser());

Future<Integer> task2 = THREAD_POOL.submit(() -> fetchOrder());


var result = new Result(task1.get(), task2.get());
```

Concurrent implementation

```
Future<String> task1 = THREAD_POOL.submit(() -> fetchUser());

Future<Integer> task2 = THREAD_POOL.submit(() -> fetchOrder());


var result = new Result(task1.get(), task2.get());
```



```
Exception in thread "main" java.util.concurrent.ExecutionException: java.lang.ArithmeticException
        at java.base/java.util.concurrent.FutureTask.report(FutureTask.java:124)
        at java.base/java.util.concurrent.FutureTask.get(FutureTask.java:193)
        at Test.main(Test.java:10)
Caused by: java.lang.ArithmeticException
        at Test.fetchUser(Test.java:15)
        at Test.lambda$main$0(Test.java:9)
        at java.base/java.util.concurrent.FutureTask.run(FutureTask.java:328)
        at java.base/java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1144)
        at java.base/java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:642)
        at java.base/java.lang.Thread.run(Thread.java:1583)
```

Concurrent implementation

Take 3

```
Future<String> task1 = THREAD_POOL.submit(() -> fetchUser());
Future<Integer> task2 = THREAD_POOL.submit(() -> fetchOrder());

var result = new Result(task1.get(), task2.get());
```

Concurrent implementation

```
Future<String> task1 = THREAD_POOL.submit(() -> fetchUser());

Future<Integer> task2 = THREAD_POOL.submit(() -> fetchOrder());


var result = new Result(task1.get(), task2.get());
```

Concurrent implementation

```
Future<String> task1 = THREAD_POOL.submit(() -> fetchUser());

Future<Integer> task2 = THREAD_POOL.submit(() -> fetchOrder());


var result = new Result(task1.get(), task2.get());
```

Concurrent implementation

```
Future<String> task1 = THREAD_POOL.submit(() -> fetchUser());

Future<Integer> task2 = THREAD_POOL.submit(() -> fetchOrder());


var result = new Result(task1.get(), task2.get());
```

Concurrent implementation

```
Future<String> task1 = THREAD_POOL.submit(() -> fetchUser());

Future<Integer> task2 = THREAD_POOL.submit(() -> fetchOrder());


var result = new Result(task1.get(), task2.get());
```

```
Exception in thread "main" java.util.concurrent.ExecutionException: java.lang.ArithmeticException
        at java.base/java.util.concurrent.FutureTask.report(FutureTask.java:124)
        at java.base/java.util.concurrent.FutureTask.get(FutureTask.java:193)
        at Test.main(Test.java:10)
Caused by: java.lang.ArithmeticException
        at Test.fetchOrder(Test.java:15)
        at Test.lambda$main$0(Test.java:9)
        at java.base/java.util.concurrent.FutureTask.run(FutureTask.java:328)
        at java.base/java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1144)
        at java.base/java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:642)
        at java.base/java.lang.Thread.run(Thread.java:1583)
```

Concurrent implementation

- Fixing the issues means complicated code
- In general, cancellation and shutdown is really hard !!

Project Loom and Structured Concurrency

- Lots of threads quickly leads to thinking about grouping, cancellation, and shutdown
- In 2018, a blog from Nathaniel J. Smith starting using the term “Structured Concurrency”
  - “*A main task splits into several concurrent subtasks, then the subtasks must complete before the main task continues*”
  - Builds on ideas from Martin Sústrik in libdill
  - Can go back further to Elliott Organick “Computer System Organisation”, 1973
- Early API for Structured Concurrency in 2018 pushed to Project Loom repo, working name was *FiberScope*
- Paused and rebooted many times, including prototypes based on *ExecutorService*
- Decided in 2021 to not introduce virtual threads without some support for thread grouping and structured concurrency

Goals for API

- Provide a basic API for Structured Concurrency
- Encourage usage that eliminates many of the bugs that arise due to cancellation and shutdown
- Not a goal to create the *definitive structured concurrency API* for the Java Platform

StructuredTaskScope, preview API since JDK 21

```
try (var scope = new StructuredTaskScope.ShutdownOnFailure()) {
    Subtask<String> subtask1 = scope.fork(() -> fetchUser());
    Subtask<Integer> subtask2 = scope.fork(() -> fetchOrder());
    scope.join().throwIfFailure();
    return new Result(subtask1.get(), subtask2.get());
}
```

Feedback

- API has been in preview for several releases
- Some developers are surprised to see constructors and subclassing
- Some developers disagree with the seat belts that ensure API is used as intended
- Opt-in to propagate exceptions with `throwIfFailed` is a hazard
- Some developers have seen Kotlin coroutines and ask why the `join` method is explicit
- Summary: not a lot of useful feedback

Notes from the project's review of the API/usages

- Subclassing puts constraints on the STS base class on how it may evolve in the future
- When using `ShutdownOnFailure`, easy to forget to `throwIfFailed`
- Subtasks can fork in their parent's scope, we should encourage subtasks to create their own scope
- A deadline or timeout should be on the block, not the `joinUntil` method
- Correlation is problematic
- Not clear where the business logic goes
- Need to validate that *try-with-resources* is the right shape

StructuredTaskScope v2

- Re-explored use-cases and API approaches in 2024, many prototypes
- Came full circle, ended up API that is the same shape but has some changes:
  - Replace public constructors and sub-classing with static factory methods
  - `join` method now returns result or throws, addresses the hazard of `join().throwIfFailed()`
  - Introduce `Joiner` interface to join the results of subtasks and produce the result for `join` method
- API update in Loom EA builds, the (limited) feedback so far has been positive
- JEP submitted with proposal to re-preview with API update

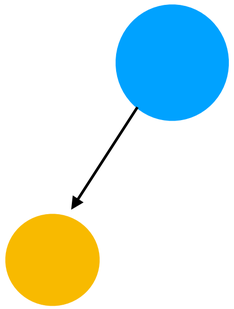
StructuredTaskScope v2

```
try (var scope = StructuredTaskScope.open()) {
    Subtask<String> subtask1 = scope.fork(() -> fetchUser());
    Subtask<Integer> subtask2 = scope.fork(() -> fetchOrder());
    scope.join();
    return new Result(subtask1.get(), subtask2.get());
}
```

StructuredTaskScope

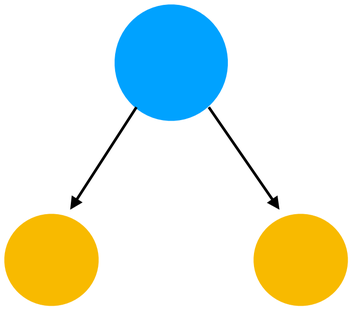
```
try (var scope = StructuredTaskScope.open()) {

    Subtask<String> subtask1 = scope.fork(() -> fetchUser());

    Subtask<Integer> subtask2 = scope.fork(() -> fetchOrder());

    scope.join();

    return new Result(subtask1.get(), subtask2.get());
}
```

StructuredTaskScope



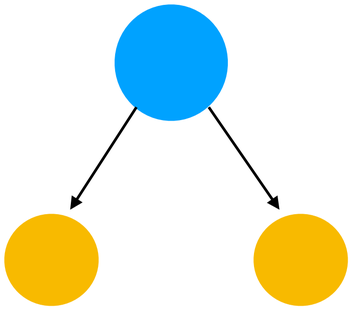
```
try (var scope = StructuredTaskScope.open()) {

    Subtask<String> subtask1 = scope.fork(() -> fetchUser());

    Subtask<Integer> subtask2 = scope.fork(() -> fetchOrder());

    scope.join();

    return new Result(subtask1.get(), subtask2.get());
}
```

StructuredTaskScope



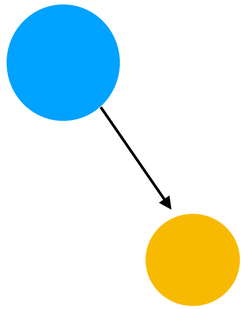
```
try (var scope = StructuredTaskScope.open()) {
    Subtask<String> subtask1 = scope.fork(() -> fetchUser());
    Subtask<Integer> subtask2 = scope.fork(() -> fetchOrder());
    scope.join();
    return new Result(subtask1.get(), subtask2.get());
}
```

StructuredTaskScope



```
try (var scope = StructuredTaskScope.open()) {

    Subtask<String> subtask1 = scope.fork(() -> fetchUser());

    Subtask<Integer> subtask2 = scope.fork(() -> fetchOrder());

    scope.join();

    return new Result(subtask1.get(), subtask2.get());
}
```

StructuredTaskScope



```
try (var scope = StructuredTaskScope.open()) {

    Subtask<String> subtask1 = scope.fork(() -> fetchUser());

    Subtask<Integer> subtask2 = scope.fork(() -> fetchOrder());

    scope.join();

    return new Result(subtask1.get(), subtask2.get());
}
```

StructuredTaskScope

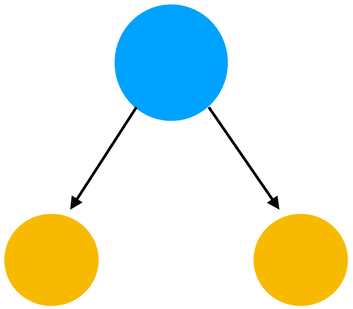
```
try (var scope = StructuredTaskScope.open()) {
    Subtask<String> subtask1 = scope.fork(() -> fetchUser());
    Subtask<Integer> subtask2 = scope.fork(() -> fetchOrder());
    scope.join();
    return new Result(subtask1.get(), subtask2.get());
}
```

StructuredTaskScope

```
try (var scope = StructuredTaskScope.open()) {
    Subtask<String> subtask1 = scope.fork(() -> fetchUser());
    Subtask<Integer> subtask2 = scope.fork(() -> fetchOrder());
    scope.join();
    return new Result(subtask1.get(), subtask2.get());
}
```

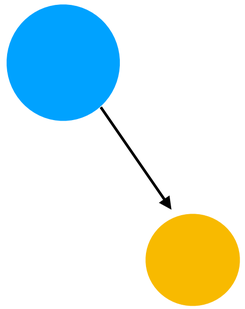
StructuredTaskScope

Take 2



```
try (var scope = StructuredTaskScope.open()) {

    Subtask<String> subtask1 = scope.fork(() -> fetchUser());

    Subtask<Integer> subtask2 = scope.fork(() -> fetchOrder());

    scope.join();

    return new Result(subtask1.get(), subtask2.get());
}
```

StructuredTaskScope



```
try (var scope = StructuredTaskScope.open()) {

    Subtask<String> subtask1 = scope.fork(() -> fetchUser());

    Subtask<Integer> subtask2 = scope.fork(() -> fetchOrder());

    scope.join();

    return new Result(subtask1.get(), subtask2.get());
}
```

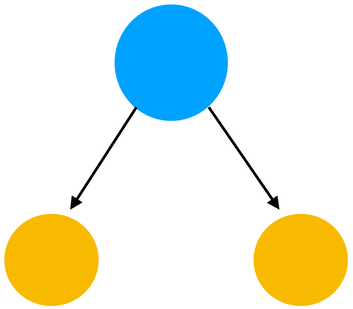
StructuredTaskScope

```
try (var scope = StructuredTaskScope.open()) {

    Subtask<String> subtask1 = scope.fork(() -> fetchUser());

    Subtask<Integer> subtask2 = scope.fork(() -> fetchOrder());

    scope.join();


}
```

```
Exception in thread "main" java.util.concurrent.StructuredTaskScope$FailedException: java.lang.ArithmeticException
        at java.base/java.util.concurrent.StructuredTaskScope.join(StructuredTaskScope.java:1216)
        at Test.test(Test.java:29)
        at Test.main(Test.java:14)
Caused by: java.lang.ArithmeticException
        at Test.fetchUser(Test.java:35)
        at Test.lambda$test$1(Test.java:26)
        at java.base/java.util.concurrent.StructuredTaskScope$SubtaskImpl.run(StructuredTaskScope.java:1335)
        at java.base/java.lang.VirtualThread.run(VirtualThread.java:454)
```

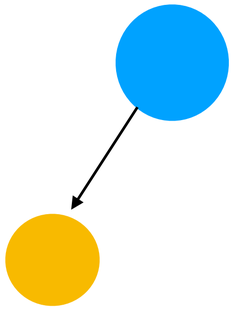
StructuredTaskScope

Take 3



```
try (var scope = StructuredTaskScope.open()) {

    Subtask<String> subtask1 = scope.fork(() -> fetchUser());

    Subtask<Integer> subtask2 = scope.fork(() -> fetchOrder());

    scope.join();

    return new Result(subtask1.get(), subtask2.get());
}
```

StructuredTaskScope



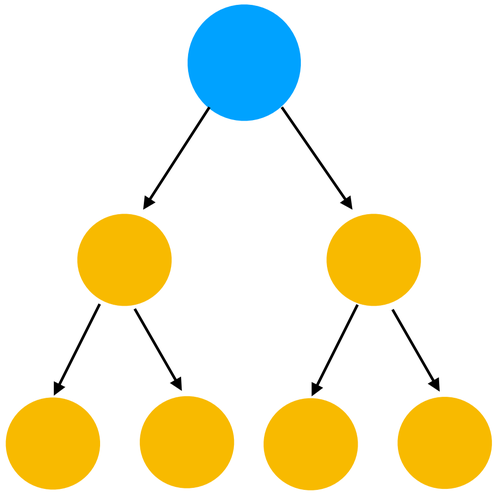
```
try (var scope = StructuredTaskScope.open()) {

    Subtask<String> subtask1 = scope.fork(() -> fetchUser());

    Subtask<Integer> subtask2 = scope.fork(() -> fetchOrder());

    scope.join();

    return new Result(subtask1.get(), subtask2.get());
}
```

StructuredTaskScope

```
try (var scope = StructuredTaskScope.open()) {

    Subtask<String> subtask1 = scope.fork(() -> fetchUser());

    Subtask<Integer> subtask2 = scope.fork(() -> fetchOrder());

    scope.join();


}
```

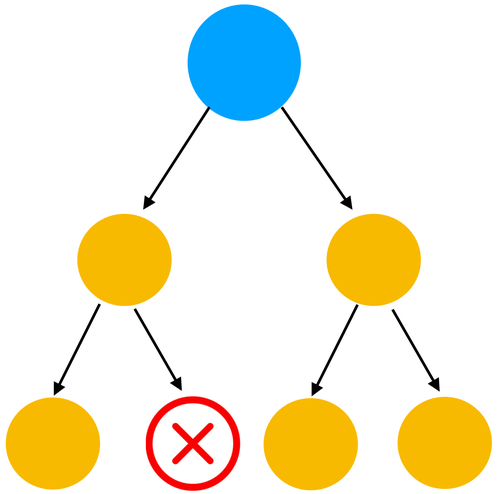
```
Exception in thread "main" java.util.concurrent.StructuredTaskScope$FailedException: java.lang.ArithmeticException
        at java.base/java.util.concurrent.StructuredTaskScope.join(StructuredTaskScope.java:1216)
        at Test.test(Test.java:29)
        at Test.main(Test.java:14)
Caused by: java.lang.ArithmeticException
        at Test.fetchOrder(Test.java:35)
        at Test.lambda$test$1(Test.java:26)
        at java.base/java.util.concurrent.StructuredTaskScope$SubtaskImpl.run(StructuredTaskScope.java:1335)
        at java.base/java.lang.VirtualThread.run(VirtualThread.java:454)
```

Subtasks may open a StructuredTaskScope and fork too



```
try (var scope = StructuredTaskScope.open()) {
    Subtask<String> subtask1 = scope.fork(() -> fetchUser());
    Subtask<Integer> subtask2 = scope.fork(() -> fetchOrder());
    scope.join();
    return new Result(subtask1.get(), subtask2.get());
}
```

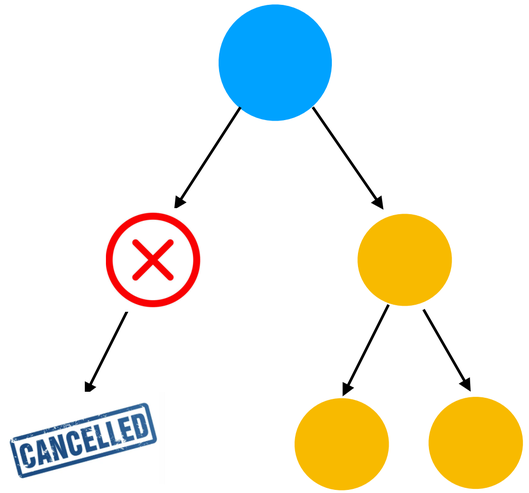
Subtasks may open a StructuredTaskScope and fork too



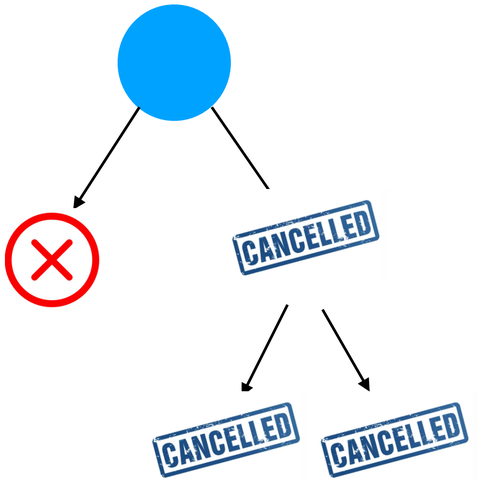
```
try (var scope = StructuredTaskScope.open()) {
    Subtask<String> subtask1 = scope.fork(() -> fetchUser());
    Subtask<Integer> subtask2 = scope.fork(() -> fetchOrder());
    scope.join();
    return new Result(subtask1.get(), subtask2.get());
}
```

Subtasks may open a StructuredTaskScope and fork too

```
try (var scope = StructuredTaskScope.open()) {
    Subtask<String> subtask1 = scope.fork(() -> fetchUser());
    Subtask<Integer> subtask2 = scope.fork(() -> fetchOrder());
    scope.join();
    return new Result(subtask1.get(), subtask2.get());
}
```



Subtasks may open a StructuredTaskScope and fork too




```
try (var scope = StructuredTaskScope.open()) {
    Subtask<String> subtask1 = scope.fork(() -> fetchUser());
    Subtask<Integer> subtask2 = scope.fork(() -> fetchOrder());
    scope.join();
    return new Result(subtask1.get(), subtask2.get());
}
```

Subtasks may open a StructuredTaskScope and fork too

```
try (var scope = StructuredTaskScope.open()) {

    Subtask<String> subtask1 = scope.fork(() -> fetchUser());

    Subtask<Integer> subtask2 = scope.fork(() -> fetchOrder());

    scope.join();

}
```

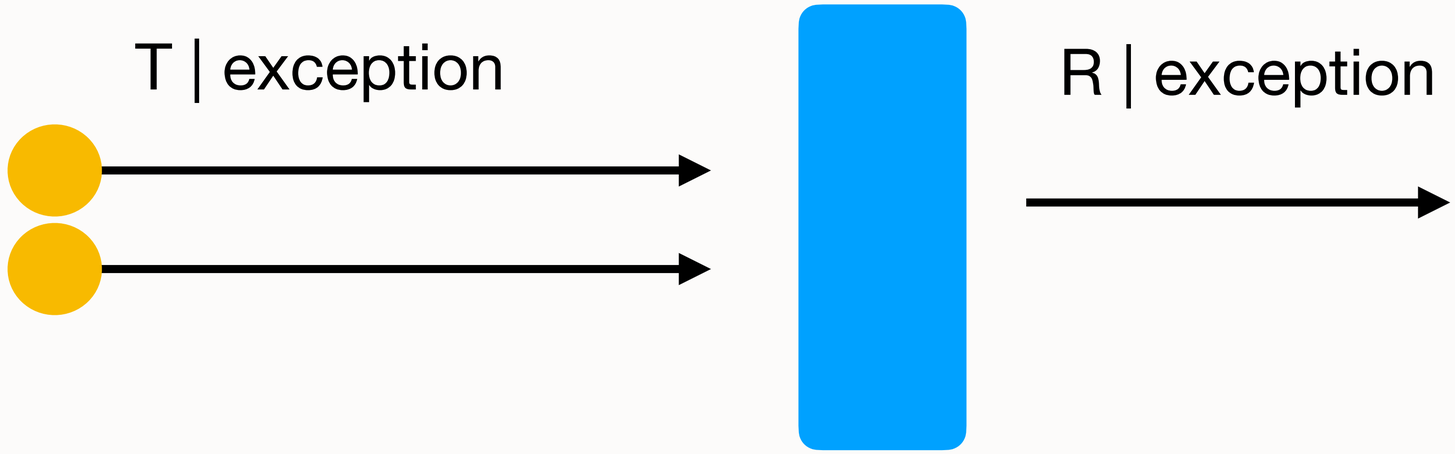
```
Exception in thread "main" java.util.concurrent.StructuredTaskScope$FailedException: java.util.concurrent.StructuredTaskScop
Test$TyreBlowoutException
  at java.base/java.util.concurrent.StructuredTaskScope.join(StructuredTaskScope.java:1216)
  at Test.main(Test.java:13)
Caused by: java.util.concurrent.StructuredTaskScope$FailedException: Test$TyreBlowoutException
  at java.base/java.util.concurrent.StructuredTaskScope.join(StructuredTaskScope.java:1216)
  at Test.fetchUser(Test.java:26)
  at Test.lambda$main$0(Test.java:11)
  at java.base/java.util.concurrent.StructuredTaskScope$SubtaskImpl.run(StructuredTaskScope.java:1335)
  at java.base/java.lang.VirtualThread.run(VirtualThread.java:454)
Caused by: Test$TyreBlowoutException
  at Test.lambda$fetchUser$0(Test.java:24)
  ... 2 more
```

StructuredTaskScope

- By default, `join` method waits for all subtasks to succeed, or throws if a subtask fails
- Existing Preview API: Extend STS to implement other *policies* and *outcome*
- New Preview API: Implement policy in a side object that we call `Joiner`

What’s a Joiner<T, R> ?

- Handles subtask completion and produces the result for the `join` method



A || B

```
<T> T testAnySuccessful(List<Callable<T>> tasks) throws InterruptedException {

    try (var scope = StructuredTaskScope.open(Joiner.<T>anySuccessfulResultOrThrow())) {
        tasks.forEach(scope::fork);
        return scope.join();
    }

}
```

A && B

```
<T> List<T> testAllSuccessful(List<Callable<T>> tasks) throws InterruptedException {

    try (var scope = StructuredTaskScope.open(Joiner.<T>allSuccessfulOrThrow())) {
        tasks.forEach(scope::fork);
        return scope.join().map(Subtask::get).toList();
    }

}
```

Custom Joiner

```
// tolerate up to 3 failures
var failedCount = new AtomicInteger();
var joiner = Joiner.<T>allUntil(s -> s.state() == Subtask.State.FAILED
                                && failedCount.incrementAndGet() >= 3);

List<Callable<T>> tasks = ..
Duration timeout = ..

try (var scope = StructuredTaskScope.open(joiner)) {
    tasks.forEach(scope::fork);
    List<T> results = scope.join()
                           .filter(s -> s.state() == Subtask.State.SUCCESS)
                           .map(Subtask::get)
                           .toList();
}
```

Configuration

```
List<Callable<T>> tasks = ..
Duration timeout = ..

try (var scope = StructuredTaskScope.open(Joiner.<T>allSuccessfulOrThrow(),
                                          cf -> cf.withTimeout(timeout))) {

    tasks.forEach(scope::fork);

    return scope.join().map(Subtask::get).toList();

}
```

StructuredTaskScope summary

- Herding threads, error handling, and cancellation is hard
- We think Structured Concurrency is a good approach to avoid and reduce bugs
- StructuredTaskScope API is small and powerful, but not intended to be the last word on the topic
- We would like to get more feedback from real world usages
- Embracing structured concurrency likely means supporting “structure” in other APIs in the future

Links

- JEP 499: Structured Concurrency (Fourth Preview) in JDK 24
  https://openjdk.org/jeps/499
- JEP 8340343: Structured Concurrency (Fifth Preview), submitted
  https://openjdk.org/jeps/8340343
- Early Access builds
  https://jdk.java.net/loom
- Mailing list
  https://mail.openjdk.org/mailman/listinfo/loom-dev