

Contents of page :

- **Key points** we need to ensure before programming :
- Explanation of **Logic** >
- Full Program/sourceCode to solve consumer producer problem using wait() and notify() method>

Here come the time to answer **very very important question from interview perspective**. Interviewers tends to check how sound you are in the inter communication. Because for solving this problem we got to **use synchronization blocks, [wait\(\) and notify\(\)](#) method very cautiously**. If **misplace [synchronization block or any of the method](#)**, that **may cause your program to go horribly wrong**. So, before going into this question first i'll recommend you to understand how to use synchronized blocks, [wait\(\) and notify\(\) methods](#).

Key points we need to ensure before programming :

>Producer will produce total of 10 products and cannot produce more than 2 products at a time until products are being consumed by consumer

Example> when [sharedQueue's size](#) is 2, wait for consumer to consume (consumer will consume by calling remove(0) method on [sharedQueue](#) and reduce [sharedQueue's size](#)). As soon as size is less than 2, producer will start producing.

>Consumer can consume only when there are some products to consume.

Example> when [sharedQueue's size](#) is 0, wait for producer to produce (producer will produce by calling add() method on [sharedQueue](#) and increase [sharedQueue's size](#)). As soon as size is greater than 0, consumer will start consuming.

Explanation of **Logic** >

It's important to know that **sharedQueue** is a [queue implemented using Linked List](#).

We will create sharedQueue that will be shared amongst Producer and Consumer. We will now start consumer and producer thread.

Note: it does not matter order in which threads are started (because rest of code has taken care of synchronization and key points mentioned above)

First we will start consumerThread >

```
consumerThread.start();
```

consumerThread will enter run method and call consume() method. There it will check for sharedQueue's size.

-if size is equal to 0 that means producer hasn't produced any product, wait for producer to produce by using below piece of code-

```
synchronized (sharedQueue) {
    while (sharedQueue.size() == 0) {
        sharedQueue.wait();
    }
}
```

-if size is greater than 0, consumer will start consuming by using below piece of code.

```
synchronized (sharedQueue) {
    Thread.sleep((long)(Math.random() * 2000));
    System.out.println("consumed : "+ sharedQueue.remove(0));
    sharedQueue.notify();
}
```

Than we will start producerThread >

```
producerThread.start();
```

producerThread will enter run method and call produce() method. There it will check for sharedQueue's size.

-if size is equal to 2 (i.e. maximum number of products which sharedQueue can hold at a time), wait for consumer to consume by using below piece of code-

```
synchronized (sharedQueue) {
    while (sharedQueue.size() == maxSize) { //maxsize is 2
        sharedQueue.wait();
    }
}
```

-if size is less than 2, producer will start producing by using below piece of code.

```
synchronized (sharedQueue) {
    System.out.println("Produced : " + i);
    sharedQueue.add(i);
    Thread.sleep((long)(Math.random() * 1000));
}
```

```
sharedQueue.notify();
```

```
}
```

Full Program/sourceCode to solve consumer producer problem using wait() and notify() method>

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

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

    private List<Integer> sharedQueue;
    private int maxSize=2; //maximum number of products which sharedQueue can hold at a time.

    public Producer(List<Integer> sharedQueue) {
        this.sharedQueue = sharedQueue;
    }

    @Override
    public void run() {
        for (int i = 1; i <= 10; i++) { //produce 10 products.
            try {
                produce(i);
            } catch (InterruptedException e) { e.printStackTrace(); }
        }

        private void produce(int i) throws InterruptedException {

            synchronized (sharedQueue) {
                //if sharedQueue is full wait until consumer consumes.
                while (sharedQueue.size() == maxSize) {
                    System.out.println("Queue is full, producerThread is waiting for "
                        + "consumerThread to consume, sharedQueue's size= "+maxSize);
                    sharedQueue.wait();
                }
            }

            /* 2 Synchronized blocks have been used means before
            * producer produces by entering below synchronized
            * block consumer can consume.
            */

            //as soon as producer produces (by adding in sharedQueue) it notifies consumerThread.
            synchronized (sharedQueue) {
                System.out.println("Produced : " + i);
                sharedQueue.add(i);
                Thread.sleep((long)(Math.random() * 1000));
                sharedQueue.notify();
            }
        }
    }

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

        private List<Integer> sharedQueue;
        public Consumer(List<Integer> sharedQueue) {
            this.sharedQueue = sharedQueue;
        }

        @Override
        public void run() {
            while (true) {
                try {
                    consume();
                    Thread.sleep(100);
                } catch (InterruptedException e) { e.printStackTrace(); }
            }

            private void consume() throws InterruptedException {

                synchronized (sharedQueue) {
                    //if sharedQueue is empty wait until producer produces.
                    while (sharedQueue.size() == 0) {
                        System.out.println("Queue is empty, consumerThread is waiting for "
                            + "producerThread to produce, sharedQueue's size= 0");
                        sharedQueue.wait();
                    }
                }
            }
        }
    }
}
```

```

/* 2 Synchronized blocks have been used means before
 * consumer start consuming by entering below synchronized
 * block producer can produce.
 */

/*If sharedQueue not empty consumer will consume
 * (by removing from sharedQueue) and notify the producerThread.
 */
synchronized (sharedQueue) {
    Thread.sleep((long)(Math.random() * 2000));
    System.out.println("CONSUMED : "+ sharedQueue.remove(0));
    sharedQueue.notify();
}
}

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

    public static void main(String args[]) {
        List<Integer> sharedQueue = new LinkedList<Integer>(); //Creating shared object

        Producer producer=new Producer(sharedQueue);
        Consumer consumer=new Consumer(sharedQueue);

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

/*OUTPUT

Queue is empty, consumerThread is waiting for producerThread to produce, sharedQueue's size= 0
Produced : 1
CONSUMED : 1
Produced : 2
CONSUMED : 2
Produced : 3
Produced : 4
CONSUMED : 3
Produced : 5
Queue is full, producerThread is waiting for consumerThread to consume, sharedQueue's size= 2
CONSUMED : 4
Produced : 6
Queue is full, producerThread is waiting for consumerThread to consume, sharedQueue's size= 2
CONSUMED : 5
Produced : 7
CONSUMED : 6
Produced : 8
Queue is full, producerThread is waiting for consumerThread to consume, sharedQueue's size= 2
CONSUMED : 7
Produced : 9
CONSUMED : 8
Produced : 10
CONSUMED : 9
CONSUMED : 10
Queue is empty, consumerThread is waiting for producerThread to produce, sharedQueue's size= 0

*/

```