: **Synchronised method** :- way for a client for solving sync issue.

=> if you make any method as synchronised, only 1 thread can execute / enter the method at a time, basically method is treated as a CS.

```
Count {
private value;
synchronised incValue( ); ←
synchronised decValue( ); ←
             getValue( ); ←
}
```

```
Count c1 = new Count( );
Count c2 = new Count( );
```

T1	T2	will it run parallely
c1.incValue( );	c1.incValue( );	✗
c1.incValue( );	c1.decValue( );	✗
c1.incValue( );	c1.getValue( );	✓
c1.incValue( );	c2.incValue( );	✓

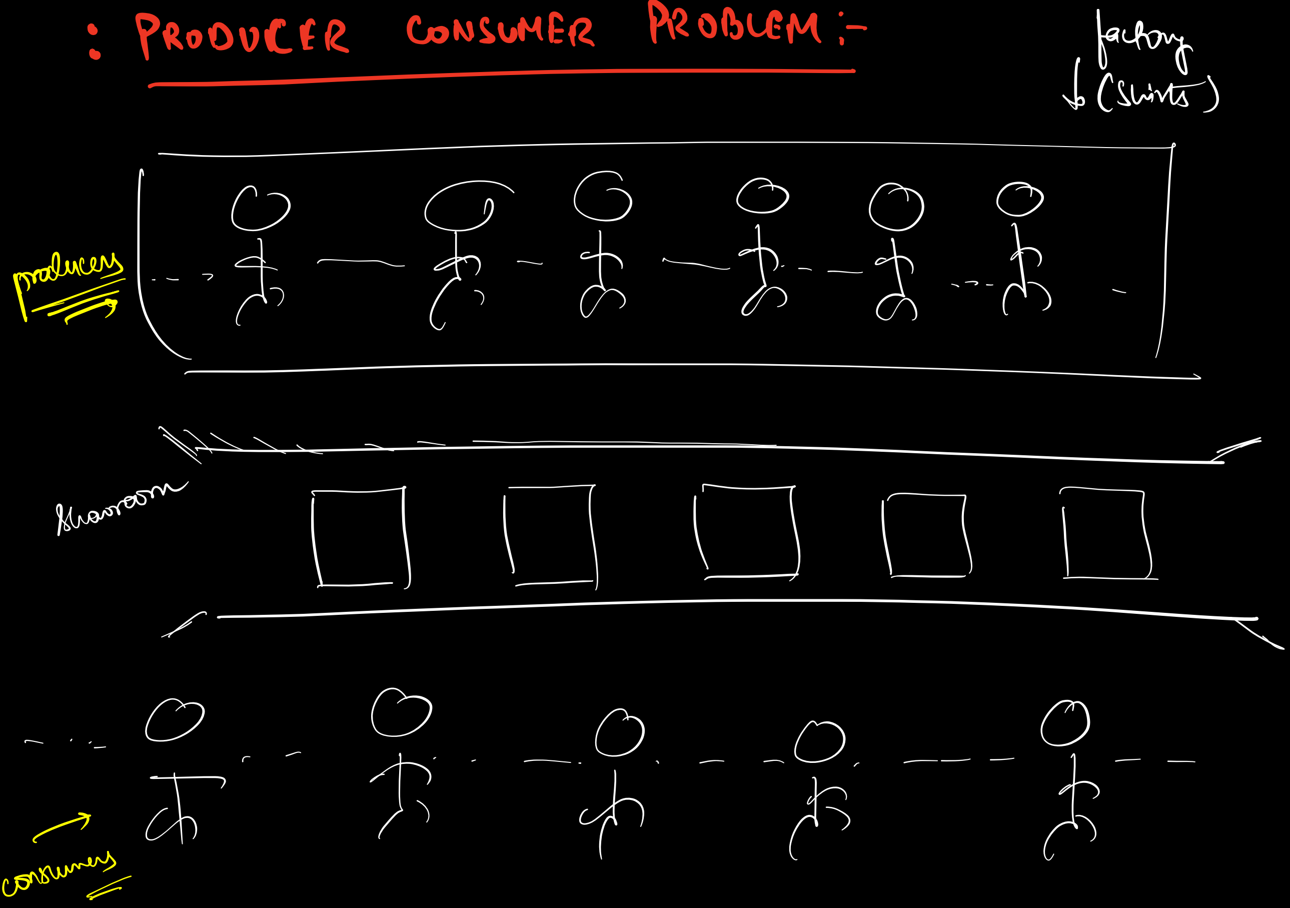
```
public void incValue();

public synchronized void incValue();
```

: SEMAPHORES :-

: How many threads are allowed in CS at a time?
⇒ 1

: Can there be a scenario, where we can allow multiple threads in CS? ⇒

: PRODUCER CONSUMER PROBLEM :-



=> A consumer can only enter the showroom if there is a shirt available for them.

=> A producer can only enter the showroom if there is space available for them inside the store.

no. of producers that can enter the store
= no. of empty spaces in the store

no. of consumers that can enter the store
= no. of filled spaces in the store

$$\left.\begin{array}{c}\text{no. of producers that can enter the store} \\ + \\ \text{no. of consumers that can enter the store}\end{array}\right\} \Rightarrow \text{size of the } \underline{\text{showroom}}$$

=> Queue<Shirt> store;

=> int maxSize; $\longrightarrow$ capacity of the $\underline{\underline{\text{store}}}$

< [ ] [ ] [ ] [ ] [ ] >

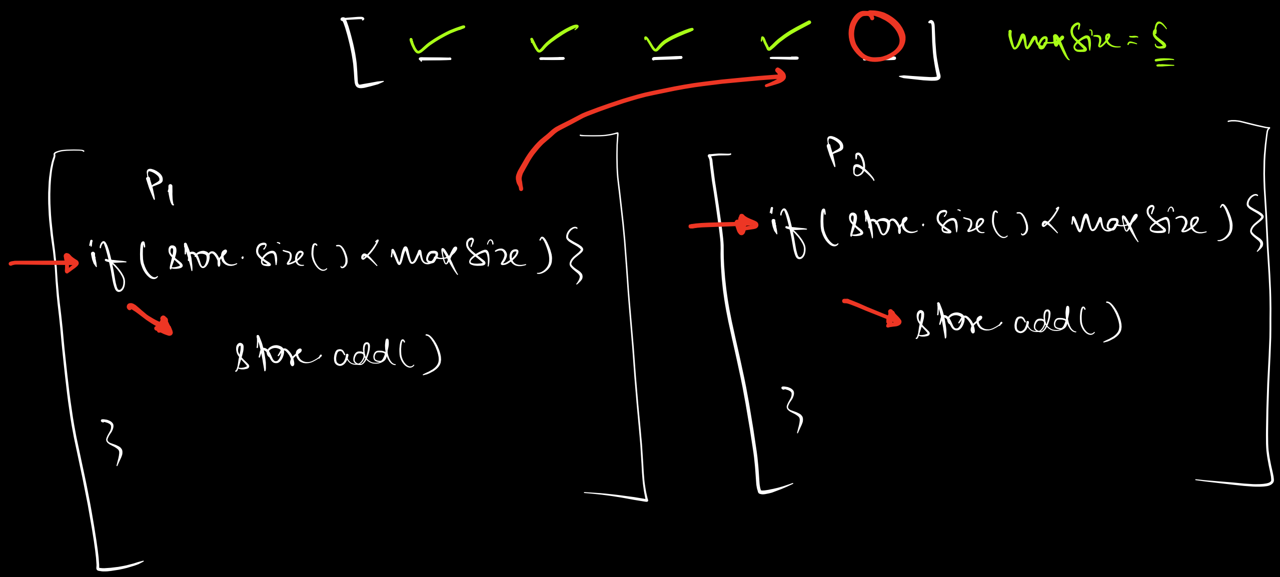
Producer

```
if( store.size() < maxSize ){
        store.add( new shirt())
}
```

Consumer

```
if( store.size() >0){
        store.remove();
}
```

[if we run with 100's of producers and consumers parallely]



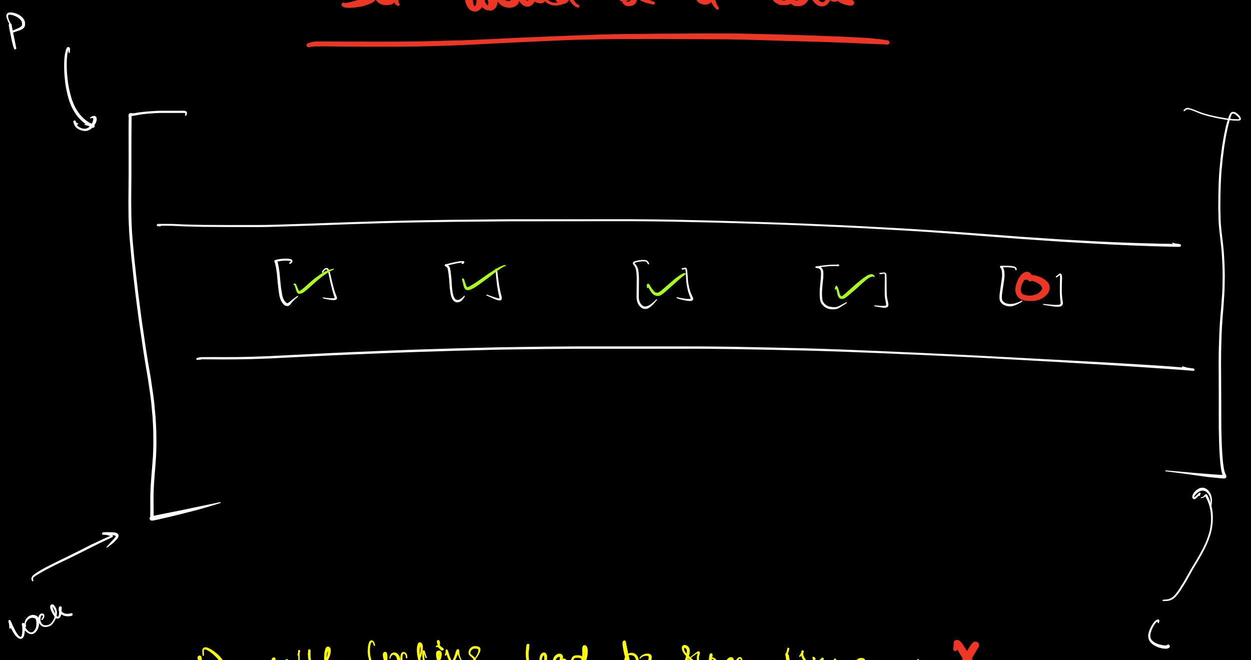
P1

```
if( store.size() < maxSize ){
        store.add()
}
```

P2

```
if( store.size() < maxSize ){
        store.add()
}
```

→ SQ



i) will locking lead to sync issue ⇒ ✗

ii) will it be fast enough ⇒ ✗

⇒ SQ^n ⇒ Semaphore

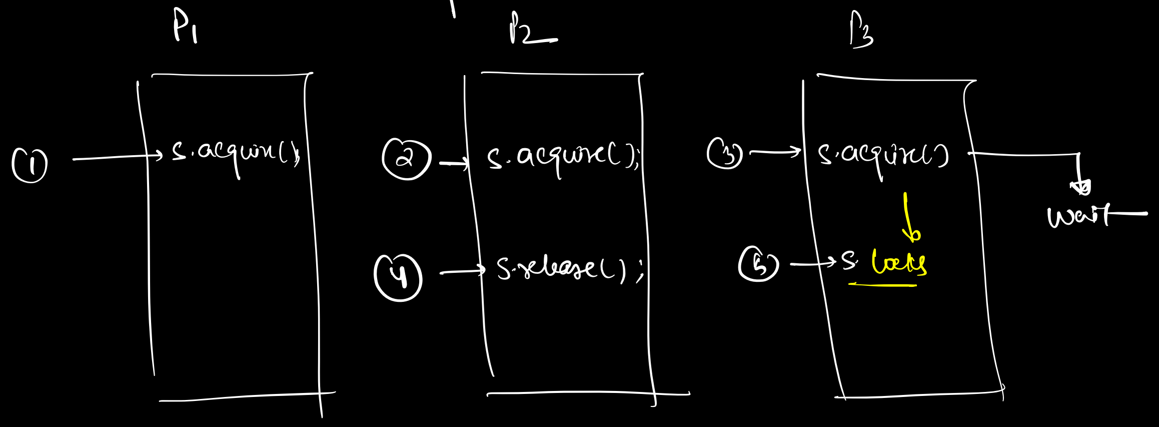
Semaphore s = new Semaphore(5);

s.acquire(); // taking a lock

⇒ you will only be able to take a lock (acquire) if no. of threads with lock < N (5)

s.release(); // unlocking

s = new Semaphore(~~2~~); ~~1~~ 0



If we have a semaphore, semaphore(x) =>

i) acquire() → x = x - 1
(only when x > 0)

ii) release() => x = x + 1

```
Semaphore semProducer = new Semaphore( storesize );

Semaphore semaConsumer = new Semaphore( 0 );
```

=> Producer:

```
run() {
    while( true ){
        semaProducer. acquire( );
        store.add(new Shirt( ));
        semaConsumer.release();
}
```

⇒ <u>Consumer</u>

```
run() {
        while( true) {
                semaConsumer.acquire();
                store.remove();
                semaProducer.release();
        }
}
```

Semaphore(1) = mutex

14/11 → session

15/11 → session

17/11 → no session

19/11 → normal Schedule