CountDownLatch is used to make sure that a task waits for other threads before it starts. To understand its application, let us consider a server where the main task can only start when all the required services have started.

**Working of CountDownLatch:**

When we create an object of CountDownLatch, we specify the number if threads it should wait for, all such thread are required to do count down by calling CountDownLatch.countDown() once they are completed or ready to the job. As soon as count reaches zero, the waiting task starts running.

Creating an object of CountDownLatch by passing an int to its constructor (the count), is actually number of invited parties (threads) for an event.

The thread, which is dependent on other threads to start processing, waits on until every other thread has called count down. All threads, which are waiting on await() proceed together once count down reaches to zero.

countDown() method decrements the count and await() method blocks until count == 0

import java.util.Random;

import java.util.concurrent.CountDownLatch;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

public class CountDownLatchExample implements Runnable {

private static final int NUMBER\_OF\_THREADS = 5;

private static final CountDownLatch latch = new CountDownLatch(NUMBER\_OF\_THREADS);

private static Random random = new Random(System.currentTimeMillis());

public static void main(String[] args) {

ExecutorService executorService = Executors.newFixedThreadPool(NUMBER\_OF\_THREADS);

for (int i = 0; i < NUMBER\_OF\_THREADS; i++) {

executorService.execute(new CountDownLatchExample());

}

executorService.shutdown();

}

public void run() {

try {

int randomSleepTime = random.nextInt(20000);

System.out.println("[" + Thread.currentThread().getName() + "] Sleeping for " + randomSleepTime);

Thread.sleep(randomSleepTime);

latch.countDown();

System.out.println("[" + Thread.currentThread().getName() + "] Waiting for latch.");

latch.await();

System.out.println("[" + Thread.currentThread().getName() + "] Finished.");

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

**Cyclic Barrier**

CyclicBarrier is used to make threads wait for each other. It is used when different threads process a part of computation and when all threads have completed the execution, the result needs to be combined in the parent thread. In other words, a CyclicBarrier is used when multiple thread carry out different sub tasks and the output of these sub tasks need to be combined to form the final output. After completing its execution, threads call await() method and wait for other threads to reach the barrier. Once all the threads have reached, the barriers then give the way for threads to proceed.

Difference between CountDownLatch and CyclicBarrier in Java which separates them apart and that is, you can not reuse same CountDownLatch instance once count reaches to zero and latch is open, on the other hand CyclicBarrier can be reused by resetting Barrier, Once barrier is broken.