**CountDownLatch**

CountDownLatch is **a concurrency construct that allows one or more threads to wait for a given set of operations to complete**. A CountDownLatch is initialized with a given count. This count is decremented by calls to the countDown() method. Threads waiting for this count to reach zero can call one of the await() methods.

CountDownLatch is used to make a thread wait till other threads finish their execution.

1. A CountDownLatch is initialized with a count which is the number of child threads for which the main thread will wait before executing further.
2. It is an integer.
3. Thread which starts other threads and which wants to wait for these threads to finish calls await.
4. Every child thread when finishes execution, calls countDown method.
5. This reduces the count by 1.
6. When all the child threads have completed(and have called countDown), the value of count becomes 0 and main thread resumes its execution.

One of the disadvantages/advantages of CountDownLatch is that it's not reusable: once count reaches zero you cannot use CountDownLatch any more. **The await methods block until the current count reaches zero due to invocations of the countDown() method.**

Use CountDownLatch when one thread (like the main thread) requires to wait for one or more threads to complete, before it can continue processing.

we can make several threads start at the same time; we can wait for several threads to finish (whereas, for example, the Thread.join() method only lets you wait for a single thread).

**Why use CountDownLatch (rather than wait/notify, Condition etc)?**

The CountDownLatch protects you against the case of a thread missing a signal which can occur if you use these other mechanisms for coordinating jobs. Something like a Condition is useful for signaling to threads if they are waiting, but where it doesn't matter if they're not, or where a thread will explicitly check if it has to wait before waiting. With a CountDownLatch, we await a signal if it hasn't been triggered yet, but immediately continue without waiting if that signal was already triggered before we start waiting.

Problem: **A teacher gives a problem to 5 students to compute. Once all the students have completed the computations, finally teacher has to disclose the result of all students.** Code is given below.

**import** java.util.Map; **import** java.util.Random;  
**import** java.util.concurrent.CountDownLatch;  
**import** java.util.concurrent.TimeUnit;  
**public class StudentThread** **implements** Runnable {  
 **private** Map<String, Integer> **dataMap**;  
 **private** String **name**;  
 **private** CountDownLatch **countDownLatch**;  
 **private int timeInSecs**;  
  
 **public** StudentThread(  
 Map<String, Integer> dataMap, String name, CountDownLatch countDownLatch, **int** timeInSecs) {  
 **this**.**dataMap** = dataMap;  
 **this**.**name** = name;  
 **this**.**countDownLatch** = countDownLatch;  
 **this**.**timeInSecs** = timeInSecs;  
 }  
 @Override  
 **public void** run() {  
 **try** {  
 System.***out***.println(Thread.*currentThread*().getName() + **" computing value ..."**);  
 TimeUnit.***SECONDS***.sleep(**timeInSecs**);  
 Random rand = **new** Random();  
 **int** value = rand.nextInt(100);  
 **dataMap**.put(**name**, value);  
 } **catch** (Exception e) { e.printStackTrace(); }  
 System.***out***.println(Thread.*currentThread*().getName() + **" completed the computation"**);  
 **countDownLatch.countDown();**  
 }  
}

**import** java.util.Map; **import** java.util.concurrent.ConcurrentHashMap;  
**import** java.util.concurrent.CountDownLatch;  
  
**public class TestCountDownLatch** {  
 **public static void** main(String[] args) {  
 CountDownLatch countDownLatch = **new** CountDownLatch(5);  
 Map<String, Integer> dataMap = **new** ConcurrentHashMap<String, Integer>();  
 Thread t1 = **new** Thread(**new** StudentThread(dataMap, **"John"**, countDownLatch, 3), **"John"**);  
 Thread t2 = **new** Thread(**new** StudentThread(dataMap, **"Vidya"**, countDownLatch, 5), **"Vidya"**);  
 Thread t3 = **new** Thread(**new** StudentThread(dataMap, **"Amit"**, countDownLatch, 7), **"Amit"**);  
 Thread t4 = **new** Thread(**new** StudentThread(dataMap, **"Ronaldo"**, countDownLatch, 3), **"Ronaldo"**);  
 Thread t5 = **new** Thread(**new** StudentThread(dataMap, **"Pihu"**, countDownLatch, 9), **"Pihu"**);  
  
 t1.start();  
 t2.start();  
 t3.start();  
 t4.start();  
 t5.start();  
 **try** {  
 **countDownLatch.await();** 🡸**It is blocking**  
 System.***out***.println(**"All students completed the computations, now show the result ..."**);  
 dataMap.forEach((key, value) -> System.***out***.println(key + **"<--->"** + value));  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
}

**OUTPUT**

John computing value ...

Vidya computing value ...

Amit computing value ...

Ronaldo computing value ...

Pihu computing value ...

John completed the computation

Ronaldo completed the computation

Vidya completed the computation

Amit completed the computation

Pihu completed the computation

All students completed the computations, now show the result ...

Ronaldo<--->97

Pihu<--->84

Vidya<--->84

John<--->57

Amit<--->25

**A precise way to writing CountDownLatch using Java 8 Lambda**

**public class** TestLatch {  
  
 **public static void** task(CountDownLatch latch, String name, **int** time) {  
 **try** {  
 Thread.*currentThread*().setName(name);  
 System.***out***.println(**"Executing Task ...."**+Thread.*currentThread*().getName());  
 TimeUnit.***SECONDS***.sleep(time);  
 System.***out***.println(**"Task completed ..."**);  
 } **catch**(InterruptedException ie) {  
 ie.printStackTrace();  
 } **finally** {  
 latch.countDown();  
 }  
 }  
  
 **public static void** main(String[] args) **throws** InterruptedException {  
 CountDownLatch latch = **new** CountDownLatch(3);  
  
 Runnable r1 = () -> *task*(latch, **"Aadhar"**,7);  
 Runnable r2 = () -> *task*(latch, **"Pan"**, 5);  
 Runnable r3 = () -> *task*(latch, **"Passport"**, 3);Thread t1 = **new** Thread(r1);  
 Thread t2 = **new** Thread(r1);  
 Thread t3 = **new** Thread(r1);  
  
 t1.start();  
 t2.start();  
 t3.start();  
  
 latch.await();  
 System.***out***.println(**"All validations completed "**);  
 }  
}