**Control concurrent access to multiple copies of a resource using Semaphore**

In previous tutorial, we learned about [**binary semaphore**](https://howtodoinjava.com/java/multi-threading/binary-semaphore-tutorial-and-example/) which can be used to control access to single copy of a resource using the counter value either 0 or 1. However, semaphores can also be used when you need to protect various copies of a resource that can be executed by more than one thread at the same time. In this example, we will learn **how to use a semaphore to protect more than one copy of a resource**.

Let’s revisit semaphore concept before moving ahead.

**How Semaphores Work?**

You can visualize a semaphore as counter which can be incremented or decremented. You initialize the semaphore with a number i.e. 5. Now this semaphore can be decremented maximum five times in a row until counter reaches to 0. Once counter is zero, you can increment it to maximum five times to make it 5. The counter value of semaphore MUST always be inside limit 0 >= n >= 5 (in our case).

Obviously, semaphores are more than just being counters. They are able to make threads wait when counter value is zero i.e. they act as Locks with counter functionality.

Talking in terms of [multi-threading](https://howtodoinjava.com/category/java/multi-threading/), when a thread wants to access one of shared resources (guarded by semaphore), first, it must acquire the semaphore. If the internal counter of the semaphore is greater than 0, the semaphore decrements the counter and allows access to the shared resource. Otherwise, if the counter of the semaphore is 0, the semaphore puts the thread to sleep until the counter is greater than 0. A value of 0 in the counter means all the shared resources are used by other threads, so the thread that wants to use one of them must wait until one is free.

When a thread has finished the use of the shared resource, it must release the semaphore so that the other threads can access the shared resource. That operation increases the internal counter of the semaphore.

**Read More:**[**How to Use Locks in Java**](https://howtodoinjava.com/java/multi-threading/how-to-use-locks-in-java-java-util-concurrent-locks-lock-tutorial-and-example/)

**How to use semaphore?**

To demonstrate the concept, we will be using semaphore for controlling 3 printers which can print multiples documents simultaneously.

**PrintingJob.java**

This class represents an independent printing job which could be submitted to printer queue. And from queue, it can be picked up by any printer and performed printing job. 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. This class has 3 main attributes which control the logic of selecting a free printer out of 3 printers and lock it for printing a job. After printing the document, printer is released so that it is again free and available for printing a new job from print queue.

This class has two methods getPrinter() and releasePrinter() which are responsible for acquiring a free printer and putting it back in free printers pool.

Another method printJob() actually does the core job i.e. acquiring a printer, execute print job and then release the printer.

It uses below two variables for doing the job:

**semaphore** : This variable keep track of no. of printers used at any point of time.  
**printerLock** : Used for locking the printer pool before checking/acquiring a free printer out of three available printers.

|  |
| --- |
| class PrinterQueue  {      //This Semaphore will keep track of no. of printers used at any point of time.      private final Semaphore semaphore;        //While checking/acquiring a free printer out of three available printers, we will use this lock.      private final Lock printerLock;        //This array represents the pool of free printers.      private boolean freePrinters[];        public PrinterQueue()      {          semaphore = new Semaphore(3);          freePrinters = new boolean[3];          for (int i = 0; i < 3; i++) {              freePrinters[i] = true;          }          printerLock = new ReentrantLock();      }        public void printJob(Object document)      {          try          {              //Decrease the semaphore counter to mark a printer busy              semaphore.acquire();                //Get the free printer              int assignedPrinter = getPrinter();                //Print the job              Long duration = (long) (Math.random() \* 10000);              System.out.println(Thread.currentThread().getName()                      + ": Printer " + assignedPrinter                      + " : Printing a Job during " + (duration / 1000)                      + " seconds :: Time - " + new Date());              Thread.sleep(duration);                //Printing is done; Free the printer to be used by other threads.              releasePrinter(assignedPrinter);          }          catch (InterruptedException e) {              e.printStackTrace();          }          finally {              System.out.printf("%s: The document has been printed\n", Thread                      .currentThread().getName());                //Increase the semaphore counter back              semaphore.release();          }      }        //Acquire a free printer for printing a job      private int getPrinter()      {          int foundPrinter = -1;          try {              //Get a lock here so that only one thread can go beyond this at a time              printerLock.lock();                //Check which printer is free              for (int i = 0; i < freePrinters.length; i++)              {                  //If free printer found then mark it busy                  if (freePrinters[i])                  {                      foundPrinter = i;                      freePrinters[i] = false;                      break;                  }              }          }          catch (Exception e) {              e.printStackTrace();          } finally          {              //Allow other threads to check for free priniter              printerLock.unlock();          }          return foundPrinter;      }        //Release the printer      private void releasePrinter(int i) {          printerLock.lock();          //Mark the printer free          freePrinters[i] = true;          printerLock.unlock();      }  } |

**Read More :**[**How to use binary semaphore?**](https://howtodoinjava.com/java/multi-threading/binary-semaphore-tutorial-and-example/)

Let’s test our printer program:

|  |
| --- |
| 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();          }      }  }    Output:    Thread 1: Going to print a document  Thread 4: Going to print a document  Thread 9: Going to print a document  Thread 8: Going to print a document  Thread 6: Going to print a document  Thread 7: Going to print a document  Thread 2: Going to print a document  Thread 5: Going to print a document  Thread 3: Going to print a document  Thread 0: Going to print a document  Thread 9: PrintQueue 2 : Printing a Job during 2 seconds :: Time - Tue Jan 13 16:28:58 IST 2015  Thread 4: PrintQueue 1 : Printing a Job during 7 seconds :: Time - Tue Jan 13 16:28:58 IST 2015  Thread 1: PrintQueue 0 : Printing a Job during 1 seconds :: Time - Tue Jan 13 16:28:58 IST 2015  Thread 1: The document has been printed  Thread 8: PrintQueue 0 : Printing a Job during 1 seconds :: Time - Tue Jan 13 16:29:00 IST 2015  Thread 9: The document has been printed  Thread 6: PrintQueue 2 : Printing a Job during 0 seconds :: Time - Tue Jan 13 16:29:01 IST 2015  Thread 6: The document has been printed  Thread 7: PrintQueue 2 : Printing a Job during 4 seconds :: Time - Tue Jan 13 16:29:01 IST 2015  Thread 8: The document has been printed  Thread 2: PrintQueue 0 : Printing a Job during 5 seconds :: Time - Tue Jan 13 16:29:02 IST 2015  Thread 7: The document has been printed  Thread 5: PrintQueue 2 : Printing a Job during 8 seconds :: Time - Tue Jan 13 16:29:05 IST 2015  Thread 4: The document has been printed  Thread 3: PrintQueue 1 : Printing a Job during 4 seconds :: Time - Tue Jan 13 16:29:06 IST 2015  Thread 2: The document has been printed  Thread 0: PrintQueue 0 : Printing a Job during 4 seconds :: Time - Tue Jan 13 16:29:08 IST 2015  Thread 3: The document has been printed  Thread 0: The document has been printed  Thread 5: The document has been printed |

**In above example, the Semaphore object is created using 3 as the parameter of the constructor. The first three threads that call the acquire() method will get the access to printers while the rest will be blocked. When a thread finishes the critical section and releases the semaphore, another thread will acquire it.**