Module java.base

Package java.util.concurrent

Since: 1.5

Class CyclicBarrier

Java.util.concurrent.CyclicBarrier

public class CyclicBarrier extends Object

A CyclicBarrier is a resettable multiway synchronization point useful in some styles of parallel programming.

A situation not uncommon in concurrent programming occurs when a set of two or more threads must wait at a predetermined execution point until all threads in the set have reached that point. To handle such a situation, the concurrent API supplies the **CyclicBarrier** class. **It enables you to define a synchronization object that suspends until the specified number of threads has reached the barrier point.**

The barrier is called cyclic because it can be re-used after the waiting threads are released.

What is CyclicBarrier in Java?

Answer: - CyclicBarrier class is a synchronization mechanism that can synchronize threads progressing through some algorithm. In other words, it is a barrier that all threads must wait at, until all threads reach it, before any of the threads can continue.

The threads wait for each other by calling the **await ()** method on the CyclicBarrier. Once N threads are waiting at the CyclicBarrier, all threads are released and can continue running.

All threads which wait for each other to reach the barrier are called **parties**, CyclicBarrier is **initialized with a number of parties to wait** and threads wait for each other by calling **CyclicBarrier.await()** method which is a blocking method in Java and blocks until all Thread or parties call **await ()**.

Note: - CyclicBarrier is a natural requirement for a concurrent program because it can be used to perform the final part of the task once individual tasks are completed.

When to use CyclicBarrier in Java?

Answer: - You can use CyclicBarrier in Java: -

- To implement multiplayer game which cannot begin until all player has joined.
- Perform lengthy calculation by breaking it into smaller individual tasks, in general, to implement Map reduce technique.

For example, to count the population of India you can have 4 threads which count population from North, South, East, and West and once complete they can wait for each other When last thread completed their task, Main thread or any other thread can add result from each zone and print total population.

Important points of CyclicBarrier in Java

- CyclicBarrier can perform a completion task once all thread reaches to the barrier
- If CyclicBarrier is initialized with 3 parties means 3 thread needs to call the await method to break the barrier.
- The thread will block on await () until all parties reach the barrier, another thread interrupts or await timed out.
- If another thread interrupts the thread which is waiting on the barrier it will throw **BrokernBarrierException**
- **CyclicBarrier.reset()** put Barrier on its initial state, other thread which is waiting or not yet reached barrier will terminate with **java.util.concurrent.BrokenBarrierException.**

Constructor

1. **CyclicBarrier (int parties)** Creates a new CyclicBarrier that will trip when the given number of parties (threads) are waiting upon it, and does not perform a predefined action when the barrier is tripped.
2. **CyclicBarrier (int parties, Runnable barrierAction)** Creates a new CyclicBarrier that will trip when the given number of parties (threads) are waiting upon it, and which will execute the given barrier action when the barrier is tripped, performed by the last thread entering the barrier.

Methods

- **int await ()** Waits until all parties have invoked await on this barrier.
- **int await (long timeout, TimeUnit unit)** Waits until all parties have invoked await on this barrier, or the specified waiting time elapses.
- **int getNumberWaiting()** Returns the number of parties currently waiting at the barrier.
- **int getParties()** Returns the number of parties required to trip this barrier.
- **boolean isBroken()** Queries if this barrier is in a broken state.
- **void reset()** Resets the barrier to its initial state.

What is Difference between CountDownLatch and CyclicBarrier?

Answer: - There are following difference between CountDownLatch and CyclicBarrier: -

- If you look at **CyclicBarrier** it also does the same thing as **CountDownLatch** but it is different you cannot reuse **CountDownLatch** once the count reaches zero while you can reuse CyclicBarrier by calling the reset () method which resets Barrier to its initial State. What it implies that **CountDownLatch** is good for one-time events like application start-up time and **CyclicBarrier** can be used in case of the recurrent event like concurrently calculating a solution of the big problem etc.
- **CyclicBarrier** allows a number of threads to wait on each other, whereas **CountDownLatch** allows one or more threads to wait for a number of tasks to complete. In short, **CyclicBarrier** maintains a count of threads whereas **CountDownLatch** maintains a count of tasks.

- One major difference is that **CyclicBarrier** takes an **(optional) Runnable task** which is run once the common barrier condition is met.
- It also allows you to get the number of clients waiting at the barrier and the number required to trigger the barrier. Once triggered the barrier is reset and can be used again.

What is Difference between CyclicBarrier (int parties) and CyclicBarrier (int parties, Runnable barrierAction)?

Answer: -

CyclicBarrier (int parties)	**CyclicBarrier (int parties, Runnable barrierAction)**
Creates a new CyclicBarrier that will trip when the given number of parties (threads) are waiting upon it, and does not perform a predefined action when the barrier is tripped.	Creates a new CyclicBarrier that will trip when the given number of parties (threads) are waiting upon it, and which will execute the given barrier action when the barrier is tripped, performed by the last thread entering the barrier.

What is Difference between await () and await (long timeout,TimeUnit unit)?

Answer: -

await () throws InterruptedException, BrokenBarrierException	**await (long timeout,TimeUnit unit) throws InterruptedException,BrokenBarrierException,TimeoutException**
Waits until all parties have invoked await on this barrier.	Waits until all parties have invoked await on this barrier, or the specified waiting time elapses.
If the current thread is not the last to arrive then it is disabled for thread scheduling purposes and lies dormant until one of the following things happens: • The last thread arrives; or	If the current thread is not the last to arrive then it is disabled for thread scheduling purposes and lies dormant until one of the following things happens: • The last thread arrives; or • The specified timeout elapses; or

• Some other thread interrupts the current thread; or • Some other thread interrupts one of the other waiting threads; or • Some other thread times out while waiting for barrier; or • Some other thread invokes reset () on this barrier.	• Some other thread interrupts the current thread; or • Some other thread interrupts one of the other waiting threads; or • Some other thread times out while waiting for barrier; or • Some other thread invokes reset () on this barrier.
If the current thread: • has its interrupted status set on entry to this method; or • is interrupted while waiting then InterruptedException is thrown and the current thread's interrupted status is cleared. If the barrier is reset () while any thread is waiting, or if the barrier is broken when await is invoked, or while any thread is waiting, then BrokenBarrierException is thrown. If any thread is interrupted while waiting, then all other waiting threads will throw BrokenBarrierException and the barrier is placed in the broken state. If the current thread is the last thread to arrive, and a non-null barrier action was supplied in the constructor, then the current thread runs the action before allowing the other threads to continue. If an exception occurs during the barrier action, then that exception will be propagated in the current thread and the barrier is placed in the broken state.	If the current thread: • has its interrupted status set on entry to this method; or • is interrupted while waiting then InterruptedException is thrown and the current thread's interrupted status is cleared. If the specified waiting time elapses, then TimeoutException is thrown. If the time is less than or equal to zero, the method will not wait at all. If the barrier is reset () while any thread is waiting, or if the barrier is broken when await is invoked, or while any thread is waiting, then BrokenBarrierException is thrown. If any thread is interrupted while waiting, then all other waiting threads will throw BrokenBarrierException and the barrier is placed in the broken state. If the current thread is the last thread to arrive, and a non-null barrier action was supplied in the constructor, then the current thread runs the action before allowing the other threads to continue. If an exception occurs during the barrier action then that exception will be

	propagated in the current thread and the barrier is placed in the broken state.