Bounded buffer solution

Pseudocode:

Producer:

```
While(true)
empty.acquire()                  //decrease the number of empty slots
s.acquire()                      // get semaphore (enter critical section)
buffer[next_in] = copy(item)     // put item in buffer
next_in=next_in +1 % N
s.release()                      //release semaphore (end critical section)
full.release()                   //increase the number of full slots
end while
```

Consumer:

```
while(true)
full.acquire()                   //decrease the number of full slots
s.acquire()                      //get semaphore (enter critical section)
data=copy(buffer[next_out])      // remove item
next_out=next_out + 1 % N
s.release()                      //release semaphore (end critical section)
empty.release()                  //increase the number of empty slots
end while
```

Deadlock

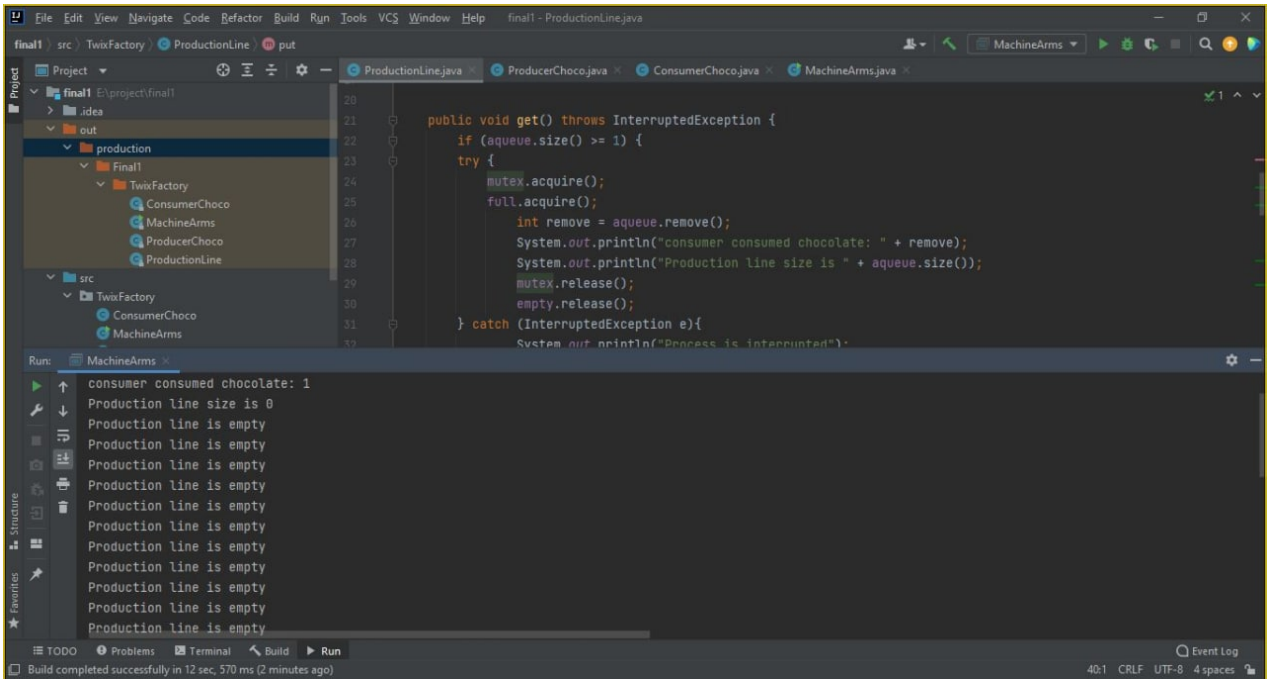
Deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.

Example:

When Producer and consumer enter the critical section and access same slot in the same time. In this situation more than one thread is blocked because it is holding the same slot in buffer.

Producer:	**Consumer:**

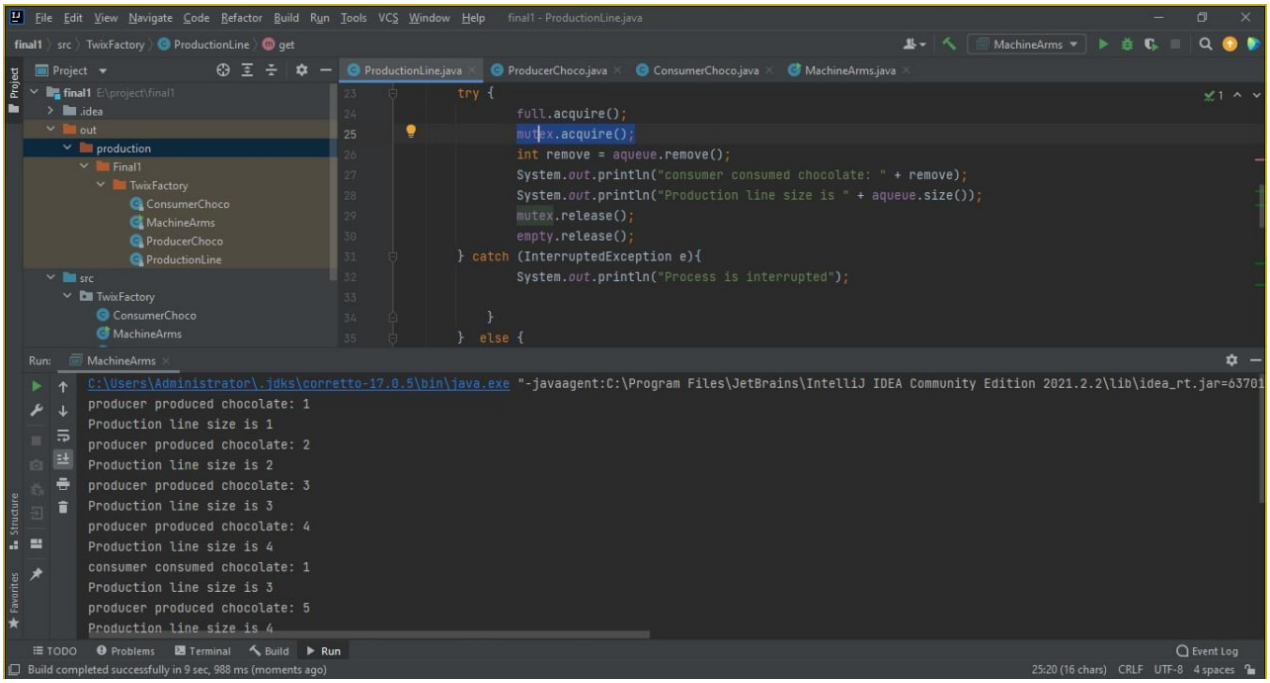
s.acquire()
empty.acquire()
// put item in buffer
buffer[next_in] = copy(item)
next_in=next_in +1 % N
full.release()
s.release()
	s.acquire()
	full.acquire()
	// remove item
	data=copy(buffer[next_out])
	next_out=next_out + 1 % N
	empty.release()
	s.release()



Solution:

The solution is **Semaphore** as semaphore limits the amount of concurrent work that can be completed at the same time. Where s.acquire() get semaphore and prevent any thread to access the critical section and s.release() release the semaphore.

Producer:	**Consumer:**
empty.acquire()	full.acquire()
s.acquire()	s.acquire()
buffer[next_in] = copy(item)	// remove item
next_in=next_in +1 % N	data=copy(buffer[next_out])
s.release()	next_out=next_out + 1 % N
full.release()	s.release()
	empty.release()



Starvation

Starvation is the problem that occurs when high priority processes keep executing and low priority processes get blocked for indefinite time.

Examples:

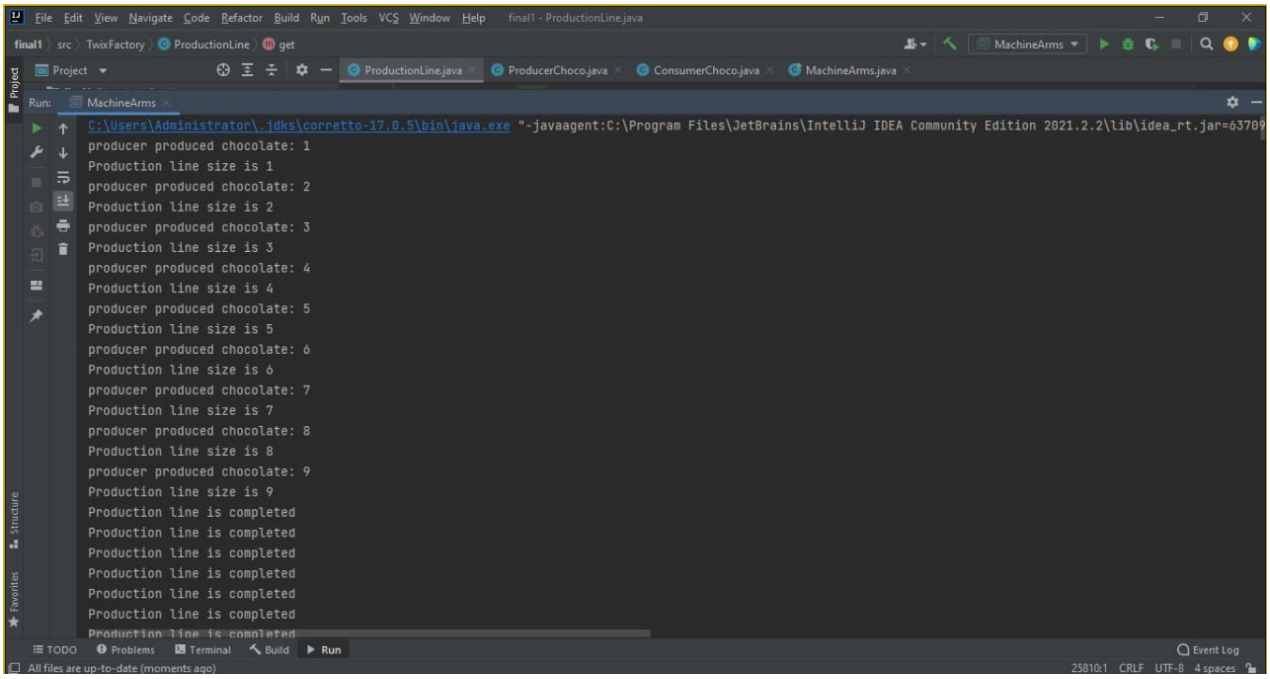
1. Consumer does not return the consumed buffer to empty buffer queue and producer wait as buffer is full.

2. Producer does not return the produced buffer to ready buffer queue and Consumer wait as buffer is empty.

Producer:	**Consumer:**

empty.acquire()
s.acquire()
// put item in buffer
buffer[next_in] = copy(item)
	full.acquire()
	s.acquire()
	// remove item
	data=copy(buffer[next_out])

next_in=next_in +1 % N s.release()	next_out=next_out + 1 % N s.release()



Solution:

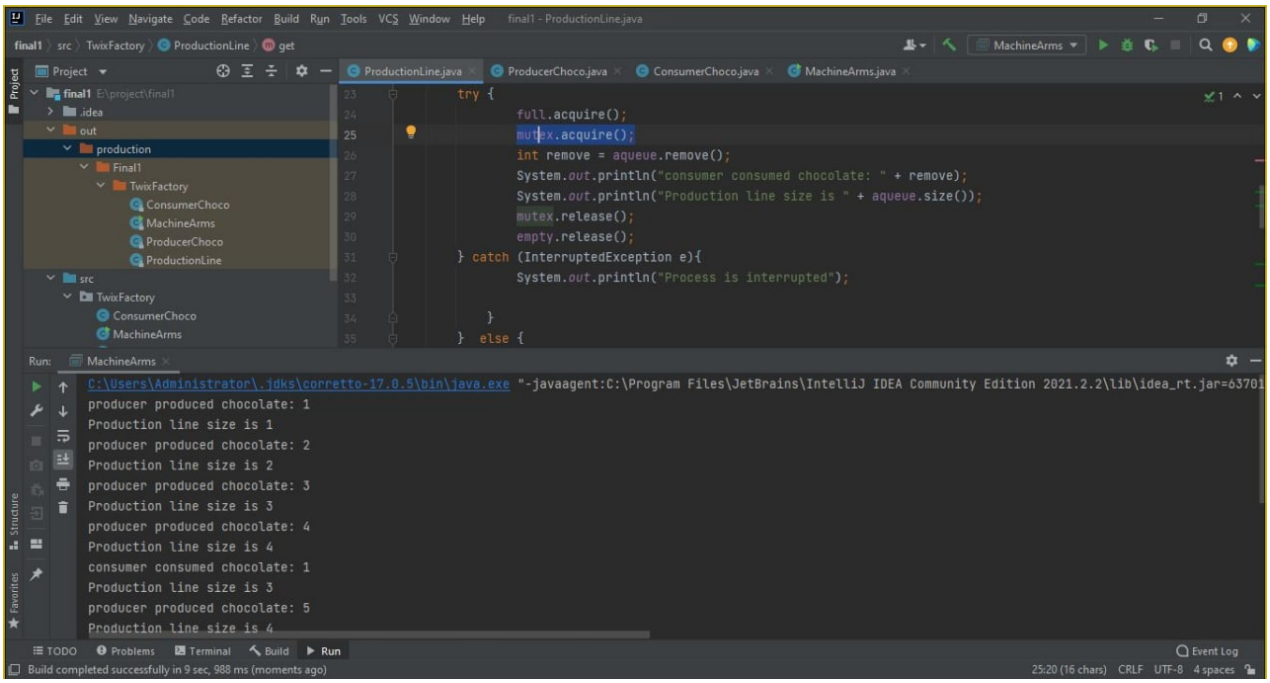
1.Producer after producing item immediately moving buffer to ready buffer queue using full.release() .

2. Consumer after consuming item immediately moving buffer to empty buffer queue using empty.release().

By this arrangement prevent the Starvation.

Producer:	**Consumer:**
empty.acquire() s.acquire()	full.acquire() s.acquire()

// put item in buffer
buffer[next_in] = copy(item)
next_in=next_in +1 % N
s.release()
full.release()
	// remove item
	data=copy(buffer[next_out])
	next_out=next_out + 1 % N
	s.release()
	empty.release()



Explanation for real world application

How application work:

- Twix factory that produces packets of chocolate, each packet has two pieces of chocolate.
- A machine(producer) produces chocolate pieces into a production line (buffer) that has a fixed size of slots n.
- The production line has 2 mechanic arms (consumer) working on it that takes 1 chocolate piece at a time from the production line and puts it into a pack.
- hence, each pack needs 2 machines to put chocolate in it, 1 machine to put in it twice.. so it has a total number of 2 chocolate pieces.

❖ A problem of 2 arms trying to put a piece into a packet that already has 1 piece might occur, Which the program will solve with Bounded Buffer solution.