CountDownLatch in Java

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 of 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.

Example of CountDownLatch in JAVA:

```
// Java Program to demonstrate how
// to use CountDownLatch, Its used
// when a thread needs to wait for other
// threads before starting its work.
import java.util.concurrent.CountDownLatch;

public class CountDownLatchDemo
{
    public static void main(String args[])
                throws InterruptedException
    {
        // Let us create task that is going to
        // wait for four threads before it starts
        CountDownLatch latch = new CountDownLatch(4);

        // Let us create four worker
        // threads and start them.
        Worker first = new Worker(1000, latch,
                                  "WORKER-1");
        Worker second = new Worker(2000, latch,
                                  "WORKER-2");
        Worker third = new Worker(3000, latch,
                                  "WORKER-3");
        Worker fourth = new Worker(4000, latch,
                                  "WORKER-4");
        first.start();
        second.start();
        third.start();
        fourth.start();

        // The main task waits for four threads
        latch.await();

        // Main thread has started
        System.out.println(Thread.currentThread().getName() +
                           " has finished");
    }
}

// A class to represent threads for which
// the main thread waits.
class Worker extends Thread
{
    private int delay;
    private CountDownLatch latch;

    public Worker(int delay, CountDownLatch latch,
                                    String name)
    {
        super(name);
        this.delay = delay;
        this.latch = latch;
    }

    @Override
    public void run()
    {
        try
        {
            Thread.sleep(delay);
            latch.countDown();
            System.out.println(Thread.currentThread().getName()
                               + " finished");
        }
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }
    }
}
```

Run on IDE

Output:

```
WORKER-1 finished
WORKER-2 finished
WORKER-3 finished
WORKER-4 finished
main has finished
```

Facts about CountDownLatch:

1. Creating an object of CountDownLatch by passing an int to its constructor (the count), is actually number of invited parties (threads) for an event.
2. 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.
3. countDown() method decrements the count and await() method blocks until count == 0