Programming with Java Structured Concurrency

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

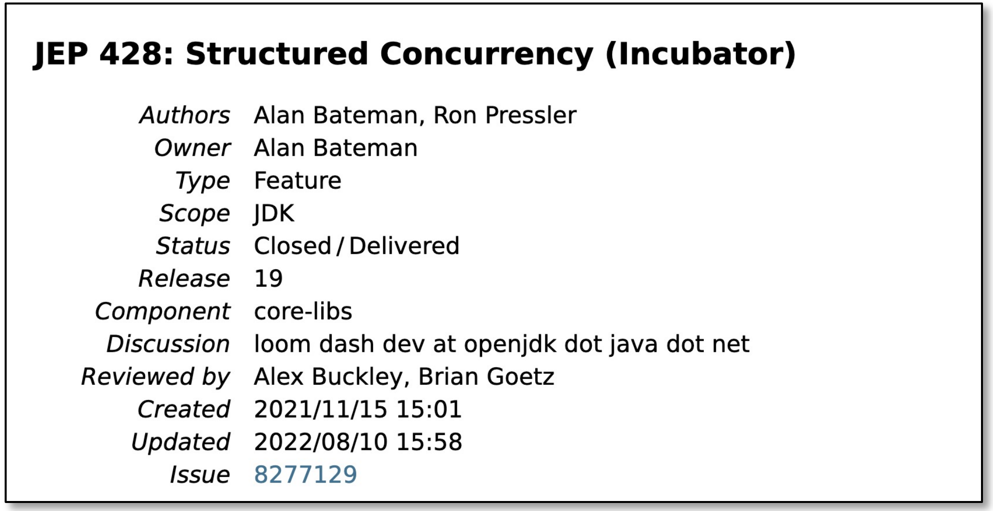
Professor of Computer Science

Institute for Software
Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

- Understand Java's structured concurrency model
- Recognize the classes used to program Java's structure concurrency model



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

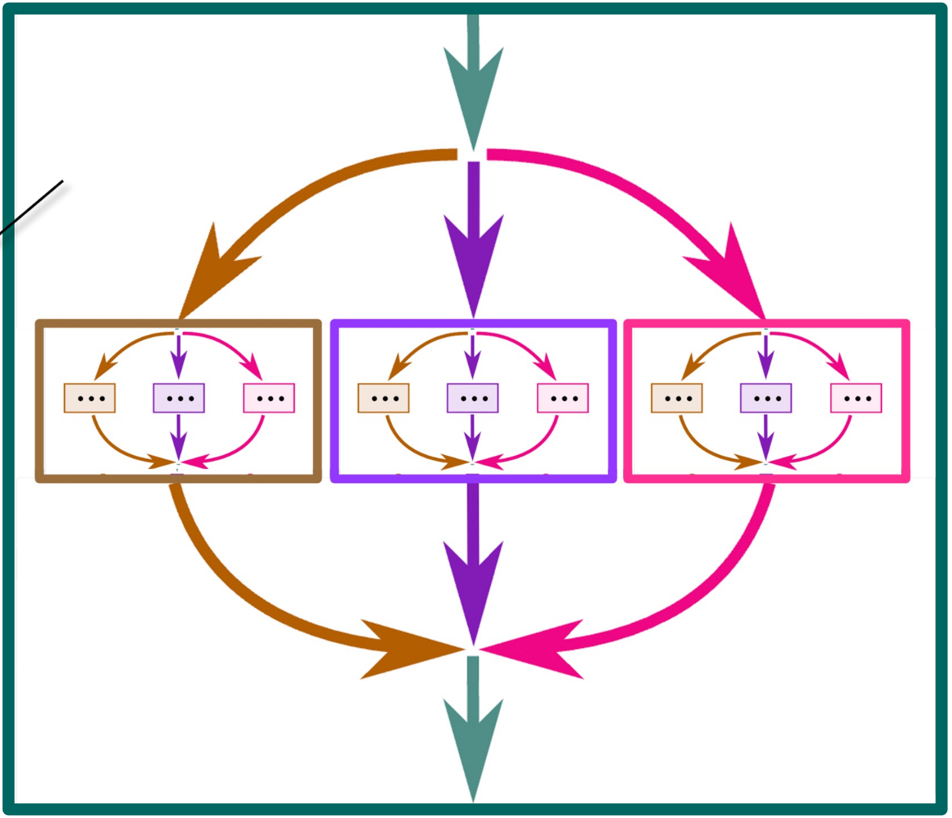
See openjdk.org/jeps/428

Programming with Java Structured Concurrency

Programming with Java Structured Concurrency

- Java structured concurrency enforces a hierarchy of tasks & subtasks

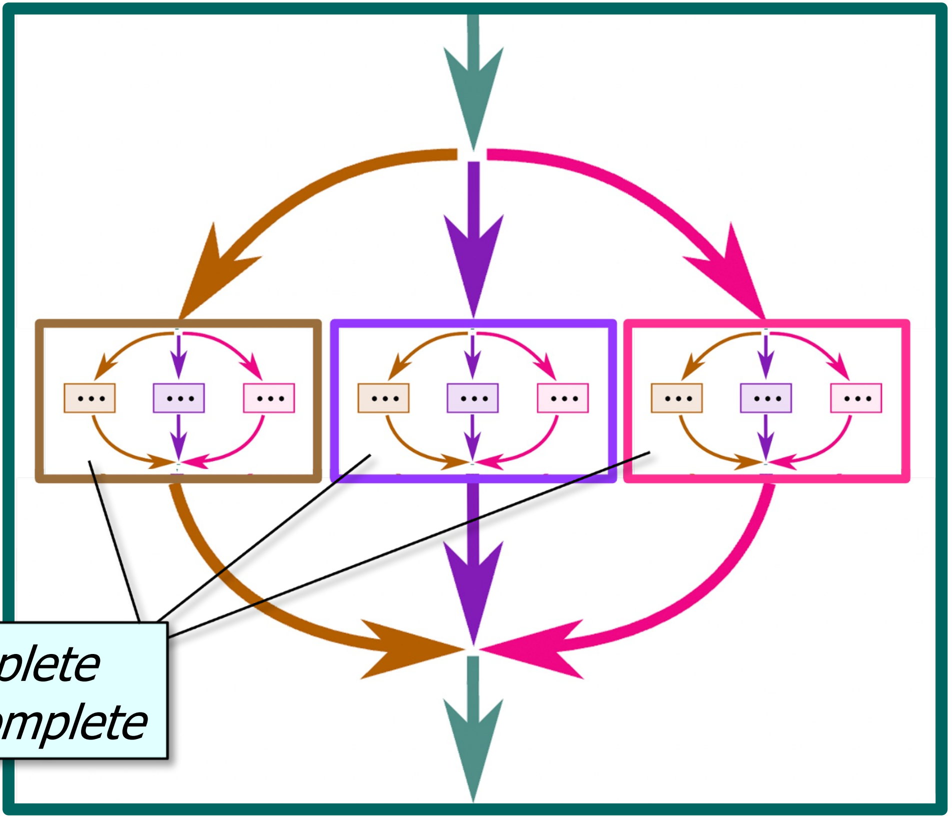
A parent task may contain multiple nested levels of subtasks



See openjdk.org/jeps/428

Programming with Java Structured Concurrency

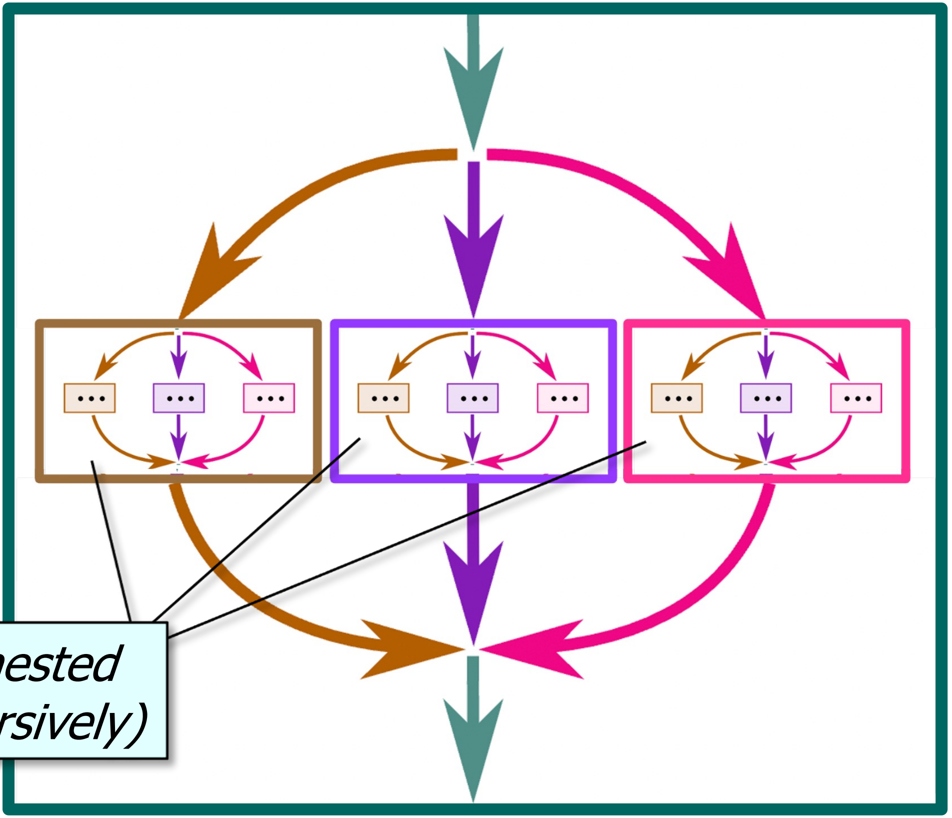
- Java structured concurrency enforces a hierarchy of tasks & subtasks
  - The lifetime of a subtask must be confined to the syntactic block of its parent task



All these subtasks must complete before each parent task can complete

Programming with Java Structured Concurrency

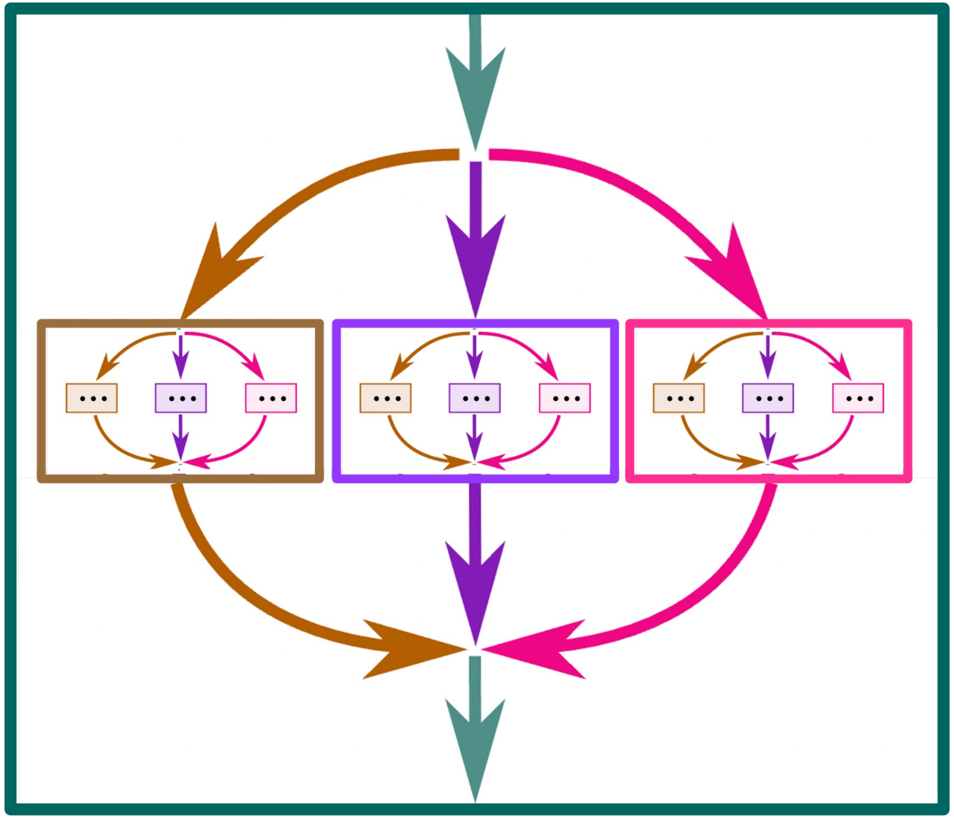
- Java structured concurrency enforces a hierarchy of tasks & subtasks
  - The lifetime of a subtask must be confined to the syntactic block of its parent task
  - Sibling subtask lifetimes are nested within a parent task



These sibling subtasks are nested within their parent task (recursively)

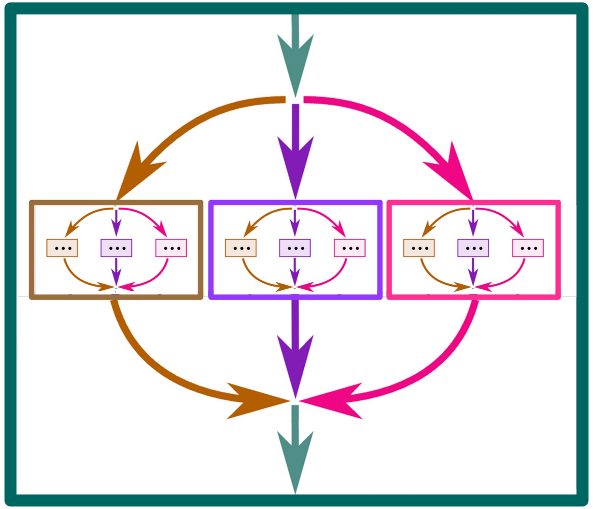
Programming with Java Structured Concurrency

- Java structured concurrency enforces a hierarchy of tasks & subtasks
  - The lifetime of a subtask must be confined to the syntactic block of its parent task
  - Sibling subtask lifetimes are nested within a parent task
    - Tasks (& subtasks) can thus be reasoned about & managed as a unit



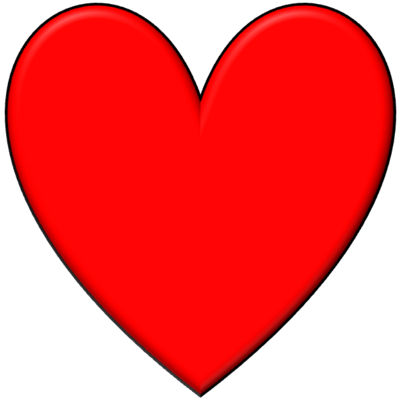
Programming with Java Structured Concurrency

- Structured concurrency is a great match for virtual threads



+

=



See openjdk.org/jeps/428

Programming with Java Structured Concurrency

- Structured concurrency is a great match for virtual threads
  - Virtual threads are lightweight, so they can represent any concurrent unit of behavior

```
try (var scope = new
          StructuredTaskScope
           .ShutdownOnFailure()) {
  var downloadedImages = ...;

  for (URL url : urlList)
    downloadededImages.add(scope
      .fork(() ->
         downloadImage(url)));

  scope.join();

  return downloadedImages;
}
```

Even behavior that involves I/O!

See github.com/douglascraigschmidt/LiveLessons/tree/master/Loom/ex4

Programming with Java Structured Concurrency

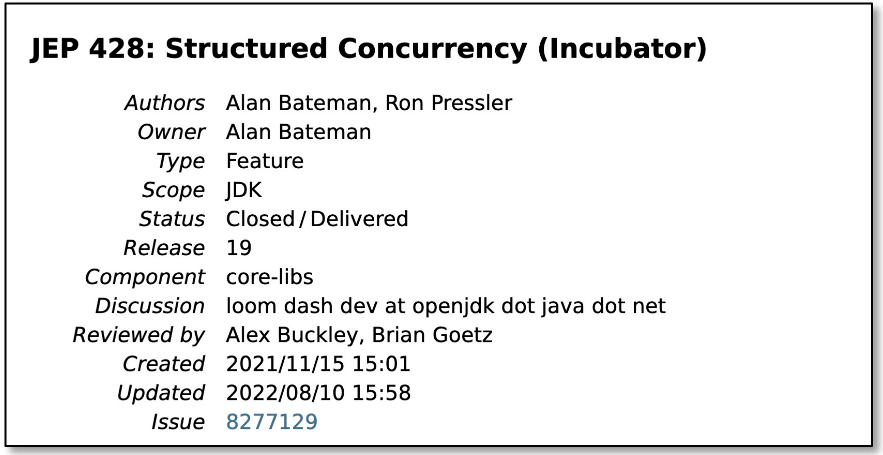
- Structured concurrency is a great match for virtual threads
  - Virtual threads are lightweight, so they can represent any concurrent unit of behavior
  - Structured concurrency ensures that virtual threads are correctly & robustly coordinated

```
try (var scope = new
        StructuredTaskScope
         .ShutdownOnFailure()) {
  var downloadedImages = ...;

  for (URL url : urlList)
    downloadededImages.add(scope
      .fork(() ->
           downloadImage(url)));

  scope.join();

  return downloadedImages;
}
```

This block of code doesn't exit until all images are downloaded

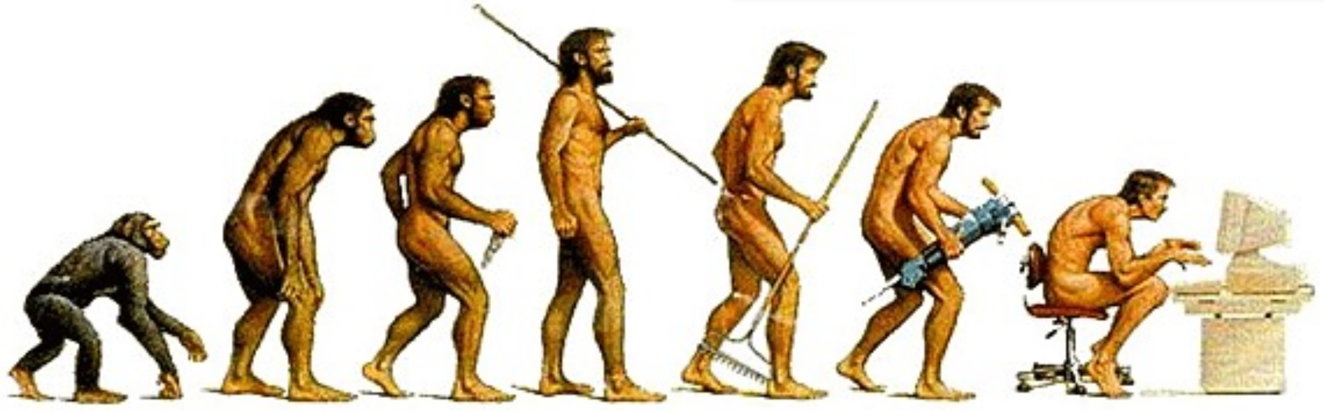
Programming with Java Structured Concurrency

- Java structured concurrency is evolving



JEP 428: Structured Concurrency (Incubator)

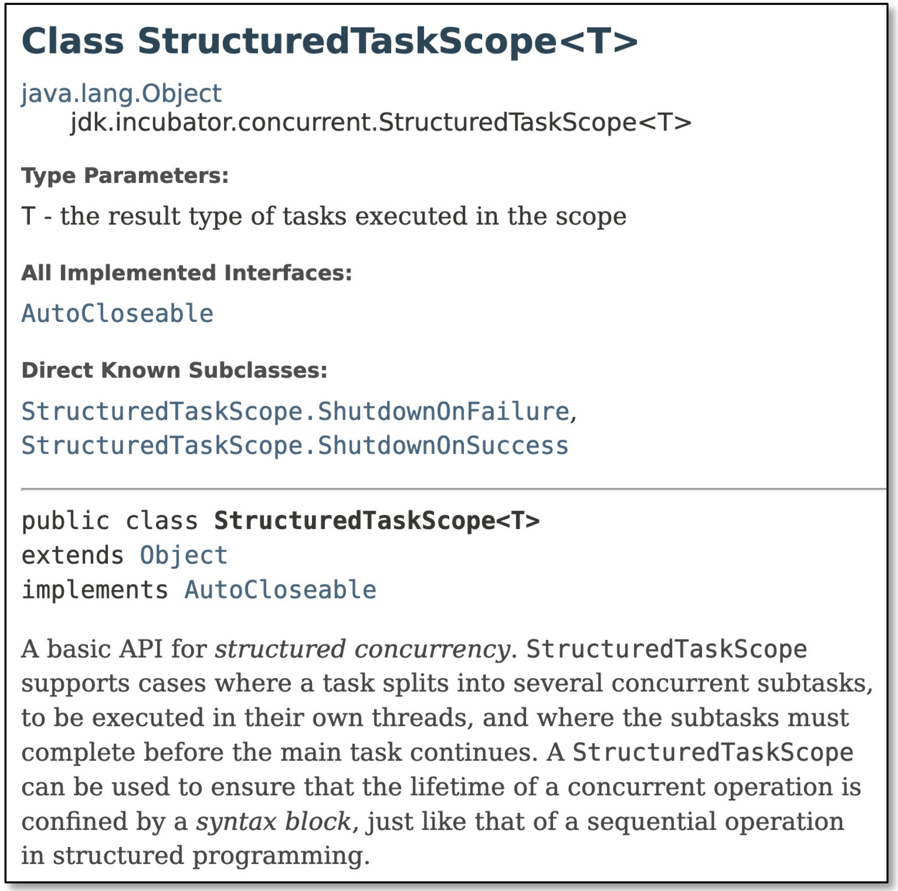
Authors	Alan Bateman, Ron Pressler
Owner	Alan Bateman
Type	Feature
Scope	JDK
Status	Closed / Delivered
Release	19
Component	core-libs
Discussion	loom dash dev at openjdk dot java dot net
Reviewed by	Alex Buckley, Brian Goetz
Created	2021/11/15 15:01
Updated	2022/08/10 15:58
Issue	8277129



See openjdk.org/jeps/428

Programming with Java Structured Concurrency

- Java structured concurrency is evolving
  - StructuredTaskScope



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 jdk/incubator/concurrent/StructuredTaskScope.html

Programming with Java Structured Concurrency

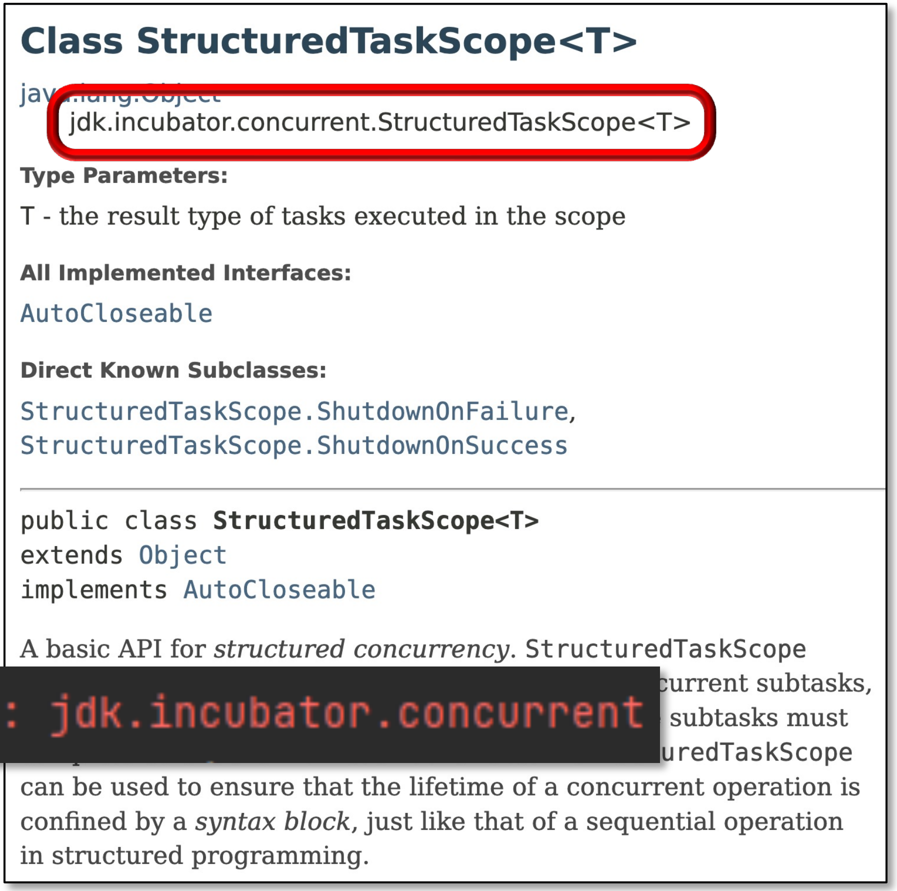
- Java structured concurrency is evolving
  - StructuredTaskScope
    - Splits a task into several concurrent subtasks within a syntax block

```
try (var scope = new
     StructuredTaskScope
     .ShutdownOnFailure()) {
  var downloadedImages = ...;

  for (URL url : urlList)
    downloadededImages
      .add(scope
      .fork(() ->
          downloadImage(url)));

  scope.join();

  return downloadedImages;
}
```

See openjdk.org/jeps/11

Programming with Java Structured Concurrency

- Java structured concurrency is evolving
  - StructuredTaskScope
    - Splits a task into several concurrent subtasks within a syntax block
    - Added in Java 19 as an "incubator feature"
      - Incubator features may iterate several times to get feedback & either be finalized or removed



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:

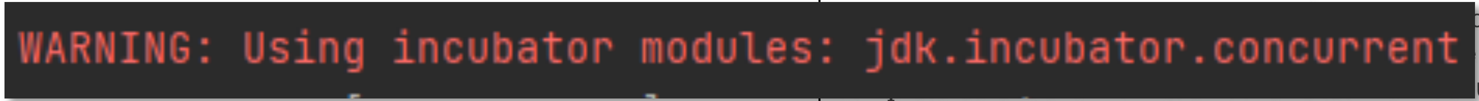
`AutoCloseable`

Direct Known Subclasses:

`StructuredTaskScope.ShutdownOnFailure`, `StructuredTaskScope.ShutdownOnSuccess`

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

A basic API for *structured concurrency*. `StructuredTaskScope` … current subtasks, … subtasks must … uredTaskScope 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.

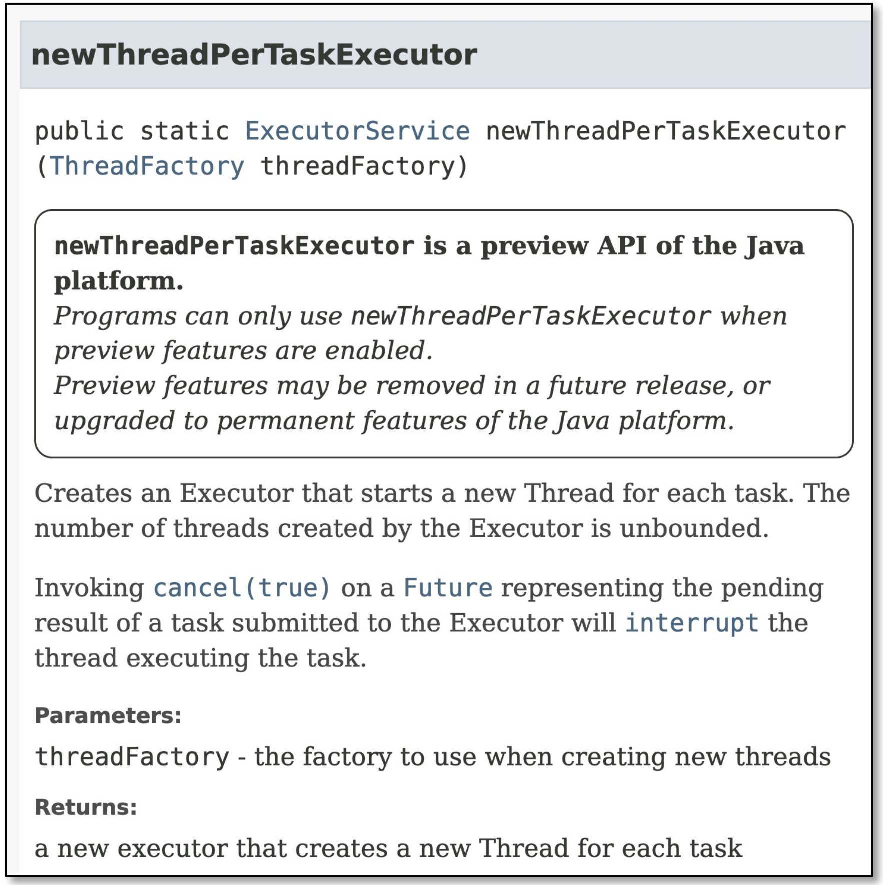


```
WARNING: Using incubator modules: jdk.incubator.concurrent
```

See openjdk.org/jeps/11

Programming with Java Structured Concurrency

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



newThreadPerTaskExecutor

```
public static ExecutorService newThreadPerTaskExecutor
(ThreadFactory threadFactory)
```

newThreadPerTaskExecutor is a preview API of the Java platform.
Programs can only use newThreadPerTaskExecutor 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 Thread for each task. The number of threads created by the Executor is unbounded.

Invoking `cancel(true)` on a `Future` representing the pending result of a task submitted to the Executor will `interrupt` the thread executing the task.

Parameters:

`threadFactory` - the factory to use when creating new threads

Returns:

a new executor that creates a new Thread for each task

See java/util/concurrent/Executors.html#newThreadPerTaskExecutor

Programming with Java Structured Concurrency

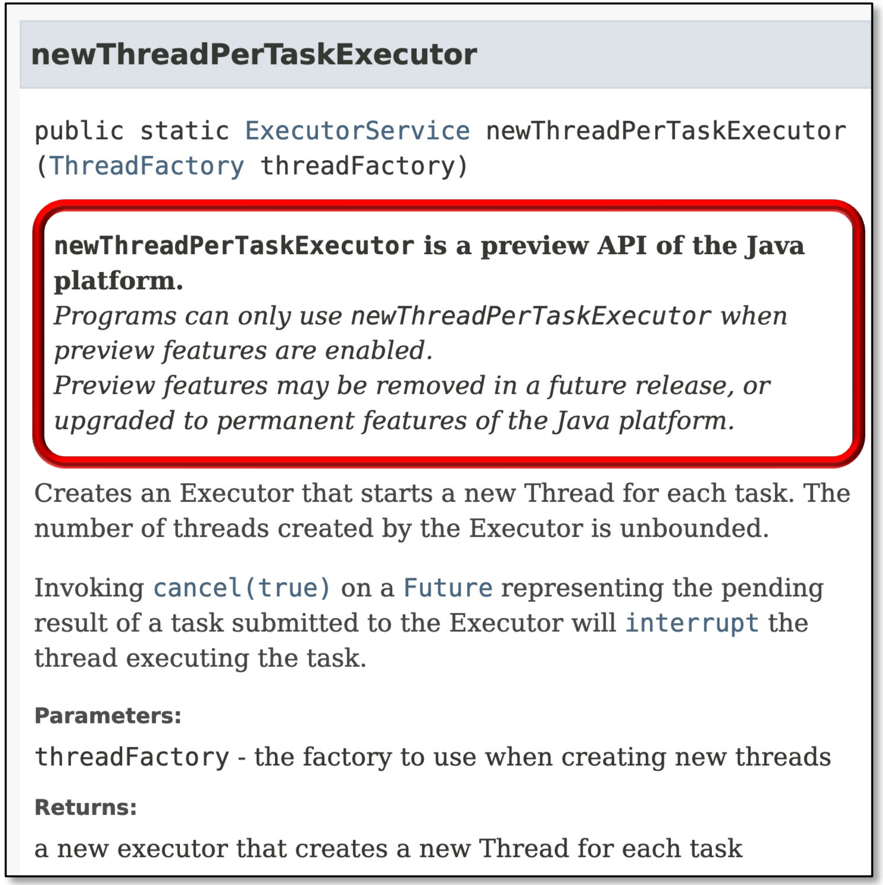
- Java structured concurrency is evolving
  - StructuredTaskScope
  - Executors/ExecutorService
    - Starts a new (virtual) Thread for each task within a syntax block

```
try (var executor = Executors
.newVirtualThreadPerTaskExecutor())
{
    IntStream
      .range(0, 10_000_000)
      .forEach(i -> executor
         .submit(() -> {
            Thread
              .sleep(Duration
               .ofSeconds(1));
            return i;
          }));
}
```

See docs.oracle.com/en/java/javase/19/language/preview-language-and-vm-features.html

Programming with Java Structured Concurrency

- Java structured concurrency is evolving
  - StructuredTaskScope
  - Executors/ExecutorService
    - Starts a new (virtual) Thread for each task within a syntax block
    - Added in Java 19 as a "preview feature"
      - Preview features are mostly finished, but are waiting for a round of feedback



newThreadPerTaskExecutor

```
public static ExecutorService newThreadPerTaskExecutor
(ThreadFactory threadFactory)
```

newThreadPerTaskExecutor is a preview API of the Java platform.
Programs can only use newThreadPerTaskExecutor 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 Thread for each task. The number of threads created by the Executor is unbounded.

Invoking `cancel(true)` on a `Future` representing the pending result of a task submitted to the Executor will `interrupt` the thread executing the task.

Parameters:

`threadFactory` - the factory to use when creating new threads

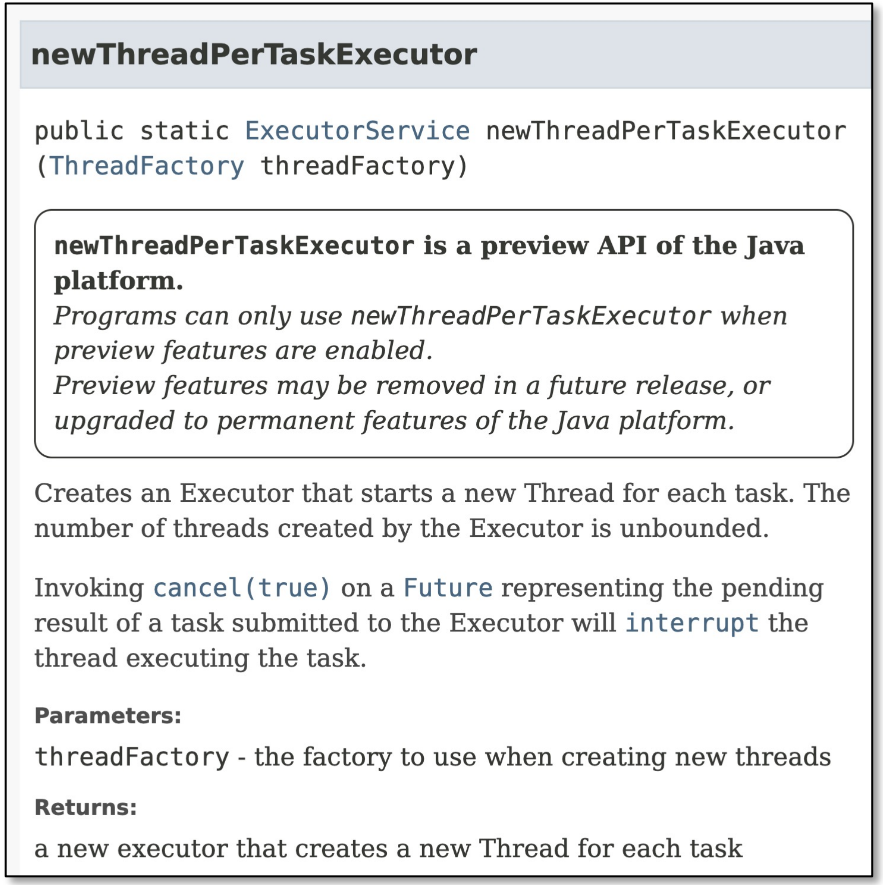
Returns:

a new executor that creates a new Thread for each task

See docs.oracle.com/en/java/javase/19/language/preview-language-and-vm-features.html

Programming with Java Structured Concurrency

- Java structured concurrency is evolving
  - StructuredTaskScope
  - Executors/ExecutorService
    - Starts a new (virtual) Thread for each task within a syntax block
    - Added in Java 19 as a "preview feature"
    - Less publicized as Structured TaskScope since it's limited



newThreadPerTaskExecutor

```
public static ExecutorService newThreadPerTaskExecutor
(ThreadFactory threadFactory)
```

newThreadPerTaskExecutor is a preview API of the Java platform.
Programs can only use newThreadPerTaskExecutor 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 Thread for each task. The number of threads created by the Executor is unbounded.

Invoking `cancel(true)` on a `Future` representing the pending result of a task submitted to the Executor will `interrupt` the thread executing the task.

Parameters:

`threadFactory` - the factory to use when creating new threads

Returns:

a new executor that creates a new Thread for each task

See upcoming lesson on "*Programming with Java ThreadPerTaskExecutor*"

End of Programming with Java Structured Concurrency