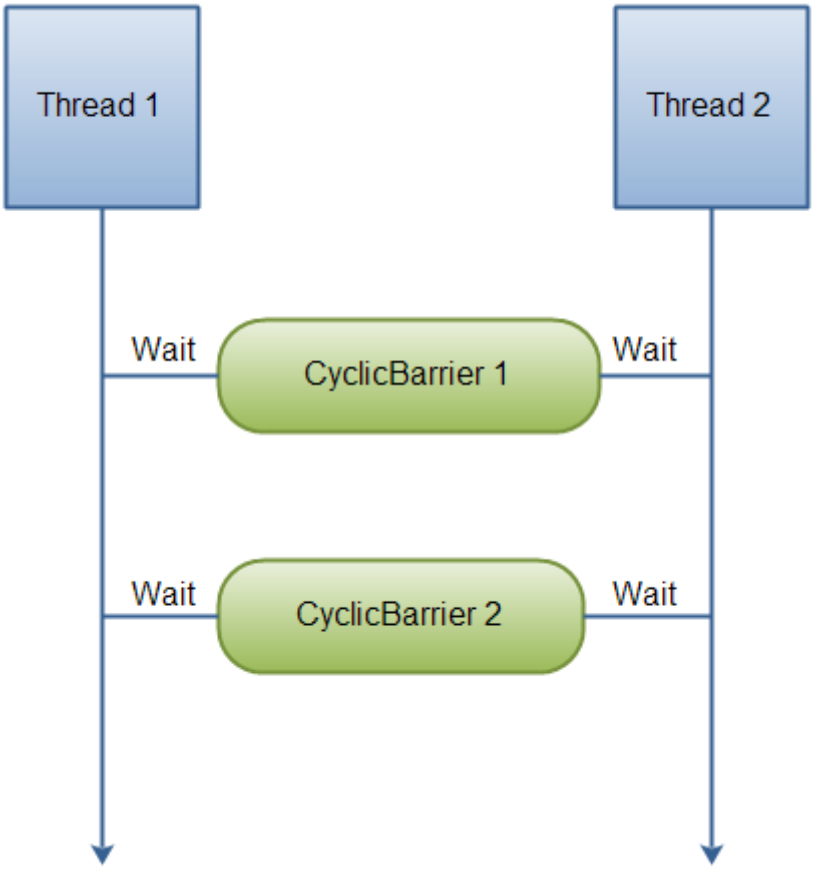
The `java.util.concurrent.CyclicBarrier` class is a synchronization mechanism that can synchronize threads progressing through some algorithm. In other words, it is a barrier that all threads must wait at, until all threads reach it, before any of the threads can continue. Here is a diagram illustrating that:



Two threads waiting for each other at CyclicBarriers.

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.

Creating a CyclicBarrier

When you create a `CyclicBarrier` you specify how many threads are to wait at it, before releasing them. Here is how you create a `CyclicBarrier`:

```
CyclicBarrier barrier = new CyclicBarrier(2);
```

Waiting at a CyclicBarrier

Here is how a thread waits at a `CyclicBarrier`:

```
barrier.await();
```

You can also specify a timeout for the waiting thread. When the timeout has passed the thread is also released, even if not all N threads are waiting at the `CyclicBarrier`. Here is how you specify a timeout:

```
barrier.await(10, TimeUnit.SECONDS);
```

The waiting threads waits at the `CyclicBarrier` until either:

- The last thread arrives (calls await())
- The thread is interrupted by another thread (another thread calls its interrupt() method)
- Another waiting thread is interrupted
- Another waiting thread times out while waiting at the `CyclicBarrier`
- The `CyclicBarrier.reset()` method is called by some external thread.

CyclicBarrier Action

The `CyclicBarrier` supports a barrier action, which is a `Runnable` that is executed once the last thread arrives. You pass the `Runnable` barrier action to the `CyclicBarrier` in its constructor, like this:

```
Runnable      barrierAction = ... ;
CyclicBarrier barrier       = new CyclicBarrier(2, barrierAction);
```

CyclicBarrier Example

Here is a code example that shows you how to use a `CyclicBarrier`:

```
Runnable barrier1Action = new Runnable() {
    public void run() {
        System.out.println("BarrierAction 1 executed ");
    }
};
Runnable barrier2Action = new Runnable() {
    public void run() {
        System.out.println("BarrierAction 2 executed ");
    }
};

CyclicBarrier barrier1 = new CyclicBarrier(2, barrier1Action);
CyclicBarrier barrier2 = new CyclicBarrier(2, barrier2Action);

CyclicBarrierRunnable barrierRunnable1 =
        new CyclicBarrierRunnable(barrier1, barrier2);

CyclicBarrierRunnable barrierRunnable2 =
        new CyclicBarrierRunnable(barrier1, barrier2);

new Thread(barrierRunnable1).start();
new Thread(barrierRunnable2).start();
```

Here is the `CyclicBarrierRunnable` class:

```
public class CyclicBarrierRunnable implements Runnable{

    CyclicBarrier barrier1 = null;
    CyclicBarrier barrier2 = null;

    public CyclicBarrierRunnable(
            CyclicBarrier barrier1,
            CyclicBarrier barrier2) {

        this.barrier1 = barrier1;
```

```
        this.barrier2 = barrier2;
    }

    public void run() {
        try {
            Thread.sleep(1000);
            System.out.println(Thread.currentThread().getName() +
                               " waiting at barrier 1");
            this.barrier1.await();

            Thread.sleep(1000);
            System.out.println(Thread.currentThread().getName() +
                               " waiting at barrier 2");
            this.barrier2.await();

            System.out.println(Thread.currentThread().getName() +
                               " done!");

        } catch (InterruptedException e) {
            e.printStackTrace();
        } catch (BrokenBarrierException e) {
            e.printStackTrace();
        }
    }
}
```

Here is the console output for an execution of the above code. Note that the sequence in which the threads gets to write to the console may vary from execution to execution. Sometimes `Thread-0` prints first, sometimes `Thread-1` prints first etc.

```
Thread-0 waiting at barrier 1
Thread-1 waiting at barrier 1
BarrierAction 1 executed
Thread-1 waiting at barrier 2
Thread-0 waiting at barrier 2
BarrierAction 2 executed
Thread-0 done!
Thread-1 done!
```