**Module** jdk.incubator.concurrent
**Package** jdk.incubator.concurrent

# Class StructuredTaskScope.ShutdownOnSuccess<T>

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

**Type Parameters:**

T - the result type

**All Implemented Interfaces:**

AutoCloseable

**Enclosing class:**

StructuredTaskScope<T>

---

```
public static final class StructuredTaskScope.ShutdownOnSuccess<T>
extends StructuredTaskScope<T>
```

A StructuredTaskScope that captures the result of the first subtask to complete successfully. Once captured, it invokes the shutdown method to interrupt unfinished threads and wakeup the owner. The policy implemented by this class is intended for cases where the result of any subtask will do ("invoke any") and where the results of other unfinished subtask are no longer needed.

Unless otherwise specified, passing a null argument to a method in this class will cause a NullPointerException to be thrown.

**Since:**

19

## Nested Class Summary

### Nested classes/interfaces declared in class jdk.incubator.concurrent.StructuredTaskScope

StructuredTaskScope.ShutdownOnFailure, StructuredTaskScope.ShutdownOnSuccess<T>

## Constructor Summary

**Constructors**

| Constructor | Description |
|---|---|
| ShutdownOnSuccess() | Constructs a new unnamed ShutdownOnSuccess that creates virtual threads. |
| ShutdownOnSuccess(String name, ThreadFactory factory) | Constructs a new ShutdownOnSuccess with the given name and thread factory. |

## Method Summary

**All Methods** | **Instance Methods** | **Concrete Methods**

| Modifier and Type | Method | Description |
|---|---|---|
| protected void | handleComplete(Future<T> future) | Shut down the given task scope when invoked for the first time with a Future for a task that completed with a result. |
| StructuredTaskScope.ShutdownOnSu | join() | Wait for all threads to finish or the task scope to shut down. |
| StructuredTaskScope.ShutdownOnSu | joinUntil(Instant deadline) | Wait for all threads to finish or the task scope to shut down, up to the given deadline. |
| T | result() | Returns the result of the first subtask that completed with a result. |
| <X extends Throwable> T | result(Function<Throwable,? extends X> esf) | Returns the result of the first subtask that completed with a result, otherwise throws an exception produced by the given exception supplying function. |

### Methods declared in class jdk.incubator.concurrent.StructuredTaskScope

close, fork, shutdown

### Methods declared in class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

## *Constructor Details*

### ShutdownOnSuccess

```
public ShutdownOnSuccess(String name,
                         ThreadFactory factory)
```

Constructs a new `ShutdownOnSuccess` with the given name and thread factory. The task scope is optionally named for the purposes of monitoring and management. The thread factory is used to `create` threads when tasks are forked. The task scope is owned by the current thread.

**Parameters:**

`name` - the name of the task scope, can be null

`factory` - the thread factory

### ShutdownOnSuccess

```
public ShutdownOnSuccess()
```

Constructs a new unnamed `ShutdownOnSuccess` that creates virtual threads.

This constructor is equivalent to invoking the 2-arg constructor with a name of `null` and a thread factory that creates virtual threads.

## *Method Details*

### handleComplete

```
protected void handleComplete(Future<T> future)
```

Shut down the given task scope when invoked for the first time with a `Future` for a task that completed with a result.

**Overrides:**

`handleComplete` in class `StructuredTaskScope<T>`

**Parameters:**

`future` - the completed task

**See Also:**

`StructuredTaskScope.shutdown()`, `Future.State.SUCCESS`

### join

```
public StructuredTaskScope.ShutdownOnSuccess<T> join()
                                            throws InterruptedException
```

Wait for all threads to finish or the task scope to shut down. This method waits until all threads started in the task scope finish execution (of both task and `handleComplete` method), the `shutdown` method is invoked to shut down the task scope, or the current thread is interrupted.

This method may only be invoked by the task scope owner.

**Overrides:**

`join` in class `StructuredTaskScope<T>`

**Returns:**

this task scope

**Throws:**

`IllegalStateException` - if this task scope is closed

`WrongThreadException` - if the current thread is not the owner

`InterruptedException` - if interrupted while waiting

## joinUntil

```
public StructuredTaskScope.ShutdownOnSuccess<T> joinUntil(Instant deadline)
                                                    throws InterruptedException,
                                                           TimeoutException
```

Wait for all threads to finish or the task scope to shut down, up to the given deadline. This method waits until all threads started in the task scope finish execution (of both task and `handleComplete` method), the `shutdown` method is invoked to shut down the task scope, the current thread is interrupted, or the deadline is reached.

This method may only be invoked by the task scope owner.

**Overrides:**

`joinUntil` in class `StructuredTaskScope<T>`

**Parameters:**

`deadline` - the deadline

**Returns:**

this task scope

**Throws:**

`IllegalStateException` - if this task scope is closed

`WrongThreadException` - if the current thread is not the owner

`InterruptedException` - if interrupted while waiting

`TimeoutException` - if the deadline is reached while waiting

## result

```
public T result()
         throws ExecutionException
```

Returns the result of the first subtask that completed with a result.

When no subtask completed with a result but a task completed with an exception then `ExecutionException` is thrown with the exception as the cause. If only cancelled subtasks were notified to the `handleComplete` method then `CancellationException` is thrown.

**API Note:**

This method is intended to be invoked by the task scope owner after it has invoked `join` (or `joinUntil`). A future release may add enforcement to prevent the method being called by other threads or before joining.

**Returns:**

the result of the first subtask that completed with a result

**Throws:**

`ExecutionException` - if no subtasks completed with a result but a subtask completed with an exception

`CancellationException` - if all subtasks were cancelled

`IllegalStateException` - if the handle method was not invoked with a completed subtask

## result

```
public <X extends Throwable> T result(Function<Throwable,? extends X> esf)
                             throws X
```

Returns the result of the first subtask that completed with a result, otherwise throws an exception produced by the given exception supplying function.

When no subtask completed with a result but a subtask completed with an exception then the exception supplying function is invoked with the exception. If only cancelled subtasks were notified to the `handleComplete` method then the exception supplying function is invoked with a `CancellationException`.

**API Note:**

This method is intended to be invoked by the task scope owner after it has invoked `join` (or `joinUntil`). A future release may add enforcement to prevent the method being called by other threads or before joining.

**Type Parameters:**

`X` - type of the exception to be thrown

**Parameters:**

`esf` - the exception supplying function

**Returns:**

the result of the first subtask that completed with a result

**Throws:**

`X` - if no subtask completed with a result

`IllegalStateException` - if the handle method was not invoked with a completed subtask

---

Report a bug or suggest an enhancement
For further API reference and developer documentation see the Java SE Documentation, which contains more detailed, developer-targeted descriptions with conceptual overviews, definitions of terms, workarounds, and working code examples. Other versions.
Java is a trademark or registered trademark of Oracle and/or its affiliates in the US and other countries.
Copyright © 1993, 2022, Oracle and/or its affiliates, 500 Oracle Parkway, Redwood Shores, CA 94065 USA.
All rights reserved. Use is subject to license terms and the documentation redistribution policy.