**CyclicBarrier**

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. Here is how you specify a timeout:

**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 broker 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 :

1) To implement multi player game which can not begin until all player has joined.

2) Perform lengthy calculation by breaking it into smaller individual tasks, In general to implement Map reduce technique.

**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.**

import java.util.Map;

import java.util.concurrent.BrokenBarrierException;

import java.util.concurrent.CyclicBarrier;

import java.util.concurrent.TimeUnit;

public class Runner1 implements Runnable {

private CyclicBarrier barrier1;

private Map<String,Long> timeMap;

public Runner1(CyclicBarrier barrier1,Map<String,Long> timeMap) {

this.barrier1 = barrier1;

this.timeMap = timeMap;

}

@Override

public void run() {

try {

System.out.println(Thread.currentThread().getName()+" started running ...");

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

TimeUnit.SECONDS.sleep(2);

long time = System.nanoTime();

timeMap.put("Runner3", time);

barrier1.await();

System.out.println(Thread.currentThread().getName()+" is going to perform other tasks to complete ...");

TimeUnit.SECONDS.sleep(2);

System.out.println(Thread.currentThread().getName()+" completed the event and runner will go home ...");

}

catch( InterruptedException ie ) {

System.out.println("Interrupted Exception thrown ... ");

ie.printStackTrace();

}

catch( BrokenBarrierException bbe ) {

System.out.println("Broken Barrier Exception thrown ... ");

bbe.printStackTrace();

}

catch (Exception e) {

System.out.println("Exception thrown ... ");

e.printStackTrace();

}

}

}

import java.util.Map;

import java.util.concurrent.BrokenBarrierException;

import java.util.concurrent.CyclicBarrier;

import java.util.concurrent.TimeUnit;

public class Runner2 implements Runnable {

private CyclicBarrier barrier2;

private Map<String,Long> timeMap;

public Runner2(CyclicBarrier barrier1,Map<String,Long> timeMap) {

this.barrier2 = barrier1;

this.timeMap = timeMap;

}

@Override

public void run() {

try {

System.out.println(Thread.currentThread().getName()+" started running ...");

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

TimeUnit.SECONDS.sleep(4);

long time = System.nanoTime();

timeMap.put("Runner3", time);

barrier2.await();

System.out.println(Thread.currentThread().getName()+" is going to perform other tasks to complete ...");

TimeUnit.SECONDS.sleep(4);

System.out.println(Thread.currentThread().getName()+" completed the event and runner will go home ...");

}

catch( InterruptedException ie ) {

System.out.println("Interrupted Exception thrown ... ");

ie.printStackTrace();

}

catch( BrokenBarrierException bbe ) {

System.out.println("Broken Barrier Exception thrown ... ");

bbe.printStackTrace();

}

catch (Exception e) {

System.out.println("Exception thrown ... ");

e.printStackTrace();

}

}

}

import java.util.Map;

import java.util.concurrent.BrokenBarrierException;

import java.util.concurrent.CyclicBarrier;

import java.util.concurrent.TimeUnit;

public class Runner3 implements Runnable {

private CyclicBarrier barrier3;

private Map<String,Long> timeMap;

public Runner3(CyclicBarrier barrier1,Map<String,Long> timeMap) {

this.barrier3 = barrier1;

this.timeMap = timeMap;

}

@Override

public void run() {

try {

System.out.println(Thread.currentThread().getName()+" started running ...");

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

TimeUnit.SECONDS.sleep(6);

long time = System.nanoTime();

timeMap.put("Runner3", time);

barrier3.await();

System.out.println(Thread.currentThread().getName()+" is going to perform other tasks to complete ...");

TimeUnit.SECONDS.sleep(6);

System.out.println(Thread.currentThread().getName()+" completed the event and runner will go home ...");

}

catch( InterruptedException ie ) {

System.out.println("Interrupted Exception thrown ... ");

ie.printStackTrace();

}

catch( BrokenBarrierException bbe ) {

System.out.println("Broken Barrier Exception thrown ... ");

bbe.printStackTrace();

}

catch (Exception e) {

System.out.println("Exception thrown ... ");

e.printStackTrace();

}

}

}

import java.util.List;

import java.util.Map;

import java.util.concurrent.ConcurrentHashMap;

import java.util.concurrent.CopyOnWriteArrayList;

import java.util.concurrent.CyclicBarrier;

import java.util.concurrent.TimeUnit;

public class TestCyclicBarrier {

public static void main(String[] args) {

final Map<String,Long> timeMap = new ConcurrentHashMap<String, Long>();

CyclicBarrier barrier = new CyclicBarrier(3, new Runnable() {

@Override

public void run() {

try {

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 ...");

TimeUnit.SECONDS.sleep(3);

System.out.println("Allow other threads to complete their remianing tasks ..");

}

catch (Exception e) {

e.printStackTrace();

}

}

});

Runnable runnable1 = new Runner1( barrier,timeMap);

Runnable runnable2 = new Runner1( barrier,timeMap);

Runnable runnable3 = new Runner1( barrier,timeMap);

Thread th1 = new Thread(runnable1);

Thread th2 = new Thread(runnable2);

Thread th3 = new Thread(runnable3);

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 ... ");

}

}

**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 do a fourth validation.

**import** java.util.concurrent.BrokenBarrierException;  
**import** java.util.concurrent.CyclicBarrier;  
**import** java.util.concurrent.TimeUnit;  
  
**public class AadharValidation** **implements** Runnable {  
 **private CyclicBarrier cyclicBarrier;** **private int timeInSecs**;  
  
 **public** AadharValidation(CyclicBarrier cyclicBarrier, **int** timeInSecs) {  
 **this**.**cyclicBarrier** = cyclicBarrier;  
 **this**.**timeInSecs** = timeInSecs;  
 }  
  
 @Override  
 **public void** run() {  
 **try** {  
 System.***out***.println(Thread.*currentThread*().getName() + **" started validation"**);  
 TimeUnit.***SECONDS***.sleep(**timeInSecs**);  
 System.***out***.println(Thread.*currentThread*().getName() + **" waiting ..."**);  
 **cyclicBarrier.await();** System.***out***.println(Thread.*currentThread*().getName() + **" completed validation"**);  
 } **catch** (InterruptedException | BrokenBarrierException e) {  
 e.printStackTrace();  
 }  
 }  
}

**import** java.util.concurrent.BrokenBarrierException;  
**import** java.util.concurrent.CyclicBarrier;  
**import** java.util.concurrent.TimeUnit;  
  
**public class PanNoValidation** **implements** Runnable {  
 **private CyclicBarrier cyclicBarrier;** **private int timeInSecs**;  
  
 **public** PanNoValidation(CyclicBarrier cyclicBarrier, **int** timeInSecs) {  
 **this**.**cyclicBarrier** = cyclicBarrier;  
 **this**.**timeInSecs** = timeInSecs;  
 }  
  
 @Override  
 **public void** run() {  
 **try** {  
 System.***out***.println(Thread.*currentThread*().getName() + **" started validation"**);  
 TimeUnit.***SECONDS***.sleep(**timeInSecs**);  
 System.***out***.println(Thread.*currentThread*().getName() + **" waiting ..."**);  
 **cyclicBarrier.await();** System.***out***.println(Thread.*currentThread*().getName() + **" completed validation"**);  
 } **catch** (InterruptedException | BrokenBarrierException e) {  
 e.printStackTrace();  
 }  
 }  
}

**import** java.util.concurrent.BrokenBarrierException;  
**import** java.util.concurrent.CyclicBarrier;  
**import** java.util.concurrent.TimeUnit;  
  
**public class PassportValidation** **implements** Runnable {  
 **private CyclicBarrier cyclicBarrier;** **private int timeInSecs**;  
  
 **public** PassportValidation(CyclicBarrier cyclicBarrier, **int** timeInSecs) {  
 **this**.**cyclicBarrier** = cyclicBarrier;  
 **this**.**timeInSecs** = timeInSecs;  
 }  
  
 @Override  
 **public void** run() {  
 **try** {  
 System.***out***.println(Thread.*currentThread*().getName() + **" started validation"**);  
 TimeUnit.***SECONDS***.sleep(**timeInSecs**);  
 System.***out***.println(Thread.*currentThread*().getName() + **" waiting ..."**);  
 **cyclicBarrier.await();** System.***out***.println(Thread.*currentThread*().getName() + **" completed validation"**);  
 } **catch** (InterruptedException | BrokenBarrierException e) {  
 e.printStackTrace();  
 }  
 }  
}

**import** java.util.concurrent.BrokenBarrierException;  
**import** java.util.concurrent.CyclicBarrier;  
**import** java.util.concurrent.TimeUnit;  
  
**public class EndTask** **implements** Runnable {  
 **private int timeInSecs**;  
  
 **public** EndTask(**int** timeInSecs) {  
 **this**.**timeInSecs** = timeInSecs;  
 }  
  
 @Override  
 **public void** run() {  
 **try** {  
 System.***out***.println(**"All the parties have arrived here..."**);  
 System.***out***.println(**"Final task is going to be performed now ..."**);  
 TimeUnit.***SECONDS***.sleep(**timeInSecs**);  
 System.***out***.println(**"Now other threads can complete the tasks"**);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
}

**import** java.util.concurrent.BrokenBarrierException;  
**import** java.util.concurrent.CyclicBarrier;  
**import** java.util.concurrent.TimeUnit;  
  
**public class** TestCyclicBarrier1 {  
 **public static void** main(String[] args) **throws** Exception {  
 *// Thread name has no impact in case of CyclicBarrier Runnable task.* Thread endTask = **new** Thread(**new** EndTask(3));  
 CyclicBarrier cyclicBarrier = **new** CyclicBarrier(3, endTask);  
 *// You can also write like this  
 // CyclicBarrier cyclicBarrier = new CyclicBarrier(3);* Thread aadharThread = **new** Thread(**new** AadharValidation(cyclicBarrier, 7), **"Aadhar"**);  
 Thread passportThread = **new** Thread(**new** PassportValidation(cyclicBarrier, 5), **"Passport"**);  
 Thread panNoThread = **new** Thread(**new** PanNoValidation(cyclicBarrier, 3), **"PanNo"**);  
  
 aadharThread.start();  
 passportThread.start();  
 panNoThread.start();  
  
 aadharThread.join();  
 passportThread.join();  
 panNoThread.join();  
  
 System.***out***.println(**"All threads completed the tasks"**);  
 }  
}

OUTPUT

Passport started validation

Aadhar started validation

PanNo started validation

PanNo waiting ...

Passport waiting ...

Aadhar waiting ...

All the parties have arrived here...

Final task is going to be performed now ...

Now other threads can complete the tasks

Aadhar completed validation

Passport completed validation

PanNo completed validation

All threads completed the tasks

**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();  
  
 }  
}