**How to Use Locks in Java | java.util.concurrent.locks.Lock Tutorial and Example**

We are already aware of basic concepts around [**thread synchronization**](https://howtodoinjava.com/java/multi-threading/thread-synchronization-object-level-locking-and-class-level-locking/) and various mechanisms using **synchronized** keyword. Java provides another mechanism for the synchronization of blocks of code based on the [Lock](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/locks/Lock.html) interface and classes that implement it (such as [ReentrantLock](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/locks/ReentrantLock.html" \o "ReentrantLock" \t "_blank)). In this tutorial, we will see a basic usage of Lock interface to solve **printer queue problem**.

**Lock Interface**

A java.util.concurrent.locks.Lock is a thread synchronization mechanism just like synchronized blocks. A Lock is, however, more flexible and more sophisticated than a synchronized block. Since Lock is an interface, you need to use one of its implementations to use a Lock in your applications. ReentrantLock is one such implementation of Lock interface.

Here is the simple use of Lock interface.

|  |
| --- |
| Lock lock = new ReentrantLock();    lock.lock();    //critical section    lock.unlock(); |

First a Lock is created. Then it’s lock() method is called. Now the Lock instance is locked. Any other thread calling lock() will be blocked until the thread that locked the lock calls unlock(). Finally unlock() is called, and the Lock is now unlocked so other threads can lock it.

**Difference between Lock Interface and synchronized keyword**

The main differences between a Lock and a synchronized block are:

1) Having a timeout trying to get access to a synchronized block is not possible. Using [Lock.tryLock(long timeout, TimeUnit timeUnit)](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/locks/Lock.html" \l "tryLock%28long,%20java.util.concurrent.TimeUnit%29" \o "lock method" \t "_blank), it is possible.  
2) The synchronized block must be fully contained within a single method. A Lock can have it’s calls to lock() and unlock() in separate methods.

**Simulating Printer Queue using Locks**

In this example, program will simulate the behavior of a printer. You can submit a number of print jobs to printer during varying time interval or simultaneously. Printer will take a job from printer queue and print it. Rest of jobs will wait there for their turn. Once printer is done with print job in hand, it will pick another job from queue and start printing. Keep this happening in a loop.

**PrintingJob.java**

This class represents an independent printing which could be submitted to printer. This class implements Runnable interface, so that printer can execute it when it’s turn come.

|  |
| --- |
| 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());     }  } |

**PrinterQueue.java**

This class represent the printer queue/ printer. A lock is maintained by printer to start new print job as soon as current print job is finished.

|  |
| --- |
| class PrinterQueue  {     private final Lock queueLock = new ReentrantLock();       public void printJob(Object document)     {        queueLock.lock();        try        {           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());           queueLock.unlock();        }     }  } |

Let’s test our printer program:

|  |
| --- |
| public class LockExample  {     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();        }     }  }    Output:    Thread 0: Going to print a document  Thread 9: Going to print a document  Thread 8: Going to print a document  Thread 7: Going to print a document  Thread 5: Going to print a document  Thread 6: Going to print a document  Thread 4: Going to print a document  Thread 3: Going to print a document  Thread 2: Going to print a document  Thread 1: Going to print a document  Thread 0: PrintQueue: Printing a Job during 8 seconds :: Time - Tue Jan 06 15:19:02 IST 2015  Thread 0: The document has been printed  Thread 9: PrintQueue: Printing a Job during 1 seconds :: Time - Tue Jan 06 15:19:11 IST 2015  Thread 9: The document has been printed  Thread 8: PrintQueue: Printing a Job during 8 seconds :: Time - Tue Jan 06 15:19:12 IST 2015  Thread 8: The document has been printed  Thread 7: PrintQueue: Printing a Job during 9 seconds :: Time - Tue Jan 06 15:19:21 IST 2015  Thread 7: The document has been printed  Thread 5: PrintQueue: Printing a Job during 7 seconds :: Time - Tue Jan 06 15:19:31 IST 2015  Thread 5: The document has been printed  Thread 6: PrintQueue: Printing a Job during 5 seconds :: Time - Tue Jan 06 15:19:39 IST 2015  Thread 6: The document has been printed  Thread 4: PrintQueue: Printing a Job during 2 seconds :: Time - Tue Jan 06 15:19:44 IST 2015  Thread 4: The document has been printed  Thread 3: PrintQueue: Printing a Job during 2 seconds :: Time - Tue Jan 06 15:19:46 IST 2015  Thread 3: The document has been printed  Thread 2: PrintQueue: Printing a Job during 5 seconds :: Time - Tue Jan 06 15:19:49 IST 2015  Thread 2: The document has been printed  Thread 1: PrintQueue: Printing a Job during 5 seconds :: Time - Tue Jan 06 15:19:54 IST 2015  Thread 1: The document has been printed |

The key to the example is in the printJob() method of the PrinterQueue class. When we want to implement a critical section using locks and guarantee that only one execution thread runs a block of code, we have to create a ReentrantLock object. At the beginning of the critical section, we have to get the control of the lock using the lock() method.

At the end of the critical section, we have to use the unlock() method to free the control of the lock and allow the other threads to run this critical section. **If you don’t call the unlock() method at the end of the critical section, the other threads that are waiting for that block will be waiting forever, causing a deadlock situation.** If you use try-catch blocks in your critical section, don’t forget to put the sentence containing the unlock() method inside the finally section.