Solution pseudocode:

Producer pseudo code solution:

```
Runflag=true;
while (runflag)
        Generate message()
        Data will be generated
        While(dataQueue.is full())
                dataQueue.waitonfull()
       process will wait in full queue
        if (!runflag)
                Break
          dataQueue.add(data)
          producer will add data
```

consumer pseudocode solution:

```
Runflag=true;
while (runflag)
        While(dataQueue.is empty())
                process will wait in empty queue
        if (!runflag)
                Break
          dataQueue.remove(data)
          consumer will remove data
```

dataqueue pseudocode solution:

Isfull()

```
Size=queue.size()
If(size=maxsize)
        Return true
Else{
        Return false
```

Isempty()

```
response=queue.isempty()
If(response)
        Return true
Else{
        Return false
```

Waitonfull()

When the queue reaches the full state it makes the producer thread waits

Notifyallforfull()

Notifies the producer that is waiting on the full queue

Waitonempty()

When the queue reaches the empty state it makes the consumer thread waits

Notifyallforempty()

When it checks whether it still needs to continue or break then notifies the consumer waiting on an empty queue

Examples of Deadlock:

If we replace order of **notifyallforfull() and notifyallforempty()** It will prevent other producer or consumer to get in critical section and the actuall beginning sequence of the critical secction

How to solve deadlock:

We must verify the order of **the notifyallforfull() and notifyallforempty() commands** before deployment

Then we re check if the queue is empty or full to make the sequence begin with the producer

Any change in order may lead to **deadlock**

Examples of starvation:

if we did not make the lock on the critical section it may lead to the starvation problem as well as not make the locks atomic

How did solve starvation:

We did solve **starvation problem** by using a **queue** then making the locks atomic using synchronized blocks which in this case makes the critical section terminated by only one process then it notifies the next thread to go to its critical section

Explanation for real world application and how did apply the problem:

Factory

the factory product lines serves multiple version of the same product and if we imagined

the factory producing line as the producer class and the destined product version is the

consumer class we can replicate the factory parallel processes using threads and in this problem

we used 2 producers as the producing line and 3 consumers as the targeted versions of the product

so the 2 producers will start filling the line with the product using the produce() method then

it notifies the consumers using **notifyAllForEmpty()** method to start feeding the products to the

targeted destination.

```java
package com.mycompany.os_2;

import com.mycompany.os_2.data;
import com.mycompany.os_2.producer;
import com.mycompany.os_2.consumer;
import com.mycompany.os_2.data_Q;

public class main {
    public static void main (String args[]) throws InterruptedException{
        data_Q dataQueue = new data_Q(4);

            producer producing_line = new producer(dataQueue);
            for(int i = 0; i < 2; i++) {
                Thread producerThread = new Thread(producing_line);
                producerThread.start();
            }

            consumer targeted_product = new consumer(dataQueue);
            for(int i = 0; i < 4; i++) {
                Thread consumerThread = new Thread(targeted_product);
                consumerThread.start();
            }

    }
}
```