Programming with 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

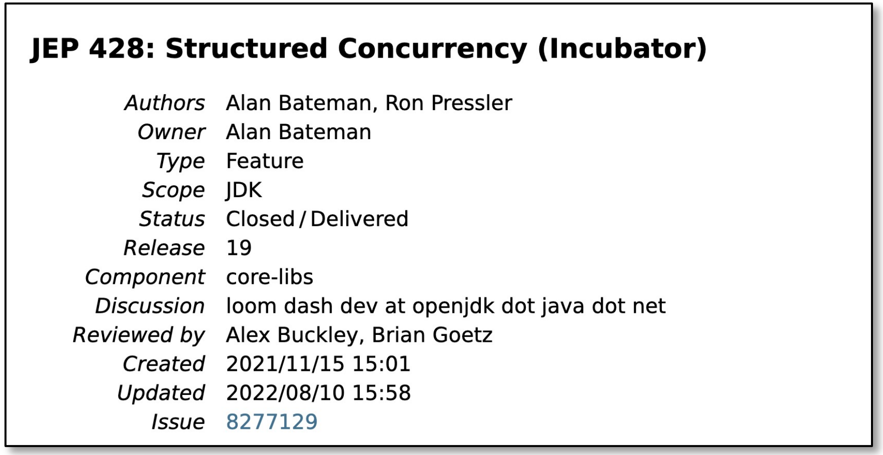
- Understand how Java structured concurrency processes tasks in parallel
- Recognize how to program Java structure concurrency mechanisms

```
try (var scope = new StructuredTaskScope
        .ShutdownOnFailure()) {
  Future<String> user = scope
        .fork(() -> findUser());
  Future<Integer> order = scope
        .fork(() -> fetchOrder());

  scope.join();
  scope.throwIfFailed();

  return new Response
        (user.resultNow(),
         order.resultNow());
}
```

Programming with Java Structured Concurrency

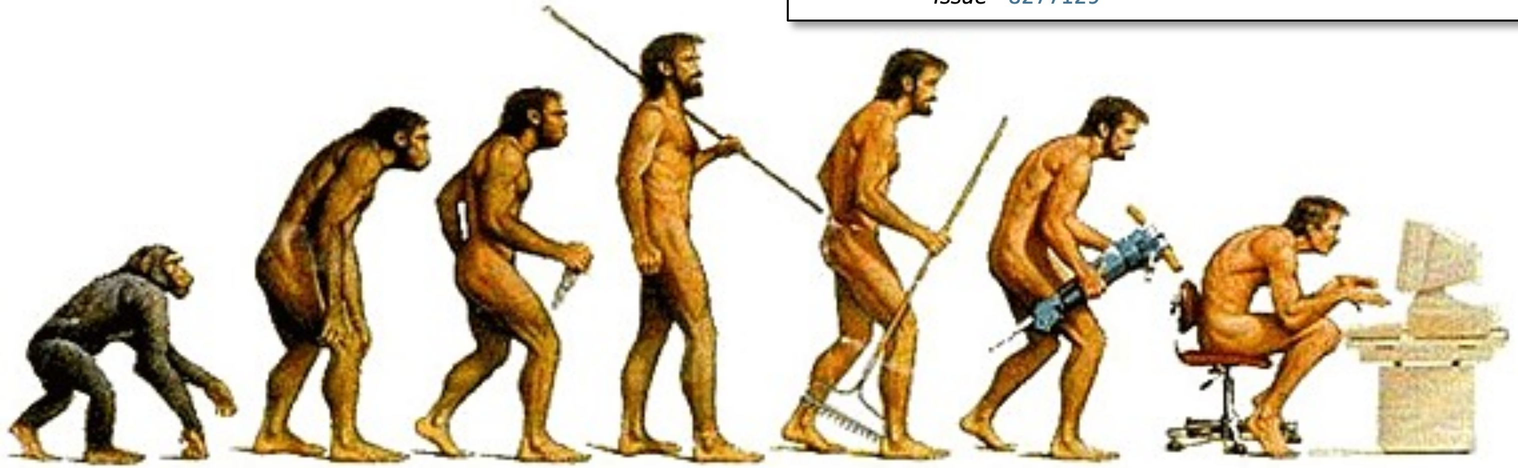
Programming with Java Structured Concurrency

- Java structured concurrency is evolving continuously



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

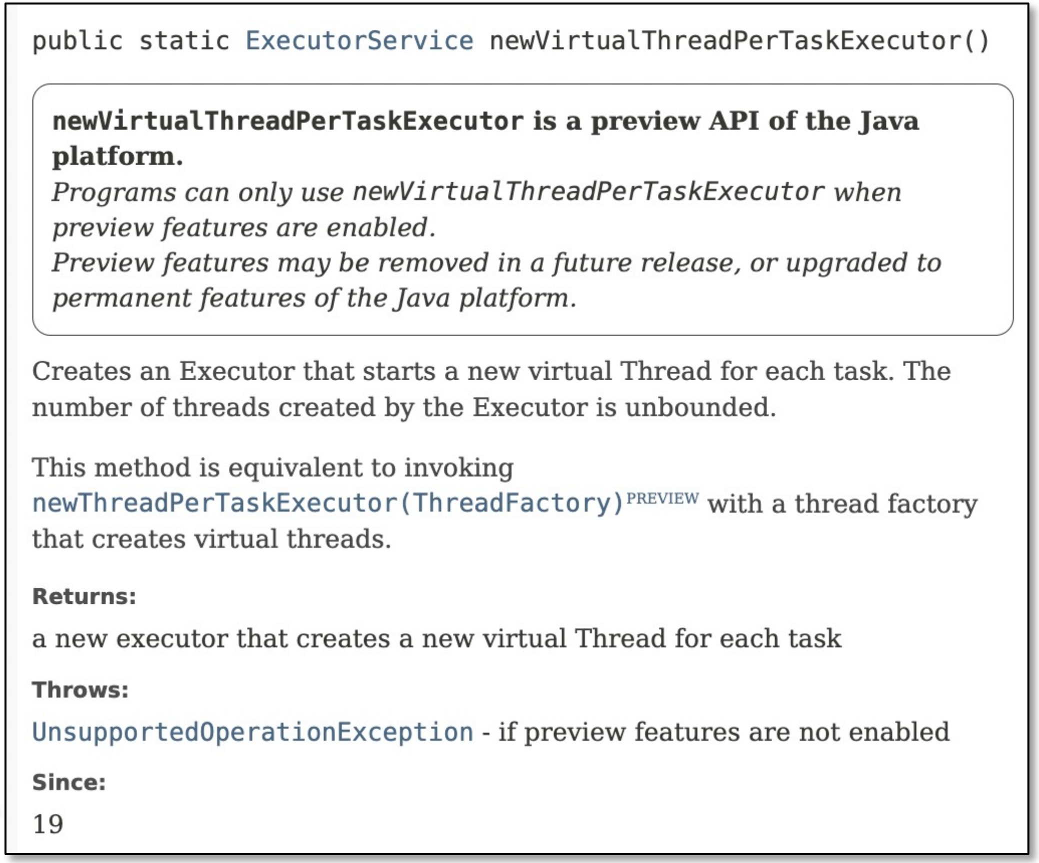


See openjdk.org/jeps/428

Programming with Java Structured Concurrency

- Java structured concurrency is evolving continuously
  - Executors/ExecutorService

Added in Java 19



```
public static ExecutorService newVirtualThreadPerTaskExecutor()
```

newVirtualThreadPerTaskExecutor is a preview API of the Java platform.
Programs can only use newVirtualThreadPerTaskExecutor when preview features are enabled.
Preview features may be removed in a future release, or upgraded to permanent features of the Java platform.

Creates an Executor that starts a new virtual Thread for each task. The number of threads created by the Executor is unbounded.

This method is equivalent to invoking `newThreadPerTaskExecutor(ThreadFactory)`PREVIEW with a thread factory that creates virtual threads.

Returns:

a new executor that creates a new virtual Thread for each task

Throws:

`UnsupportedOperationException` - if preview features are not enabled

Since:

19

See docs.oracle.com/en/java/javase/19/docs/api/java.base/java/util/concurrent/Executors.html#newVirtualThreadPerTaskExecutor()

Programming with Java Structured Concurrency

- Java structured concurrency is evolving continuously
  - Executors/ExecutorService
    - Used with the Java try-with-resources feature

```
try (var executor = Executors
          .newVirtualThreadPerTaskExecutor()){
    IntStream
      .range(0, 10_000)
      .forEach(i -> executor
         .submit(() -> {
            Thread.sleep(Duration
                        .ofSeconds(1));
            return i;
        }));
}
```

Creates an Executor that starts a new virtual Thread for each task

See howtodoinjava.com/java/multi-threading/virtual-threads/#43-using-executorsnewvirtualthreadpertaskexecutor

Programming with Java Structured Concurrency

- Java structured concurrency is evolving continuously
  - Executors/ExecutorService
    - Used with the Java try-with-resources feature
    - This Executor creates a new virtual thread for each request

```
try (var executor = Executors
        .newVirtualThreadPerTaskExecutor()){
   IntStream
     .range(0, 10_000)
     .forEach(i -> executor
        .submit(() -> {
           Thread.sleep(Duration
                        .ofSeconds(1));
           return i;
       }));
}
```

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

See howtodoinjava.com/java/multi-threading/virtual-threads/#43-using-executorsnewvirtualthreadpertaskexecutor

Programming with Java Structured Concurrency

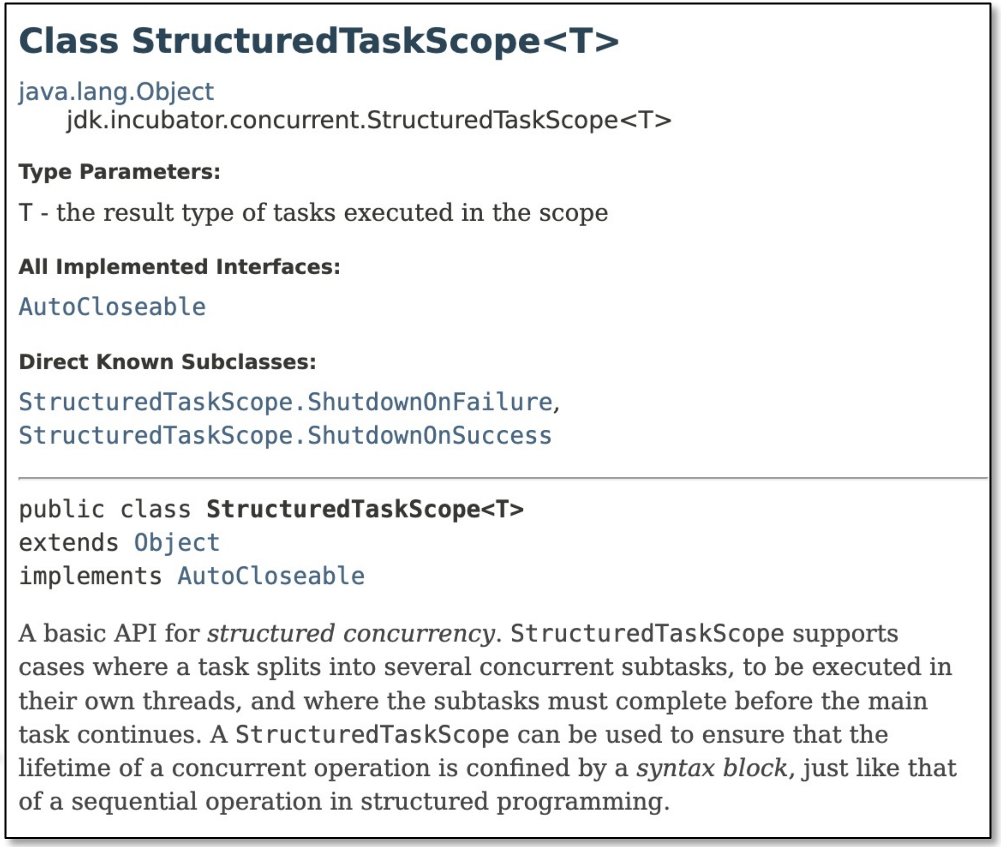
- Java structured concurrency is evolving continuously
  - Executors/ExecutorService
    - Used with the Java try-with-resources feature
    - This Executor creates a new virtual thread for each request
    - The try-with-resources scope is a bit limiting..

```
try (var executor = Executors
          .newVirtualThreadPerTaskExecutor()){
    IntStream
      .range(0, 10_000)
      .forEach(i -> executor
         .submit(() -> {
            Thread.sleep(Duration
                          .ofSeconds(1));
            return i;
         }));
}
```

Programming with Java Structured Concurrency

- Java structured concurrency is evolving continuously
  - Executors/ExecutorService
  - StructuredTaskScope

Added in Java 19 in the "incubator"



Class StructuredTaskScope<T>

java.lang.Object
    jdk.incubator.concurrent.StructuredTaskScope<T>

Type Parameters:

T - the result type of tasks executed in the scope

All Implemented Interfaces:

AutoCloseable

Direct Known Subclasses:

StructuredTaskScope.ShutdownOnFailure, StructuredTaskScope.ShutdownOnSuccess

```
public class StructuredTaskScope<T>
extends Object
implements AutoCloseable
```

A basic API for *structured concurrency*. `StructuredTaskScope` supports cases where a task splits into several concurrent subtasks, to be executed in their own threads, and where the subtasks must complete before the main task continues. A `StructuredTaskScope` can be used to ensure that the lifetime of a concurrent operation is confined by a *syntax block*, just like that of a sequential operation in structured programming.

See download.java.net/java/early_access/loom/docs/api/jdk.incubator.concurrent/jdk/incubator/concurrent/StructuredTaskScope.html

Programming with Java Structured Concurrency

- Java structured concurrency is evolving continuously
  - Executors/ExecutorService
  - StructuredTaskScope
    - Also used with the try-with-resources feature

```
try (var scope = new StructuredTaskScope
        .ShutdownOnFailure()) {
  Future<String> user = scope
      .fork(() -> findUser());
  Future<Integer> order = scope
      .fork(() -> fetchOrder());

  scope.join();
  scope.throwIfFailed();

  return new Response
      (user.resultNow(),
       order.resultNow());
}
```

See howtodoinjava.com/java/multi-threading/structured-concurrency

Programming with Java Structured Concurrency

- Java structured concurrency is evolving continuously
  - Executors/ExecutorService
  - StructuredTaskScope
    - Also used with the try-with-resources feature

Creates a new virtual Thread every time it is called

```
try (var scope = new StructuredTaskScope
        .ShutdownOnFailure()) {
    Future<String> user = scope
        .fork(() -> findUser());
    Future<Integer> order = scope
        .fork(() -> fetchOrder());

    scope.join();
    scope.throwIfFailed();

    return new Response
        (user.resultNow(),
         order.resultNow());
}
```

See download.java.net/java/early_access/loom/docs/api/jdk.incubator.concurrent/jdk/incubator/concurrent/StructuredTaskScope.html#fork

Programming with Java Structured Concurrency

- Java structured concurrency is evolving continuously
  - Executors/ExecutorService
  - StructuredTaskScope
    - Also used with the try-with-resources feature
      - However, it's more flexible due to the join() method

```
try (var scope = new StructuredTaskScope
       .ShutdownOnFailure()) {
  Future<String> user = scope
       .fork(() -> findUser());
  Future<Integer> order = scope
       .fork(() -> fetchOrder());

  scope.join();
  scope.throwIfFailed();

  return new Response
       (user.resultNow(),
        order.resultNow());
}
```

Wait for all threads to finish or the task scope to shut down

See download.java.net/java/early_access/loom/docs/api/jdk.incubator.concurrent/jdk/incubator/concurrent/StructuredTaskScope.ShutdownOnFailure.html#join

Programming with Java Structured Concurrency

- Java structured concurrency is evolving continuously
  - Executors/ExecutorService
  - StructuredTaskScope
    - Also used with the try-with-resources feature
      - However, it's more flexible due to the join() method

```
try (var scope = new StructuredTaskScope
       .ShutdownOnFailure()) {
  Future<String> user = scope
       .fork(() -> findUser());
  Future<Integer> order = scope
       .fork(() -> fetchOrder());

  scope.join();
  scope.throwIfFailed();

  return new Response
       (user.resultNow(),
        order.resultNow());
}
```

Throws an Exception if a sub-task completed abnormally

See download.java.net/java/early_access/loom/docs/api/jdk.incubator.concurrent/jdk/incubator/concurrent/StructuredTaskScope.ShutdownOnFailure.html#throwIfFailed

Programming with Java Structured Concurrency

- Java structured concurrency is evolving continuously
  - Executors/ExecutorService
  - StructuredTaskScope
    - Also used with the try-with-resources feature
      - However, it's more flexible due to the join() method

```
try (var scope = new StructuredTaskScope
       .ShutdownOnFailure()) {
  Future<String> user = scope
       .fork(() -> findUser());
  Future<Integer> order = scope
       .fork(() -> fetchOrder());

  scope.join();
  scope.throwIfFailed();

  return new Response
       (user.resultNow(),
        order.resultNow());
}
```

Return a result using new Future methods

See download.java.net/java/early_access/loom/docs/api/java.base/java/util/concurrent/Future.html#resultNow

End of Programming with Java Structured Concurrency