Overview of Java Structured Concurrency

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

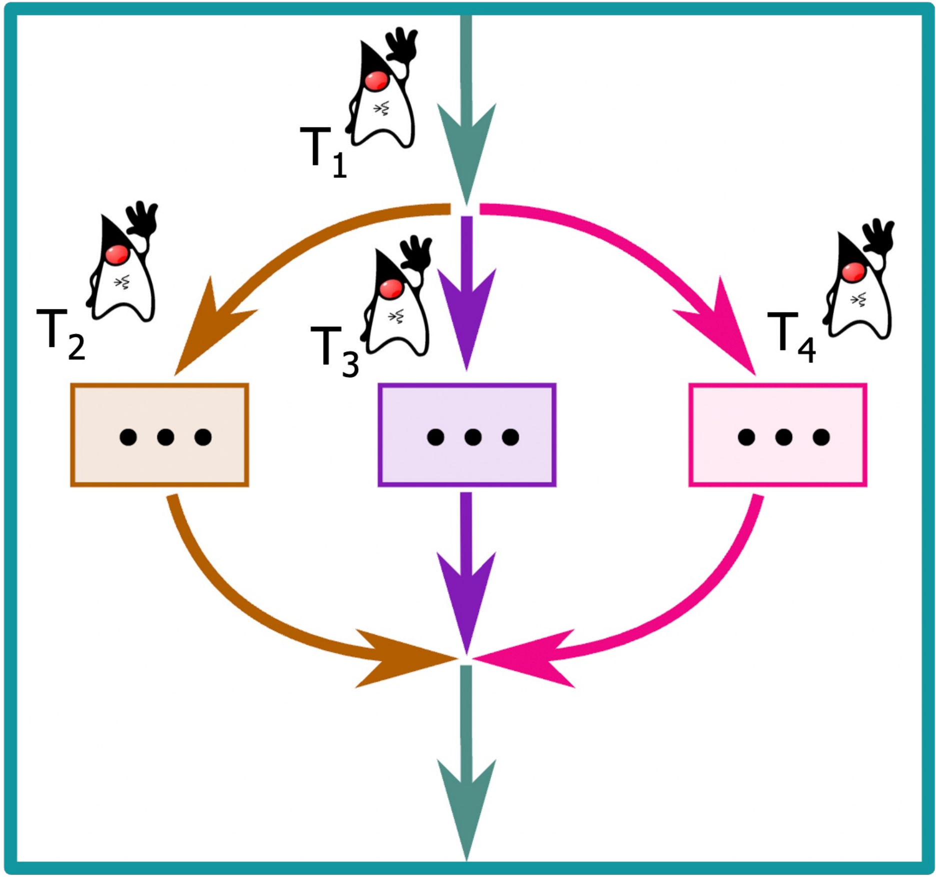
Professor of Computer Science

Institute for Software
Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

- Understand Java's structured concurrency model
  - This model is designed to enable the processing of "embarrassingly parallel" tasks atop the virtual threading mechanisms available in Java 19 (& beyond)



See www.happycoders.eu/java/structured-concurrency-structuredtaskscope

Overview of Java Structured Concurrency

Overview of Java Structured Concurrency

- Structured concurrency was added fairly recently to very modern Java as a concurrent programming paradigm

JEP 428: Structured Concurrency (Incubator)

Authors	Alan Bateman, Ron Pressler
Owner	Alan Bateman
Type	Feature
Scope	JDK
Status	Closed / Delivered
Release	19
Component	core-libs
Discussion	loom dash dev at openjdk dot java dot net
Reviewed by	Alex Buckley, Brian Goetz
Created	2021/11/15 15:01
Updated	2022/08/10 15:58
Issue	8277129

Summary

Simplify multithreaded programming by introducing an API for *structured concurrency*. Structured concurrency treats multiple tasks running in different threads as a single unit of work, thereby streamlining error handling and cancellation, improving reliability, and enhancing observability. This is an incubating API.

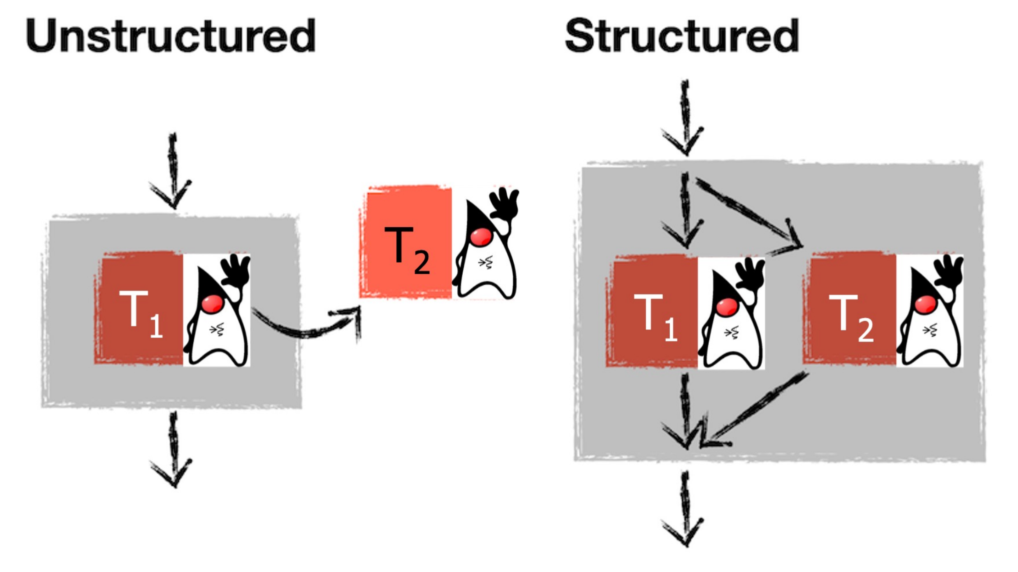
Goals

- Improve the maintainability, reliability, and observability of multithreaded code.
- Promote a style of concurrent programming which can eliminate common risks arising from cancellation and shutdown, such as thread leaks and cancellation delays.

See openjdk.org/jeps/428

Overview of Java Structured Concurrency

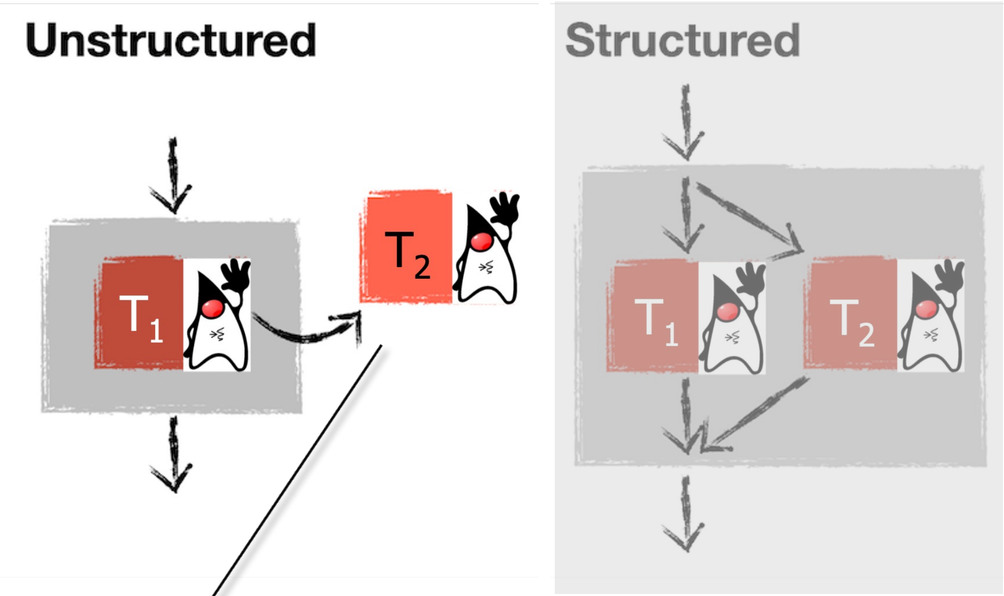
- Structured concurrency was added fairly recently to very modern Java as a concurrent programming paradigm
  - It's intended to make programs easier to read & understand, quicker to write, & safer



See en.wikipedia.org/wiki/Structured_concurrency

Overview of Java Structured Concurrency

- Structured concurrency was added fairly recently to very modern Java as a concurrent programming paradigm
  - It's intended to make programs easier to read & understand, quicker to write, & safer
    - "Safer" == avoiding thread leaks & orphan threads

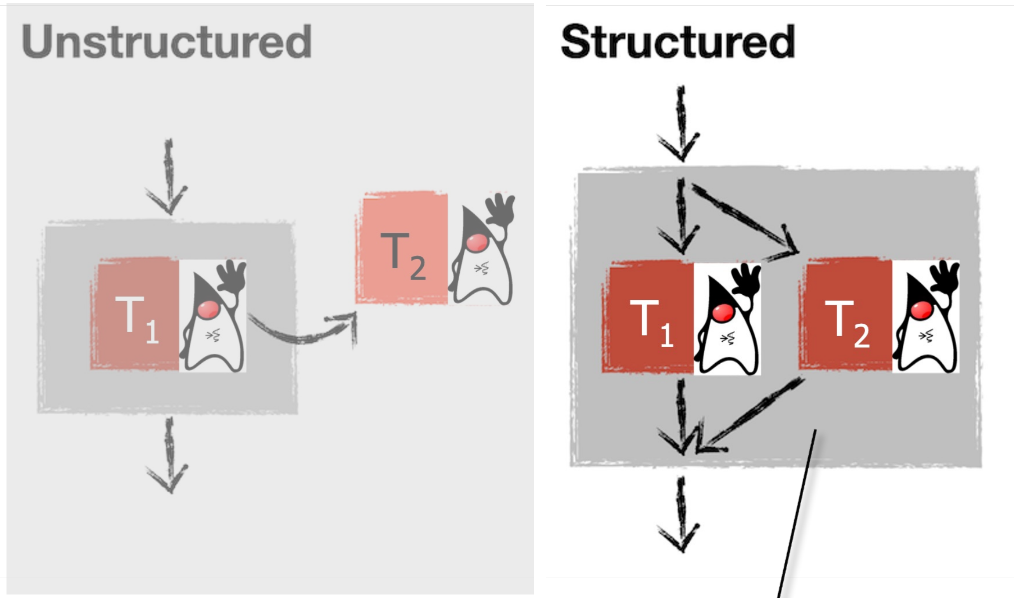


Thread T_2 may become an orphan & leak relative to Thread T_1

See en.wikipedia.org/wiki/Orphan_process

Overview of Java Structured Concurrency

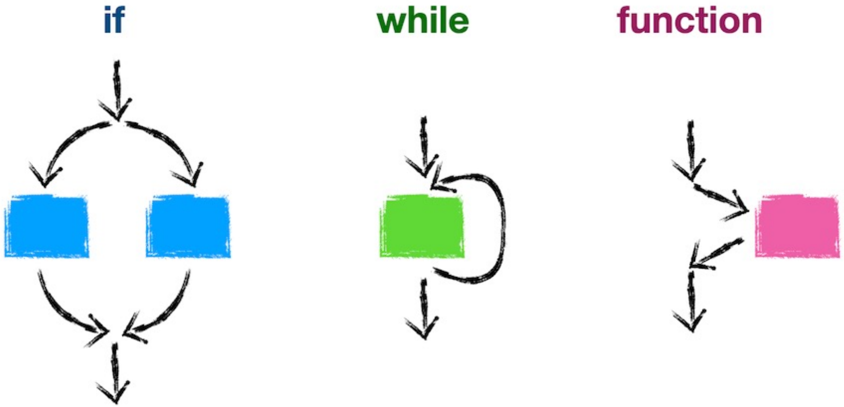
- Structured concurrency was added fairly recently to very modern Java as a concurrent programming paradigm
  - It's intended to make programs easier to read & understand, quicker to write, & safer
    - "Safer" == avoiding thread leaks & orphan threads

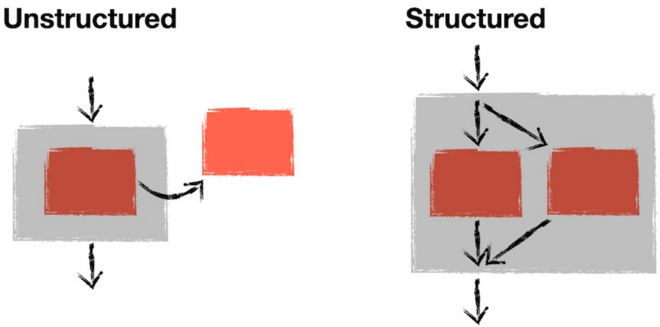


The lifetime of Thread T_1 & Thread T_2 are constrained to the enclosing scope

Overview of Java Structured Concurrency

- Java's structured concurrency paradigm is designed to mimic structured programming

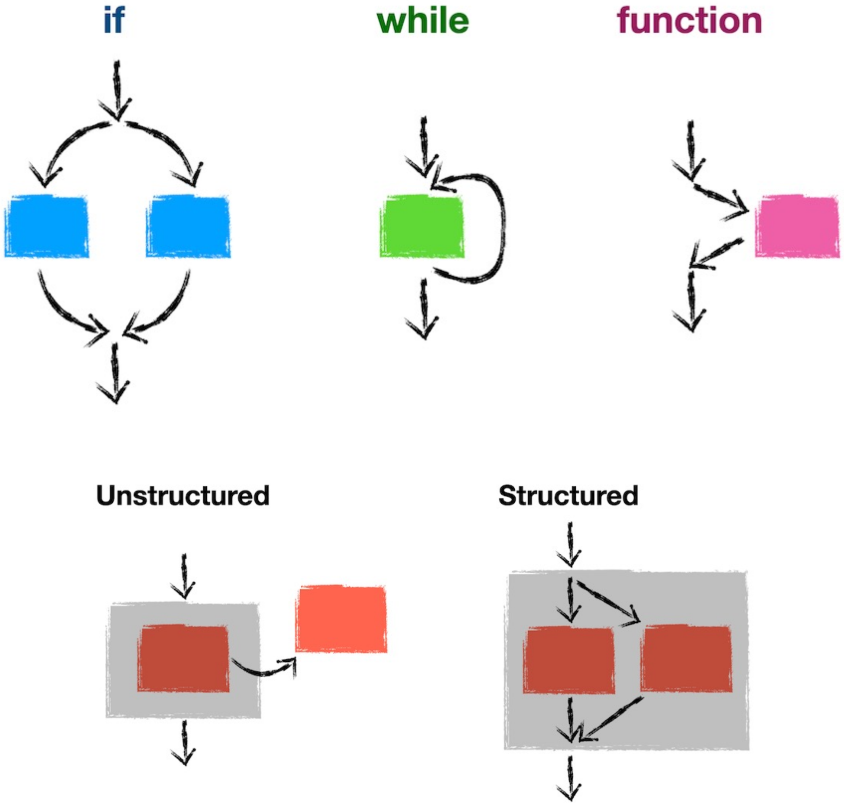




See en.wikipedia.org/wiki/Structured_programming

Overview of Java Structured Concurrency

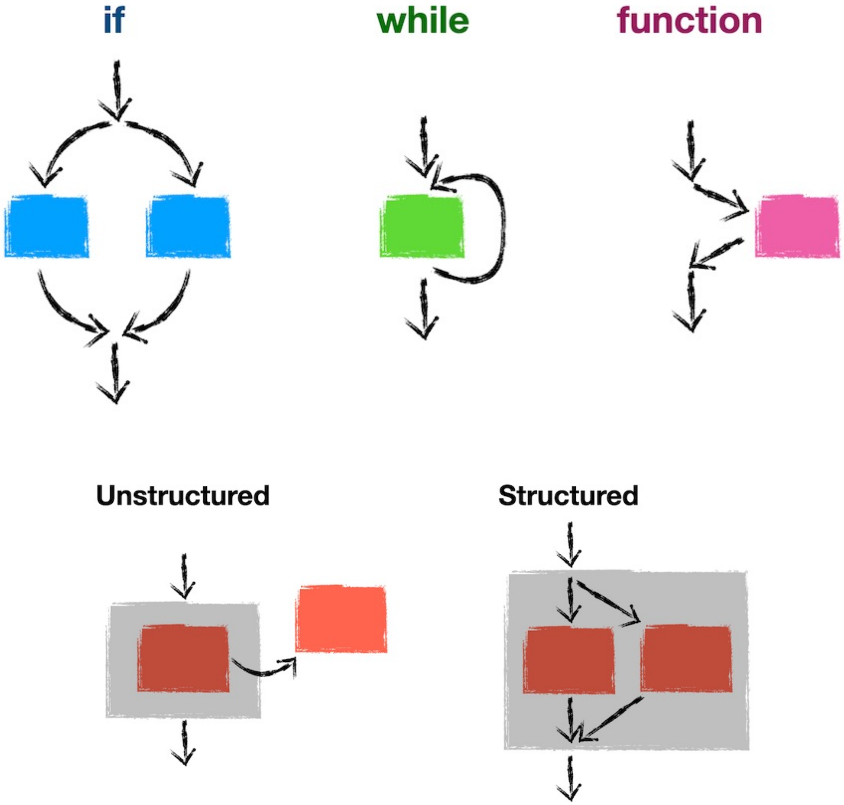
- Java's structured concurrency paradigm is designed to mimic structured programming, i.e.
  - Well-defined entry & exit points for the flow of execution through a block of code



See auroratide.com/posts/understanding-kotlin-coroutines

Overview of Java Structured Concurrency

- Java's structured concurrency paradigm is designed to mimic structured programming, i.e.
  - Well-defined entry & exit points for the flow of execution through a block of code
  - A strict nesting of the lifetimes of operations in a way that mirrors their syntactic nesting in the code



Overview of Java Structured Concurrency

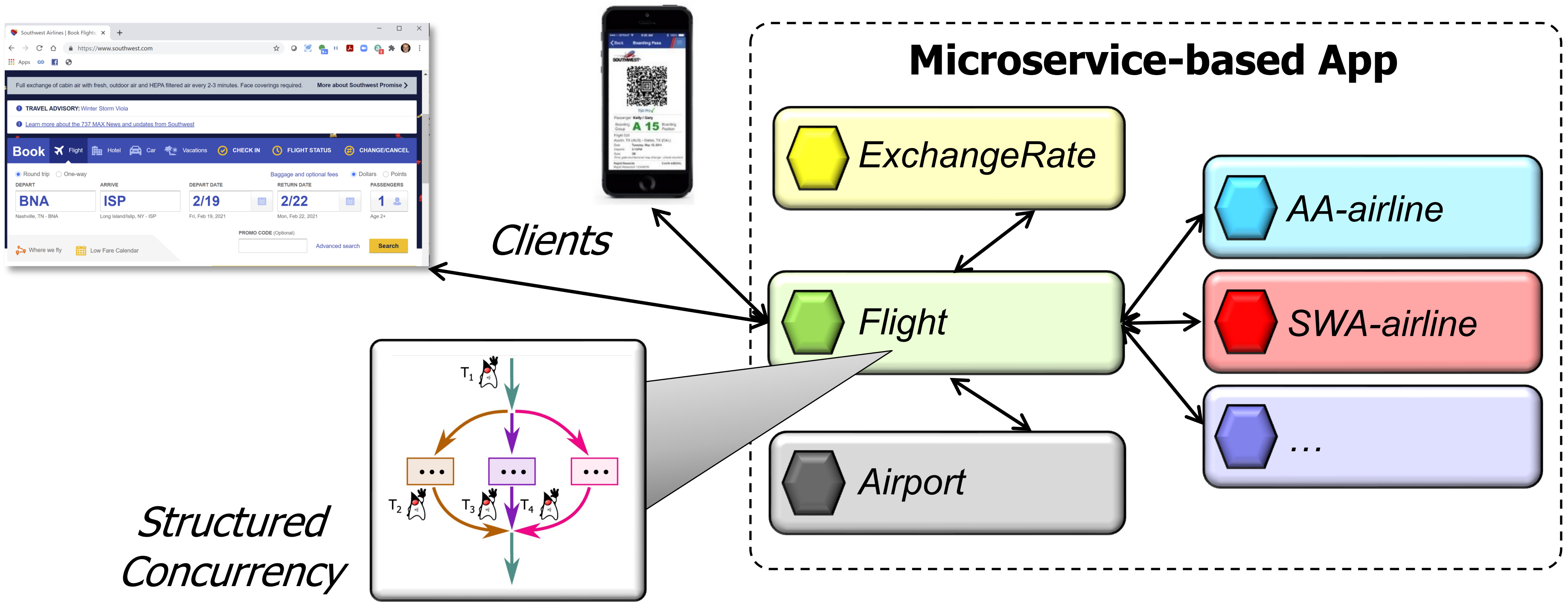
- Java structured concurrency is intended for "embarrassingly parallel" programs

"Embarrassingly parallel" tasks have little/no dependency or need for communication between tasks or for sharing results between them

See en.wikipedia.org/wiki/Embarrassingly_parallel

Overview of Java Structured Concurrency

- Java structured concurrency is intended for "embarrassingly parallel" programs
  - e.g., interacting with many micro-services in a cloud computing environment



See en.wikipedia.org/wiki/Microservices

Java Structured Concurrency Example

Java Structured Concurrency Example

- Java structured concurrency makes the start & end of concurrent code explicit

```
try (var scope = new StructureTaskScope.ShutdownOnFailure()) {
  var results = new ArrayList<Future<BigFraction>>()

  for (var bigFraction :
       generateRandomBigFractions(count))
    results.add(scope
      .fork(() ->
           reduceAndMultiply(bigFraction,
                             sBigReducedFraction));

  scope.join().throwIfFailed();
  ...
  sortAndPrintList(results);
}
```

We will walk through this example quickly now & will explore it in detail later on

See github.com/douglascraigschmidt/LiveLessons/tree/master/Loom/ex3

Java Structured Concurrency Example

- Java structured concurrency makes the start & end of concurrent code explicit

```
try (var scope = new StructureTaskScope.ShutdownOnFailure()) {
  var results = new ArrayList<Future<BigFraction>>()

  for (var bigFraction :
       generateRandomBigFractions(count))
    results.add(scope
      .fork(() ->
           reduceAndMultiply(bigFraction,
                             sBigReducedFraction));

  scope.join().throwIfFailed();
  ...
  sortAndPrintList(results);
}
```

Define a scope for splitting a task into concurrent subtasks that all run complete or first failure

See jdk.incubator.concurrent/jdk/incubator/concurrent/StructuredTaskScope.ShutdownOnFailure.html

Java Structured Concurrency Example

- Java structured concurrency makes the start & end of concurrent code explicit

```
try (var scope = new StructureTaskScope.ShutdownOnFailure()) {
  var results = new ArrayList<Future<BigFraction>>()

  for (var bigFraction :
       generateRandomBigFractions(count))
    results.add(scope
      .fork(() ->
            reduceAndMultiply(bigFraction,
                              sBigReducedFraction));

  scope.join().throwIfFailed();
  ...
  sortAndPrintList(results);
}
```

Start new virtual threads to reduce & multiply random BigFraction objects concurrently

See jdk.incubator.concurrent/jdk/incubator/concurrent/StructuredTaskScope.html#fork

Java Structured Concurrency Example

- Java structured concurrency makes the start & end of concurrent code explicit

```
try (var scope = new StructureTaskScope.ShutdownOnFailure()) {
  var results = new ArrayList<Future<BigFraction>>()

  for (var bigFraction :
       generateRandomBigFractions(count))
    results.add(scope
      .fork(() ->
            reduceAndMultiply(bigFraction,
                              sBigReducedFraction));

  scope.join().throwIfFailed();
  ...
  sortAndPrintList(results);
}
```

Add a Future to each computation result

See docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html#add

Java Structured Concurrency Example

- Java structured concurrency makes the start & end of concurrent code explicit

```
try (var scope = new StructureTaskScope.ShutdownOnFailure()) {
  var results = new ArrayList<Future<BigFraction>>()

  for (var bigFraction :
       generateRandomBigFractions(count))
    results.add(scope
      .fork(() ->
           reduceAndMultiply(bigFraction,
                             sBigReducedFraction));

  scope.join().throwIfFailed();
  ...
  sortAndPrintList(results);
}
```

Wait for all threads to finish or the task scope to shut down if an exception is thrown

See jdk.incubator.concurrent/jdk/incubator/concurrent/StructuredTaskScope.ShutdownOnFailure.html#join()

Java Structured Concurrency Example

- Java structured concurrency makes the start & end of concurrent code explicit

```
try (var scope = new StructureTaskScope.ShutdownOnFailure()) {
  var results = new ArrayList<Future<BigFraction>>()

  for (var bigFraction :
         generateRandomBigFractions(count))
    results.add(scope
      .fork(() ->
           reduceAndMultiply(bigFraction,
                             sBigReducedFraction));

  scope.join().throwIfFailed();
  ...
  sortAndPrintList(results);
}
```

Process the results, which are all stored in completed future objects

Java Structured Concurrency Example

- Java structured concurrency makes the start & end of concurrent code explicit

```
try (var scope = new StructureTaskScope.ShutdownOnFailure()) {
  var results = new ArrayList<Future<BigFraction>>()

  for (var bigFraction :
         generateRandomBigFractions(count))
    results.add(scope
      .fork(() ->
           reduceAndMultiply(bigFraction,
                             sBigReducedFraction));

  scope.join().throwIfFailed();
  ...
  sortAndPrintList(results);
}
```

The close() method of 'scope' is called automatically when the block of code exits

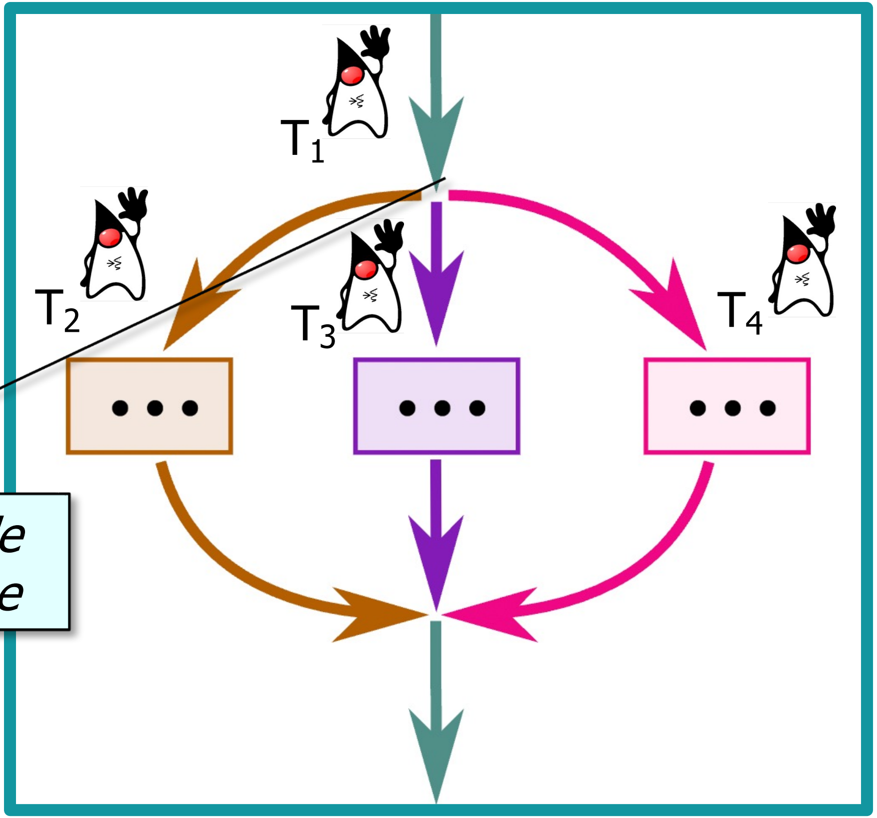
Java Structured Concurrency Benefits

Java Structured Concurrency Benefits

- Java structured concurrency provides several guarantees

Java Structured Concurrency Benefits

- Java structured concurrency provides several guarantees
  - When a program's flow of control is split into multiple threads these threads always complete at the end of a flow

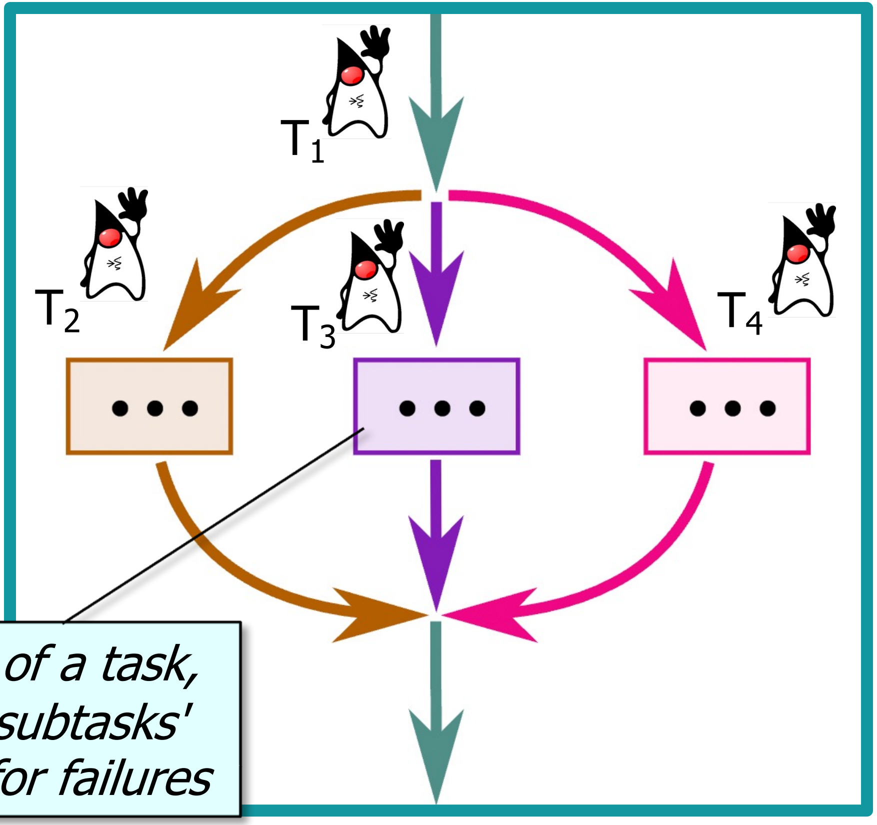


The flow of control splits into multiple threads at the beginning of the scope

See theboreddev.com/understanding-structured-concurrency

Java Structured Concurrency Benefits

- Java structured concurrency provides several guarantees
  - When a program’s flow of control is split into multiple threads these threads always complete at the end of a flow

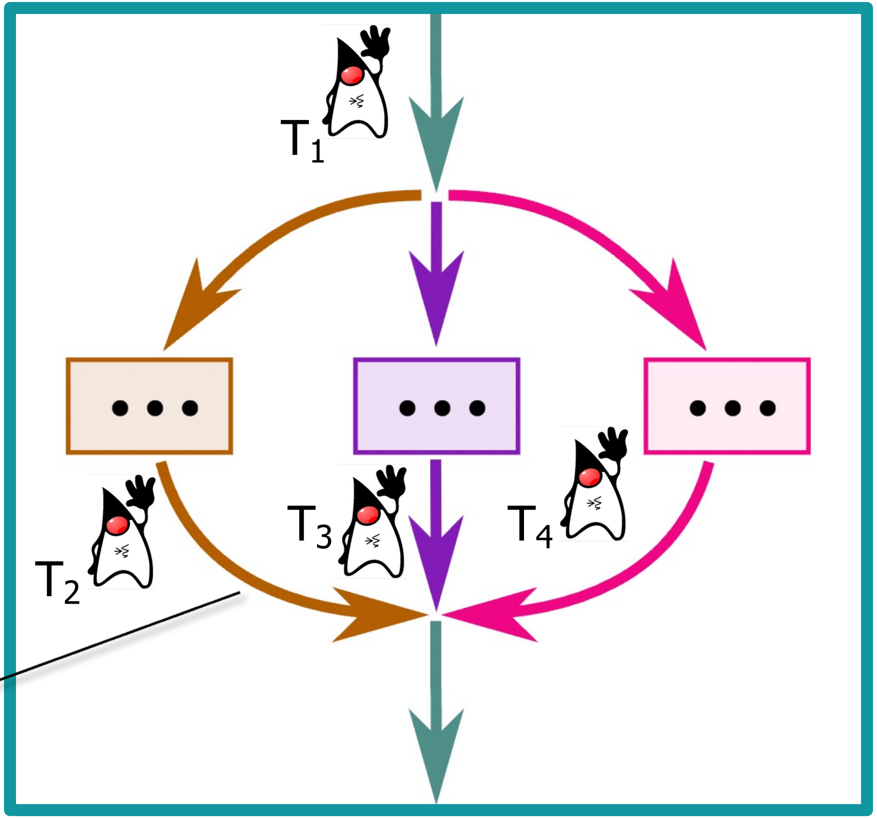


Subtasks work on behalf of a task, i.e., the task awaits the subtasks' results & monitors them for failures

Java Structured Concurrency Benefits

- Java structured concurrency provides several guarantees
  - When a program's flow of control is split into multiple threads these threads always complete at the end of a flow

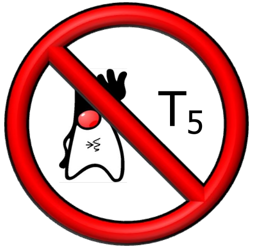
All these threads must complete by the end of the enclosing scope

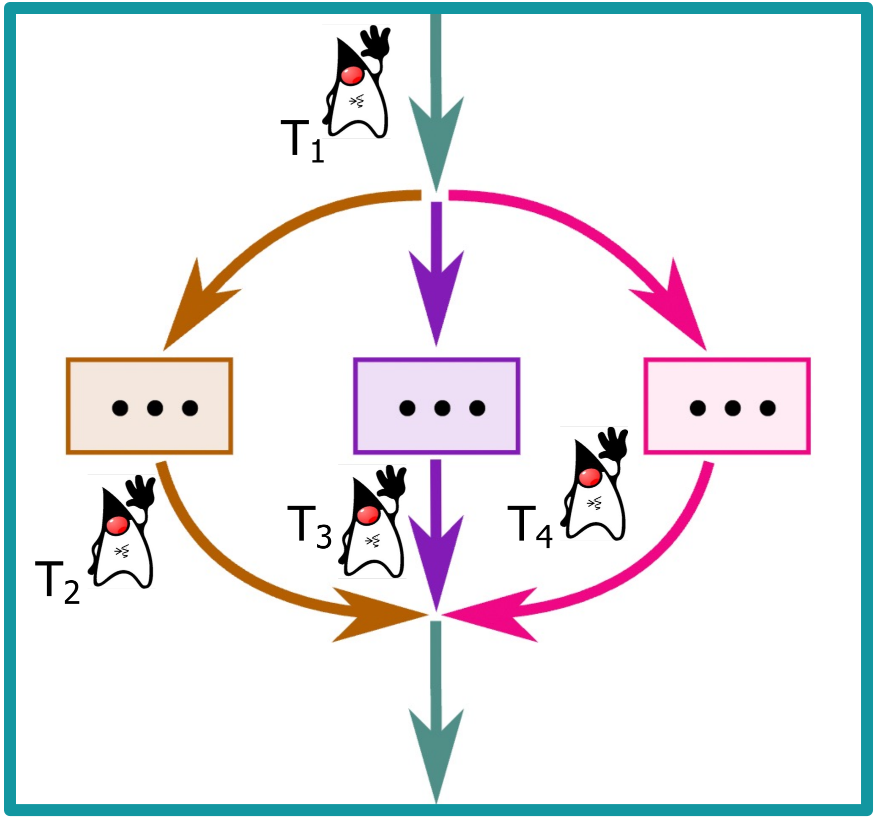


The lifetime of a subtask is confined to the syntactic block of its parent task

Java Structured Concurrency Benefits

- Java structured concurrency provides several guarantees
  - When a program's flow of control is split into multiple threads these threads always complete at the end of a flow
  - No "orphaned threads" occur in an application





End of Overview of Java Structured Concurrency