***Pseudocode***

Producer:

L: if (s - r) mod k = b then goto L fi;

put message in buffer;

s := (s + 1) mod k;

goto L;

Consumer:

L: if (s - r) mod k = 0 then goto L fi;

take message from buffer;

r := (r + 1) mod k;

goto L;

***Examples of starvation***

Consider there is 2 producer threads and one consumer thread.

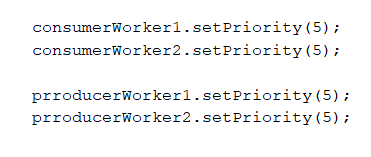
Suppose goods is full.

Two producer thread goes to wait state because goods is full.

Consumer thread takes element from goods and notifyAll so one of the producer thread adds element and comes out and another producer thread remains in wait state and again another producer threads add elements and coming out.

So if you observe there are chances that one thread can be in wait state always

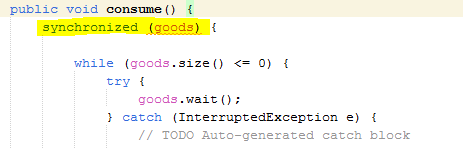
We solve starvation by setting all of our threads to the same priority

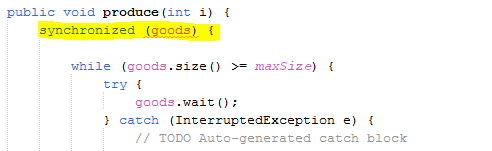


this makes it so no one thread always gets chosen by the scheduler, and considering the uniform nature of our threads in the consumer producer problem, no one thread will take significantly more time than the others and starve them

***Example of Deadlocks***

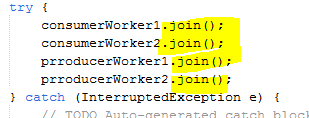
* If two threads are waiting for each other forever such type of infinite waiting is called deadlock in java. Synchronized keyword is the only reason for deadlock situation hence while using synchronized keyword we have to take special care.





We can avoid Deadlock situation in the following ways:

* Using Thread.join() Method: We can get a deadlock if two threads are waiting for each other to finish indefinitely using thread join. Then our thread has to wait for another thread to finish, it is always best to use Thread.join() method with the maximum time you want to wait for the thread to finish.



**Real World Application**

**Consider a bakery that has an online ordering app, the bakery produces 5 cakes every 10 seconds, the number of customers ordering every 10 seconds is random, and the bakery can only store 20 cakes at a time.**

**We apply the Bounded Buffer problem to ensure that**

**1. The bakers will never bake more than 20 cakes if there’s no demand**

**2. Customers ordering online will never order the same piece of cake**

**3. An order can’t be fulfilled until there’s a cake available**

**The App simulates the bakers and other online customers using threads that run without user input, and creates an runs a separate thread for the customer when they input order confirmation, which informs the user when the order is done and how much time it took to make.**