Module java.base

Package java.util.concurrent

Since: **1.7**

Class Phaser

java.util.concurrent.Phaser

A Phaser provides a more flexible form of barrier that may be used to control phased computation among multiple threads.

Phaser = CountdownLatch + CyclicBarrier.

What is a Phaser?

Answer: - Phaser in Java is also one of the synchronization aids provided in concurrency util.

Phaser is similar to other synchronization barrier utils like **CountDownLatch** and **CyclicBarrier**.

What sets Phaser apart is it is reusable (like CyclicBarrier) and more flexible in usage.

In both **CountDownLatch** and **CyclicBarrier** number of parties (thread) that are registered for waiting **can't change** whereas in Phaser that number can vary.

Also note that Phaser has been introduced in **Java 7**.

Phaser in Java is more suitable for use where it is required to synchronize threads over one or more phases of activity. Though Phaser can be used to synchronize a single phase, in that case it acts more like a CyclicBarrier. But it is more suited where threads should wait for a phase to finish, then advance to next phase, wait again for that phase to finish and so on.

Phaser Features

- **Phaser is more flexible-** Unlike the case for other barriers, the number of parties registered to synchronize on a Phaser may vary over time. **Tasks may be registered at any time** (using methods register (), bulkRegister(int), or by specifying initial number of parties in the constructor). **Tasks may also be optionally deregistered upon any arrival** (using arriveAndDeregister()).
- **Phaser termination-** A Phaser may enter a termination state, that may be checked using method **isTerminated**(). Upon termination, all synchronization methods immediately return without waiting for advance, as indicated by a negative return value. Similarly, attempts to register upon termination have no effect.
- **Phaser Tiering-** Phasers in Java may be tiered (i.e., constructed in tree structures) to reduce contention. Phasers with large numbers of parties may experience heavy synchronization contention costs. These may be set up as a group of sub-phasers which share a common parent. This may greatly increase throughput even though it incurs greater per-operation overhead.

Overriding onAdvance() method in Phaser

If you want to perform an action before advancing from one phase to another, it can be done by overriding the **onAdvance**() method of the Phaser class. This method is invoked when the Phaser advances from one phase to another.

If this method returns true, this phaser will be set to a final termination state upon advance, and subsequent calls to isTerminated() will return true.

If this method returns false, phaser will be kept alive.

How Phaser in Java works

- First thing is to create a new instance of Phaser.
- Next thing is to register one or more parties with the Phaser. That can be done using register (), bulkRegister(int) or by specifying number of parties in the constructor.
- Now since Phaser is a synchronization barrier so we have to make phaser wait until all registered parties finish a phase. That waiting can be done using arrive () or any of the variants of arrive () method. When the number of arrivals is

equal to the parties which are registered, that phase is considered completed and it advances to next phase (if there is any), or terminate.

- Note that each generation of a phaser has an associated phase number. The phase number starts at zero, and advances when all parties arrive at the phaser, wrapping around to zero after reaching Integer.MAX_VALUE.

How parties register?

In a Phaser task may be registered at any time and can optionally be deregistered upon any arrival. Like a **CyclicBarrier** tasks may be **repeatedly awaited** and method **arriveAndAwaitAdvance** has similar effect as **await** method of **CyclicBarrier**. Each generation of a phaser is represented by a phase number which starts from zero to Integer.MAX_VALUE and then again is wrapped to zero.

How synchronization works?

The methods **arrive** and **arriveAndDeregister** record arrival and return the arrival phase number (the phase number to which this arrival is applicable) without blocking. When the final party (thread) for a particular phase arrives an action (optional action) may be performed and phase is advanced (incremented).

The method awaitAdvance(int phaseNumber) takes arrival phase number and it returns when the phaser advances to (or already is at) a different phase.

How can a phaser terminate?

A phaser terminates when the method onAdvance returns true. In default implementation of this method, it returns true when all the partied have deregistered and number of registered parties becomes zero. We can check whether a phaser has terminated or not by calling method isTerminated on phaser instance.

CyclicBarrier vs CountdownLatch vs Phaser

1. CountDownLatch:

- Created with a fixed number of threads
- Cannot be reset
- Allows threads to wait(CountDownLatch#await()) or continue with its execution(CountDownLatch#countDown()).

2. CyclicBarrier:

- Can be reset.
- Does not a provide a method for the threads to advance. The threads have to wait till all the threads arrive.
- Created with fixed number of threads.

3. Phaser:

- Number of threads need not be known at Phaser creation time. They can be added dynamically.
- Can be reset and hence is, reusable.
- Allows threads to wait(Phaser#arriveAndAwaitAdvance()) or continue with its execution(Phaser#arrive()).
- Supports multiple Phases.

Summary: -

CountdownLatch	CyclicBarrier	Phaser
Fixed number of parties	Fixed number of parties	Dynamic number of parties
Non-cyclic in nature hence not reusable	cyclic in nature hence reusable	Reusable
Can be advanced using countDown (advance) and await (must wait)	Cannot be advanced	Can be advanced using relevant methods.

Methods: -

- **int arrive ()** Arrives at this phaser, without waiting for others to arrive.
- **int arriveAndAwaitAdvance()** Arrives at this phaser and awaits others.
- **int arriveAndDeregister()** Arrives at this phaser and deregisters from it without waiting for others to arrive.
- **int awaitAdvance (int phase)** Awaits the phase of this phaser to advance from the given phase value, returning immediately if the current phase is not equal to the given phase value or this phaser is terminated.
- **int awaitAdvanceInterruptibly (int phase)** Awaits the phase of this phaser to advance from the given phase value, throwing InterruptedException if

interrupted while waiting, or returning immediately if the current phase is not equal to the given phase value or this phaser is terminated.

- **int awaitAdvanceInterruptibly (int phase, long timeout, TimeUnit unit)** Awaits the phase of this phaser to advance from the given phase value or the given timeout to elapse, throwing **InterruptedException** if interrupted while waiting, or returning immediately if the current phase is not equal to the given phase value or this phaser is terminated.
- **int bulkRegister (int parties)** Adds the given number of new unarrived parties to this phaser.
- **void forceTermination()** Forces this phaser to enter termination state.
- **int getArrivedParties()** Returns the number of registered parties that have arrived at the current phase of this phaser.
- **Phaser getParent()** Returns the parent of this phaser, or null if none.
- **int getPhase()** Returns the current phase number.
- **int getRegisteredParties()** Returns the number of parties registered at this phaser.
- **Phaser getRoot()** Returns the root ancestor of this phaser, which is the same as this phaser if it has no parent.
- **int getUnarrivedParties()** Returns the number of registered parties that have not yet arrived at the current phase of this phaser.
- **boolean isTerminated()** Returns true if this phaser has been terminated.
- **protected boolean onAdvance (int phase, int registeredParties)** Overridable method to perform an action upon impending phase advance, and to control termination.
- **int register ()** Adds a new unarrived party to this phaser.
- **String toString()** Returns a string identifying this phaser, as well as its state.

Register

public int register ()

Adds a new unarrived party to this phaser. If an ongoing invocation of onAdvance(int, int) is in progress, this method may await its completion before returning. If this phaser has a parent, and this phaser previously had no registered parties, this child phaser is also registered with its parent. If this phaser is terminated, the attempt to register has no effect, and a negative value is returned.

Returns: the arrival phase number to which this registration applied. If this value is negative, then this phaser has terminated, in which case registration has no effect.

Throws: IllegalStateException - if attempting to register more than the maximum supported number of parties

BulkRegister

public int bulkRegister (int parties)

Adds the given number of new unarrived parties to this phaser. If an ongoing invocation of onAdvance(int, int) is in progress, this method may await its completion before returning. If this phaser has a parent, and the given number of parties is greater than zero, and this phaser previously had no registered parties, this child phaser is also registered with its parent. If this phaser is terminated, the attempt to register has no effect, and a negative value is returned.

Parameters: parties - the number of additional parties required to advance to the next phase

Returns: the arrival phase number to which this registration applied. If this value is negative, then this phaser has terminated, in which case registration has no effect.

Throws: IllegalStateException - if attempting to register more than the maximum supported number of parties

IllegalArgumentException - if parties < 0

Arrive

public int arrive ()

Arrives at this phaser, without waiting for others to arrive.

It is a usage error for an unregistered party to invoke this method. However, this error may result in an IllegalStateException only upon some subsequent operation on this phaser, if ever.

Returns: the arrival phase number, or a negative value if terminated

Throws: IllegalStateException - if not terminated and the number of unarrived parties would become negative

ArriveAndDeregister

public int arriveAndDeregister()

Arrives at this phaser and deregisters from it without waiting for others to arrive. Deregistration reduces the number of parties required to advance in future phases. If this phaser has a parent, and deregistration causes this phaser to have zero parties, this phaser is also deregistered from its parent.

It is a usage error for an unregistered party to invoke this method. However, this error may result in an IllegalStateException only upon some subsequent operation on this phaser, if ever.

Returns: the arrival phase number, or a negative value if terminated

Throws: IllegalStateException - if not terminated and the number of registered or unarrived parties would become negative

ArriveAndAwaitAdvance

public int arriveAndAwaitAdvance()

Arrives at this phaser and awaits others. Equivalent in effect to awaitAdvance(arrive()). If you need to await with interruption or timeout, you can arrange this with an analogous construction using one of the other forms of the awaitAdvance method. If instead you need to deregister upon arrival, use awaitAdvance(arriveAndDeregister()).

It is a usage error for an unregistered party to invoke this method. However, this error may result in an IllegalStateException only upon some subsequent operation on this phaser, if ever.

Returns: the arrival phase number, or the (negative) current phase if terminated

Throws: IllegalStateException - if not terminated and the number of unarrived parties would become negative

AwaitAdvance

public int awaitAdvance (int phase)

Awaits the phase of this phaser to advance from the given phase value, returning immediately if the current phase is not equal to the given phase value or this phaser is terminated.

Parameters: phase - an arrival phase number, or negative value if terminated; this argument is normally the value returned by a previous call to arrive or arriveAndDeregister.

Returns: the next arrival phase number, or the argument if it is negative, or the (negative) current phase if terminated

AwaitAdvanceInterruptibly

public int awaitAdvanceInterruptibly (int phase) throws InterruptedException

Awaits the phase of this phaser to advance from the given phase value, throwing InterruptedException if interrupted while waiting, or returning immediately if the current phase is not equal to the given phase value or this phaser is terminated.

Parameters: phase - an arrival phase number, or negative value if terminated; this argument is normally the value returned by a previous call to arrive or arriveAndDeregister.

Returns: the next arrival phase number, or the argument if it is negative, or the (negative) current phase if terminated

Throws: InterruptedException - if thread interrupted while waiting

awaitAdvanceInterruptibly

public int awaitAdvanceInterruptibly (int phase, long timeout, TimeUnit unit) throws InterruptedException, TimeoutException

Awaits the phase of this phaser to advance from the given phase value or the given timeout to elapse, throwing InterruptedException if interrupted while waiting, or returning immediately if the current phase is not equal to the given phase value or this phaser is terminated.

Parameters: phase - an arrival phase number, or negative value if terminated; this argument is normally the value returned by a previous call to arrive or arriveAndDeregister.

timeout - how long to wait before giving up, in units of unit

unit - a TimeUnit determining how to interpret the timeout parameter

Returns: the next arrival phase number, or the argument if it is negative, or the (negative) current phase if terminated

Throws: InterruptedException - if thread interrupted while waiting

TimeoutException - if timed out while waiting