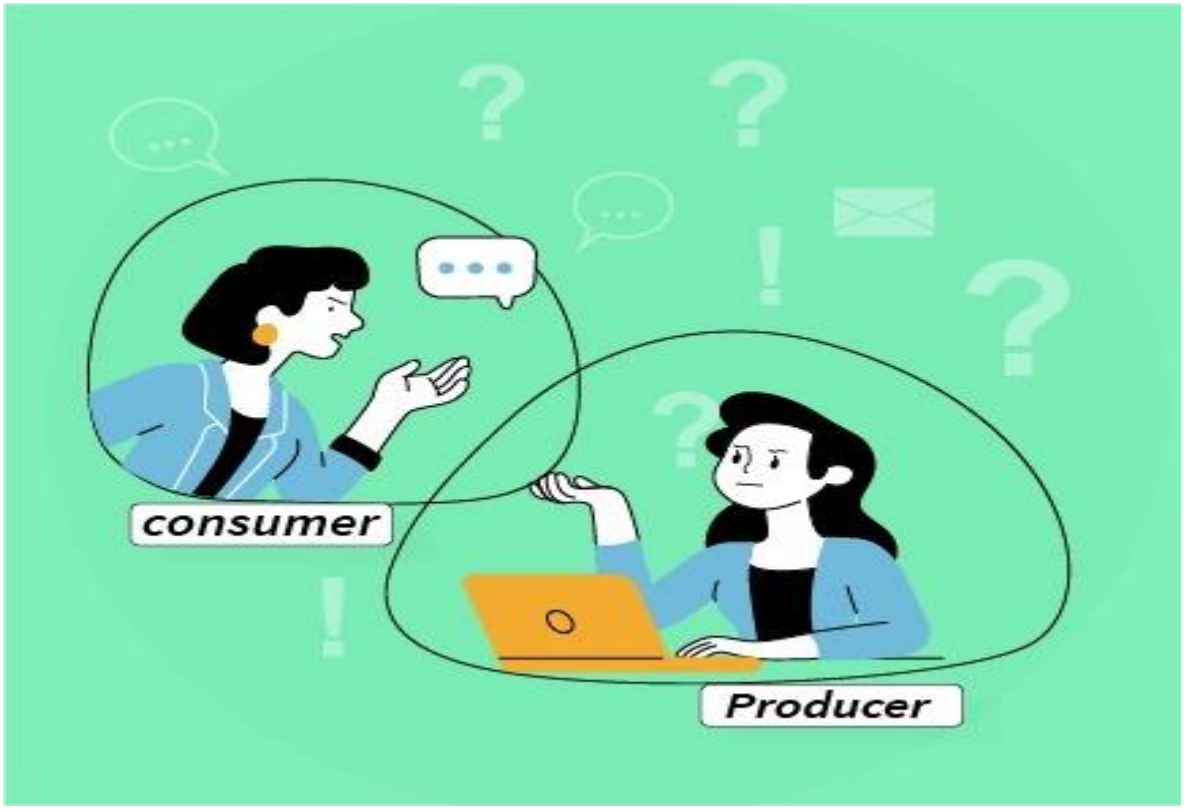
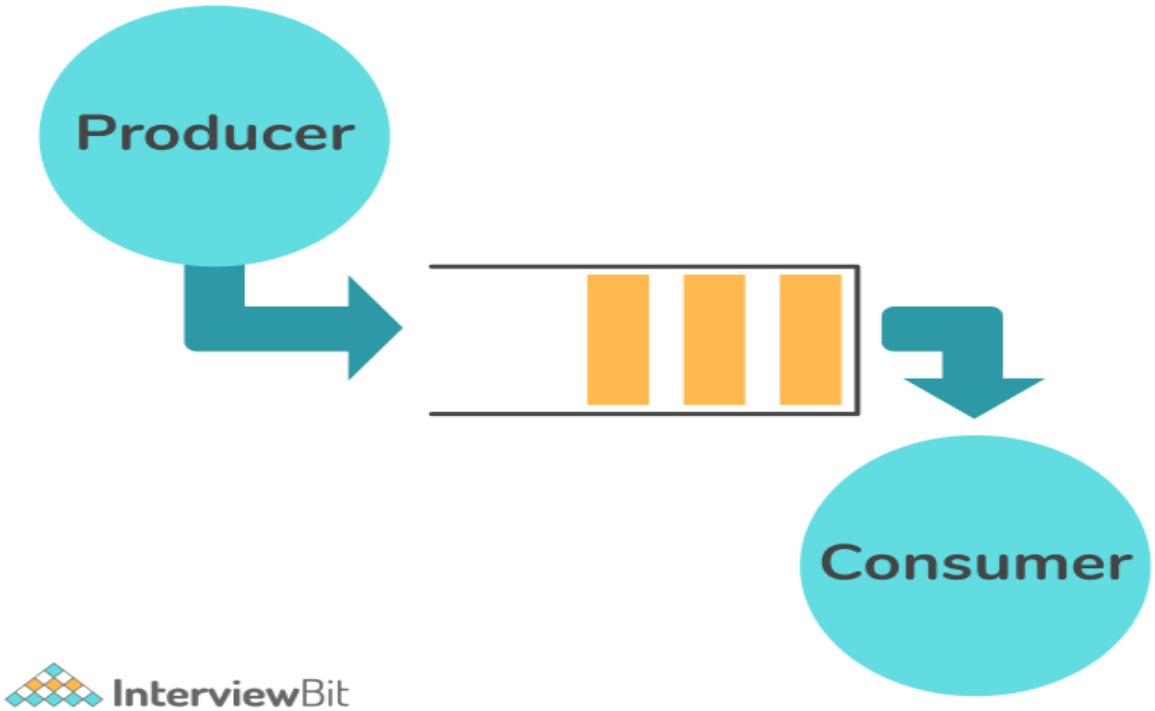
Producer Consumer problem :



Producer-Consumer problem is a classical synchronization problem in the operating system. With the presence of more than one process and limited resources in the system the synchronization problem arises. If one resource is shared between more than one process at the same time, then it can lead to data inconsistency. In the producer-consumer problem, the producer produces an item, and the consumer consumes the item produced by the producer.



Solution pseudocode:

Pseudocode using wait() and notify():

1- ***function consume****(sharedQ)*

2- intialize queue with size

2- initialize consumer sharedQ with sharedQ

3- ***end function***

4- ***function run****()*

5- ***while*** *condition is true*

6- try //to make project don't give error

7- intiallize variable name

8- if size of quee equal 0

9- then queue is empty

7- wait in loop

8- try //to make project don't give error

8- catch interrupted exception

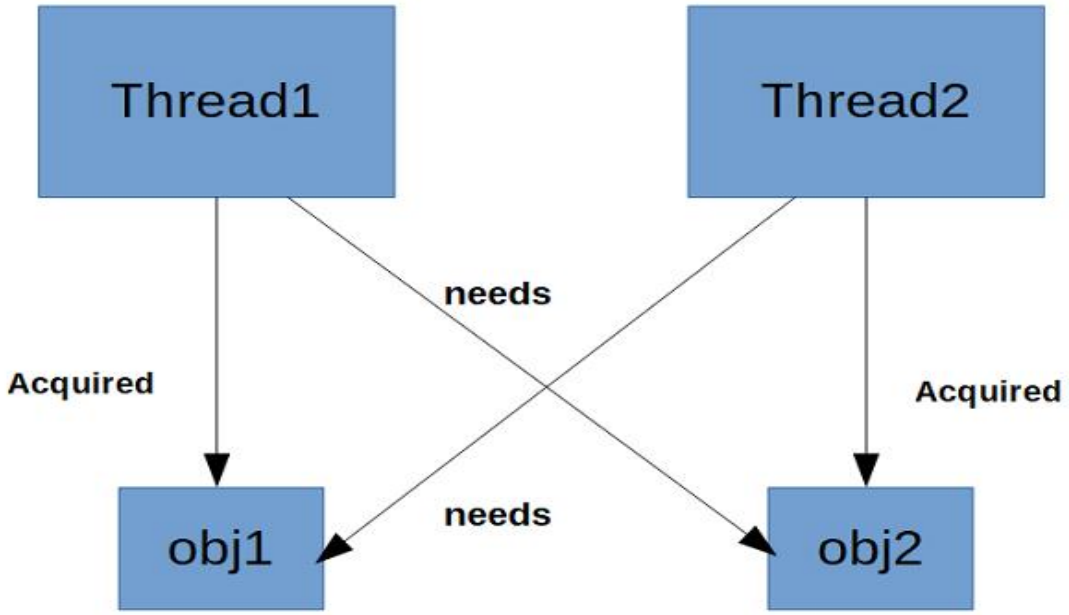
9- initialize variable number //to hold first index in queue

10- call function notify

11- if size of queue

*9- **end while***

Deadlock:



Deadlock occurs when each process holds a resource and wait for other resource held by any other process. Necessary conditions for deadlock to occur are Mutual Exclusion, Hold and Wait, No Preemption and Circular Wait. In this no process holding one resource and waiting for another get executed. For example, in the below diagram, thread 1 is holding obj1 and waiting for obj2 which is acquired by thread2, and thread2 is waiting for obj1.

They are three ways to handle deadlock:

1- **Deadlock prevention:** The possibility of deadlock is excluded before making requests, by eliminating one of the necessary conditions for deadlock.

2- **Deadlock avoidance:** Operating system runs an algorithm on requests to check for a safe state. Any request that may result in a deadlock is not granted.

3- **Deadlock detection & recovery:** OS detects deadlock by regularly checking the system state, and recovers to a safe state using recovery techniques.

If we remove notify() from either producer or consumer .

All process of consumer or producer keep waiting for other to complete and none get executed

Examble : consumer

```
public Consumer(LinkedList<Integer> sharedQ) {
    this.sharedQ = sharedQ;
}

public void run() {
    while(true) {
        String name = "tarek";
        synchronized(this.sharedQ) {
            while(this.sharedQ.size() == 0) {
                try {
```

```
                    System.out.println("Queue is empty, i don,t have
any orders from "
                            +name);
                    this.sharedQ.wait();
                } catch (Exception var6) {
                }
            }



            try {
                Thread.sleep(300L);
            } catch (Exception var5) {
            }

            int number = (Integer)this.sharedQ.poll();
            System.out.println("order " + number + " taken by " +
name);
           // this.sharedQ.notify();
            if (number == 10) {
                return;
            }
```

The output:

```
chef take  order 1 from tarek

order 1 taken by tarek

chef take  order 2 from tarek

chef take  order 3 from tarek

chef take  order 4 from tarek

Queue is full, can't take any orders from tarek

order 2 taken by tarek

order 3 taken by tarek

chef take  order 5 from tarek

order 4 taken by tarek
```

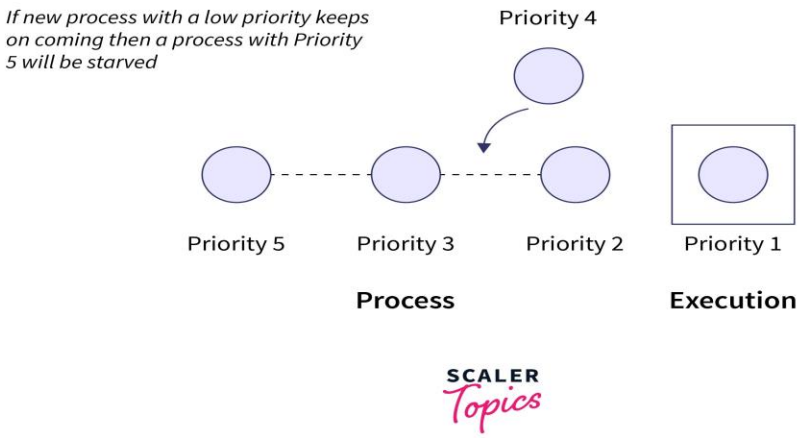
```
order 5 taken by tarek
chef take  order 6 from tarek
order 6 taken by tarek
Queue is empty, i don,t have any orders from tarek
chef take  order 7 from tarek
chef take  order 8 from tarek
chef take  order 9 from tarek
order 7 taken by tarek
chef take  order 10 from tarek
order 8 taken by tarek
order 9 taken by tarek
order 10 taken by tarek
```

as we see that the program stuck at waiting producer to produce but it cant because we removed notify() that can waking up the thread in waiting state

The solution:

Solution: USE notify()

Starvation:



Starvation *is the problem that occurs when low priority processes get jammed for an unspecified time as the high priority processes keep executing. A steady stream of higher-priority methods will stop a low-priority process from ever obtaining the processor.*

To solve starvation problem:

A possible solution to starvation is to use a scheduling algorithm with priority queue that also uses the aging technique. Aging is a technique of gradually increasing the priority of processes that wait in the system for a long time

Starvation:

If we remove wait() from either producer or consumer .

All process of consumer or producer keep executing and other processes are being blocked

Examble : producer

```
public Consumer(LinkedList<Integer> sharedQ) {
    this.sharedQ = sharedQ;
}
```

```
public void run() {
    while(true) {
        String name = "tarek";
        synchronized(this.sharedQ) {
            while(this.sharedQ.size() == 0) {
                try {
                    System.out.println("Queue is empty, i don,t have
any orders from "
                            +name);
                   // this.sharedQ.wait();
                } catch (Exception var6) {
                }
            }



            try {
                Thread.sleep(300L);
            } catch (Exception var5) {
            }

            int number = (Integer)this.sharedQ.poll();
            System.out.println("order " + number + " taken by " +
name);
            this.sharedQ.notify();
            if (number == 10) {
                return;
            }
```

The output:

```
chef take  order 1 from tarek

order 1 taken by tarek

chef take  order 2 from tarek

chef take  order 3 from tarek

chef take  order 4 from tarek

Queue is full, can't take any orders from tarek

order 2 taken by tarek
```

```
order 3 taken by tarek
chef take  order 5 from tarek
order 4 taken by tarek
order 5 taken by tarek
chef take  order 6 from tarek
order 6 taken by tarek
Queue is empty, i don,t have any orders from tarek

chef take  order 7 from tarek
chef take  order 8 from tarek
chef take  order 9 from tarek
order 7 taken by tarek
chef take  order 10 from tarek
order 8 taken by tarek
order 9 taken by tarek
order 10 taken by tarek
```

as we see that the producer keep executing and the consumer being blocked

because we removed wait() that make the thread wait until the notify() method invoked

The solution:

Solution: USE wait()

Real world application is restaurant

When a person want to eat from a restaurant he give his order to chef (producing process)and when the order finished he take his order (consuming process)