**CyclicBarrier-2024**

A CyclicBarrier is **a synchronizer that allows a set of threads to wait for each other to reach a common execution point**, also called a barrier. CyclicBarriers are used in programs in which we have a fixed number of threads that must wait for each other to reach a common point before continuing execution. The barrier is called cyclic because it can be re-used after the waiting threads are released.

**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.

**barrier.await(10, TimeUnit.SECONDS)🡺** The waiting threads waits at the CyclicBarrier until either:

**When to use CyclicBarrier in Java**

Given the nature of CyclicBarrier it can be very handy to implement map reduce kind of task similar to fork-join framework of Java 7, where a big task is broken down into smaller pieces and to complete the task you need output from individual small task e.g. to count population of India you can have 4 threads which counts population from North, South, East and West and once complete they can wait for each other, When last thread completed there task, Main thread or any other thread can add result from each zone and print total population. You can use CyclicBarrier in Java :

**Scenario - 2 : In a sports event there are 3 runners and they are running. They will run independently.**

**1. Runners will run and finally come to the barrier point where based upon the shortest time prize will be given**

**2. After the prize distribution, runners will go go home.**

**public class** TestCyclicBarrier {  
  
 **public void** sleep(**long** time) {  
 **try** {  
 TimeUnit.***SECONDS***.sleep(time);  
 } **catch** (InterruptedException e) {  
 **throw new** RuntimeException(e);  
 }  
 }  
  
 **public void** runner(CyclicBarrier barrier, Map<String, Long> timeMap, **long** time2Sleep) {  
 String threadName = Thread.*currentThread*().getName();  
 System.***out***.println(threadName + " started running ...");  
 System.***out***.println(threadName + " waiting ...");  
 sleep(time2Sleep);  
 **long** time = System.*nanoTime*();  
 timeMap.put(threadName, time);  
 **try** {  
 barrier.await();  
 System.***out***.println(threadName + " completed the event and runner will go home ...");  
 } **catch** (InterruptedException e) {  
 **throw new** RuntimeException(e);  
 } **catch** (BrokenBarrierException e) {  
 **throw new** RuntimeException(e);  
 }  
 }  
  
 **public void** distributePrize() {  
 System.***out***.println("All the threads have completed their running event.");  
 System.***out***.println("All the threads have come to this barrier point.");  
 System.***out***.println("Result will be declared from the time and prize will be awarded ...");  
 sleep(3);  
 System.***out***.println("Allow other threads to complete their remianing tasks ..");  
 }  
  
 **public void** check() {  
 **final** Map<String,Long> timeMap = **new** ConcurrentHashMap<String, Long>();  
 Runnable finalTask = () -> distributePrize();  
  
 CyclicBarrier cb = **new** CyclicBarrier(3, finalTask);  
 Runnable r1 = () -> runner(cb, timeMap, 5);  
 Runnable r2 = () -> runner(cb, timeMap, 3);  
 Runnable r3 = () -> runner(cb, timeMap, 2);  
  
 Thread th1 = **new** Thread(r1);  
 Thread th2 = **new** Thread(r2);  
 Thread th3 = **new** Thread(r3);  
  
 th1.start();  
 th2.start();  
 th3.start();  
  
 **boolean** flag = **true**;  
 **while**( flag ) {  
 **if**( !th1.isAlive() & !th2.isAlive() & !th3.isAlive() )  
 flag = **false**;  
 }  
 System.***out***.println("All the multi threaded tasks have been completed ... ");  
 }  
  
 **public static void** main(String[] args) {  
 **new** TestCyclicBarrier().check();  
 }  
}

**Problem: You need to do 3 validations in parallel and perform a final validation after this.**

Imagine, make Aadhar, PanNo, Passport validation, after this, you have to open an account.

**public class** TestCyclicBarrier {  
  
 **public void** sleep(**long** time) {  
 **try** {  
 TimeUnit.***SECONDS***.sleep(time);  
 } **catch** (InterruptedException e) {  
 **throw new** RuntimeException(e);  
 }  
 }  
  
 **public void** doAadharValdn(CyclicBarrier cyclicBarrier, **long** time) {  
 System.***out***.println(Thread.*currentThread*().getName() + " started validation");  
 sleep(time);  
 **try** {  
 cyclicBarrier.await();  
 } **catch** (InterruptedException e) {  
 **throw new** RuntimeException(e);  
 } **catch** (BrokenBarrierException e) {  
 **throw new** RuntimeException(e);  
 }  
 System.***out***.println(Thread.*currentThread*().getName() + " completed validation");  
 }

**public void** doPanValdn(CyclicBarrier cyclicBarrier, **long** time) {  
 System.***out***.println(Thread.*currentThread*().getName() + " started validation");  
 sleep(time);  
 **try** {  
 cyclicBarrier.await();  
 } **catch** (InterruptedException e) {  
 **throw new** RuntimeException(e);  
 } **catch** (BrokenBarrierException e) {  
 **throw new** RuntimeException(e);  
 }  
 System.***out***.println(Thread.*currentThread*().getName() + " completed validation");  
 }  
  
 **public void** doPassportVldn(CyclicBarrier cyclicBarrier, **long** time) {  
 System.***out***.println(Thread.*currentThread*().getName() + " started validation");  
 sleep(time);  
 **try** {  
 cyclicBarrier.await();  
 } **catch** (InterruptedException e) {  
 **throw new** RuntimeException(e);  
 } **catch** (BrokenBarrierException e) {  
 **throw new** RuntimeException(e);  
 }  
 System.***out***.println(Thread.*currentThread*().getName() + " completed validation");  
 }  
  
 **public void** openAccount() {  
 System.***out***.println("Assuming all successfull validations...");  
 sleep(2);  
 System.***out***.println("Account opened successfully ...");  
  
 }  
  
 **public void** check() {  
 **final** Map<String, Long> timeMap = **new** ConcurrentHashMap<String, Long>();  
 Runnable finalTask = () -> openAccount();  
  
 CyclicBarrier cb = **new** CyclicBarrier(3, finalTask);  
 Runnable r1 = () -> doAadharValdn(cb, 5);  
 Runnable r2 = () -> doPanValdn(cb, 3);  
 Runnable r3 = () -> doPassportVldn(cb, 2);  
  
 Thread th1 = **new** Thread(r1);  
 Thread th2 = **new** Thread(r2);  
 Thread th3 = **new** Thread(r3);  
  
 th1.start();  
 th2.start();  
 th3.start();  
  
 **try** {  
 th1.join();  
 th2.join();  
 th3.join();  
 } **catch** (InterruptedException e) {  
 **throw new** RuntimeException(e);  
 }  
  
 System.***out***.println("All the multi threaded tasks have been completed ... ");  
 }  
  
 **public static void** main(String[] args) {  
 **new** TestCyclicBarrier().check();  
 }  
}

**OUTPUT**

All the multi threaded tasks have been completed ...

Thread-1 started validation

Thread-0 started validation

Thread-2 started validation

Assuming all successfull validations...

Account opened successfully ...

Thread-0 completed validation

Thread-2 completed validation

Thread-1 completed validation

**Note: In the end task, thread name has no impact with CyclicBarrier.**

**How to change the thread name ?**

In case extending thread class, in the constructor use **super(“Thread-Name”)**. In case of Runnable, you have to user like this.

**Thread t1 = new Thread(new RunnableTask(),”Thread-Name”);**

**Good Example on CyclicBarrier as per Java 8**

**import** java.util.concurrent.BrokenBarrierException;  
**import** java.util.concurrent.CyclicBarrier;  
**import** java.util.concurrent.TimeUnit;  
  
**public class** TestBarrier {  
  
 **public static void** task(CyclicBarrier barrier, String name, **int** time) {  
 **try** {  
 Thread.*currentThread*().setName(name);  
 System.***out***.println(**"Executing Task ...."**+Thread.*currentThread*().getName());  
 TimeUnit.***SECONDS***.sleep(time);  
 barrier.await();  
 TimeUnit.***SECONDS***.sleep(2);  
 System.***out***.println(**"Task completed by ..."**+Thread.*currentThread*().getName());  
 } **catch**(InterruptedException ie) {  
 ie.printStackTrace();  
 } **catch** (BrokenBarrierException e) {  
 e.printStackTrace();  
 }  
 }  
  
 **public static void** endTask() {  
 System.***out***.println(**"All Task completed .."**);  
 }  
  
 **public static void** main(String[] args) {  
 Runnable runnable = () -> *endTask*();  
 CyclicBarrier barrier = **new** CyclicBarrier(3, runnable);  
  
 Runnable r1 = () -> *task*(barrier, **"John"**, 7);  
 Runnable r2 = () -> *task*(barrier, **"Vidya"**, 7);  
 Runnable r3 = () -> *task*(barrier, **"Peter"**, 7);  
  
 Thread t1 = **new** Thread(r1);  
 Thread t2 = **new** Thread(r2);  
 Thread t3 = **new** Thread(r3);  
  
 t1.start();  
 t2.start();  
 t3.start();  
  
 }  
}