Semaphore used for implementing Producer Consumer pattern in java

In previous thread concurrency tutorial we learned what is java.util.concurrent.**Semaphore** in java. Now, we will learn ***Application*** *of Semaphore in real world (for solving Producer Consumer problem in java).*

Contents of page:

In last post we learned how to use Semaphores in java. Now, let's use Semaphore for implementing Producer Consumer pattern

1) Logic behind using Semaphore for implementing Producer Consumer pattern >

Semaphore on producer is created with permit =1. So, that **producer can get the permit to produce. Semaphore on consumer is created with permit =0.** So, that **consumer could wait for permit to consume.** [because initially producer hasn't produced any product]

Producer gets permit by calling **semaphoreProducer.acquire()** and **starts producing**, **after producing** it calls **semaphoreConsumer.release().** So, that **consumer could get the permit to consume.**

```
semaphoreProducer.acquire();
System.out.println("Produced : "+i);
semaphoreConsumer.release();
```

Consumer gets permit by calling **semaphoreConsumer.acquire()** and **starts consuming**, **after consuming** it calls **semaphoreProducer.release()**. So, that **producer could get the permit to produce.**

```
semaphoreConsumer.acquire();
System.out.println("Consumed : "+i);
semaphoreProducer.release();
```

2) Program to demonstrate usage of Semaphore for implementing Producer Consumer pattern >

```
import java.util.concurrent.Semaphore;

/** Copyright (c), AnkitMittal JavaMadeSoEasy.com */
public class ConsumerProducer{

    public static void main(String[] args) {

           Semaphore semaphoreProducer=new Semaphore(1);
           Semaphore semaphoreConsumer=new Semaphore(0);
           System.out.println("semaphoreProducer permit=1 | semaphoreConsumer permit=0");

       Producer producer=new Producer(semaphoreProducer,semaphoreConsumer);
       Consumer consumer=new Consumer(semaphoreConsumer,semaphoreProducer);

        Thread producerThread = new Thread(producer, "ProducerThread");
        Thread consumerThread = new Thread(consumer, "ConsumerThread");

        producerThread.start();
        consumerThread.start();

    }
}




/**
 * Producer Class.
 */
class Producer implements Runnable{

    Semaphore semaphoreProducer;
    Semaphore semaphoreConsumer;


    public Producer(Semaphore semaphoreProducer,Semaphore semaphoreConsumer) {
           this.semaphoreProducer=semaphoreProducer;
           this.semaphoreConsumer=semaphoreConsumer;
    }

    public void run() {
```

```
            for(int i=1;i<=5;i++){
                  try {
                        semaphoreProducer.acquire();
                        System.out.println("Produced : "+i);
                        semaphoreConsumer.release();

                  } catch (InterruptedException e) {
                        e.printStackTrace();
                  }
            }
      }
}

/**
 * Consumer Class.
 */
class Consumer implements Runnable{

      Semaphore semaphoreConsumer;
      Semaphore semaphoreProducer;

      public Consumer(Semaphore semaphoreConsumer,Semaphore semaphoreProducer) {
            this.semaphoreConsumer=semaphoreConsumer;
            this.semaphoreProducer=semaphoreProducer;
      }

      public void run() {

            for(int i=1;i<=5;i++){
                  try {
                        semaphoreConsumer.acquire();
                        System.out.println("Consumed : "+i);
                        semaphoreProducer.release();
                  } catch (InterruptedException e) {
                        e.printStackTrace();
                  }
            }
      }

}


/*OUTPUT

semaphoreProducer permit=1 | semaphoreConsumer permit=0
Produced : 1
Consumed : 1
Produced : 2
Consumed : 2
Produced : 3
Consumed : 3
Produced : 4
Consumed : 4
Produced : 5
Consumed : 5

*/
```

Let's discuss output in detail, to get better understanding of how we have used Semaphore for implementing Producer Consumer pattern >

Note : (I have mentioned output in `green` text and it's explanation is given in line immediately followed by it)

`semaphoreProducer permit=1 | semaphoreConsumer permit=0`
semaphoreProducer created with permit=1. So, that producer can get the permit to produce | semaphoreConsumer created with permit=0. So, that consumer could wait for permit to consume.

semaphoreProducer.acquire() is called, Producer has got the permit and it can produce [Now, semaphoreProducer permit=0]
`Produced : 1` [as producer has got permit, it is producing]
semaphoreConsumer.release() is called, `Permit has been released on` semaphoreConsumer means consumer can consume `[Now,` semaphoreConsumer permit=1`]`

semaphoreConsumer.acquire() is called, `Consumere has got the permit and it can consume [Now,` semaphoreConsumer permit=0`]`
`Consumed : 1` [as consumer has got permit, it is consuming]
semaphoreProducer.release() is called, `Permit has been released on` semaphoreProducer means producer can produce `[Now,` semaphoreProducer permit=1`]`

```
Produced : 2
Consumed : 2
Produced : 3
Consumed : 3
Produced : 4
Consumed : 4
Produced : 5
Consumed : 5
```