solve Consumer Producer problem by using wait() and notify() methods, where consumer can consume only when production is over in java

solve **Consumer Producer** problem by using **wait() and notify()** methods, where **consumer can consume only when production is over.**

Producer will allow consumer to consume only when 10 products have been produced (i.e. when production is over).
It's important to know that **sharedQueue is a queue implemented using Linked List.**

In program consumer thread will start() and wait by calling wait() method till producer is producing. Once production is over, producer thread will call notify() method, which will notify consumer thread and consumer will start consuming.

```
import java.util.LinkedList;
import java.util.List;

/**
 * Producer Class in java, Producer will allow consumer to
 * consume only when 10 products have been produced
 * (i.e. when production is over).
 */
class Producer implements Runnable{

    List<Integer> sharedQueue;

    Producer(){
           sharedQueue=new LinkedList<Integer>();
    }

    @Override
    public void run(){

           synchronized (this) {
                  for(int i=1;i<=10;i++){ //Producer will produce 10 products
                        sharedQueue.add(i);
                        System.out.println("Producer is still Producing, Produced : "+i);

                        try{
                               Thread.sleep(1000);
                        }catch(InterruptedException e){e.printStackTrace();}

                  }
                  System.out.println("Production is over, consumer can consume.");
                  //Production is over, notify consumer thread so that consumer can consume.
                  this.notify();
           }
    }

}

/**
 * Consumer Class.
 */
class Consumer extends Thread{
    Producer prod;

    Consumer(Producer obj){
     prod=obj;
    }

    public void run(){
           /*
            * consumer will wait till producer is producing.
            */
           synchronized (this.prod) {

                  System.out.println("Consumer waiting for production to get over.");
                     try{
                         this.prod.wait();
                        }catch(InterruptedException e){e.printStackTrace();}

           }


           /*production is over, consumer will start consuming.*/
           int productSize=this.prod.sharedQueue.size();
           for(int i=0;i<productSize;i++)
                  System.out.println("CONSUMED : "+ this.prod.sharedQueue.remove(0) +" ");

    }

}
```

```java
/** Copyright (c), AnkitMittal JavaMadeSoEasy.com */
public class ProducerConsumerWithWaitNotify {
    public static void main(String args[]) throws InterruptedException{

            Producer prod=new Producer();
            Consumer cons=new Consumer(prod);

            Thread prodThread=new Thread(prod,"prodThread");
            Thread consThread=new Thread(cons,"consThread");

            consThread.start();     //start consumer thread.
            Thread.sleep(100);      //This minor delay will ensure that consumer thread starts
before producer thread.
            prodThread.start();     //start producer thread.


    }

}

/*OUTPUT

Consumer waiting for production to get over.
Producer is still Producing, Produced : 1
Producer is still Producing, Produced : 2
Producer is still Producing, Produced : 3
Producer is still Producing, Produced : 4
Producer is still Producing, Produced : 5
Producer is still Producing, Produced : 6
Producer is still Producing, Produced : 7
Producer is still Producing, Produced : 8
Producer is still Producing, Produced : 9
Producer is still Producing, Produced : 10
Production is over, consumer can consume.
CONSUMED : 1
CONSUMED : 2
CONSUMED : 3
CONSUMED : 4
CONSUMED : 5
CONSUMED : 6
CONSUMED : 7
CONSUMED : 8
CONSUMED : 9
CONSUMED : 10

*/
```