***PrinterQueue.java file***

import java.util.Date;

import java.util.concurrent.Semaphore;

class PrinterQueue

{

private final Semaphore semaphore;

public PrinterQueue()

{

semaphore = new Semaphore(1);

}

public void printJob(Object document)

{

try

{

semaphore.acquire();

Long duration = (long) (Math.random() \* 10000);

System.out.println(Thread.currentThread().getName() + ": PrintQueue: Printing a Job during " + (duration / 1000) + " seconds :: Time - " + new Date());

Thread.sleep(duration);

}

catch (InterruptedException e)

{

e.printStackTrace();

}

finally

{

System.out.printf("%s: The document has been printed\n", Thread.currentThread().getName());

semaphore.release();

}

}

}

***PrintingJob.java file:***

class PrintingJob implements Runnable

{

private PrinterQueue printerQueue;

public PrintingJob(PrinterQueue printerQueue)

{

this.printerQueue = printerQueue;

}

@Override

public void run()

{

System.out.printf("%s: Going to print a document\n", Thread.currentThread().getName());

printerQueue.printJob(new Object());

}

}

***SemaphoreDemo.java file:***

package com.techvidvan.semaphore;

import java.util.concurrent. \* ;

//A shared resource/class.

class SharedResource {

static int count = 0;

}

class MyThread extends Thread {

Semaphore semaphore;

String threadName;

public MyThread(Semaphore semaphore, String threadName) {

super(threadName);

this.semaphore = semaphore;

this.threadName = threadName;

}@Override

public void run() {

if (this.getName().equals("Thread1")) {

System.out.println("Starting " + threadName);

try {

// First, get a permit.

System.out.println(threadName + " is waiting for a permit.");

//Acquiring the lock

semaphore.acquire();

System.out.println(threadName + " gets a permit.");

// Accessing the shared resource.

// other waiting threads will wait, until this thread releases the lock

for (int i = 0; i < 5; i++) {

SharedResource.count++;

System.out.println(threadName + ": " + SharedResource.count);

//Allowing a context switch if possible.for thread B to execute

Thread.sleep(10);

}

}

catch(InterruptedException exc) {

System.out.println(exc);

}

// Release the permit.

System.out.println(threadName + " releases the permit.");

semaphore.release();

}

// Run by thread B

else {

System.out.println("Starting " + threadName);

try {

// First, get a permit.

System.out.println(threadName + " is waiting for a permit.");

// acquiring the lock

semaphore.acquire();

System.out.println(threadName + " gets a permit.");

// Now, accessing the shared resource.

// other waiting threads will wait, until this thread release the lock

for (int i = 0; i < 5; i++) {

SharedResource.count--;

System.out.println(threadName + ": " + SharedResource.count);

Thread.sleep(10);

}

}

catch(InterruptedException exc) {

System.out.println(exc);

}

// Release the permit.

System.out.println(threadName + " releases the permit.");

semaphore.release();

}

}

}

//Main class

public class SemaphoreDemo {

public static void main(String args[]) throws InterruptedException {

// Creating a Semaphore object with number of permits = 1

Semaphore semaphore = new Semaphore(1);

// Creating two threads with name t1 and t2

// Note that thread A will increment the count and thread B will decrement the count

MyThread t1 = new MyThread(semaphore, "Thread1");

MyThread t2 = new MyThread(semaphore, "Thread2");

t1.start();

t2.start();

// waiting for threads t1 and t2

t1.join();

t2.join();

//count will always be 0 after both threads complete their execution

System.out.println("count: " + SharedResource.count);

}

}

***SemaphoreExample.java file:***

public class SemaphoreExample

{

public static void main(String[] args)

{

PrinterQueue printerQueue = new PrinterQueue();

Thread thread[] = new Thread[10];

for (int i = 0; i < 10; i++)

{

thread[i] = new Thread(new PrintingJob(printerQueue), "Thread " + i);

}

for (int i = 0; i < 10; i++)

{

thread[i].start();

}

}

}