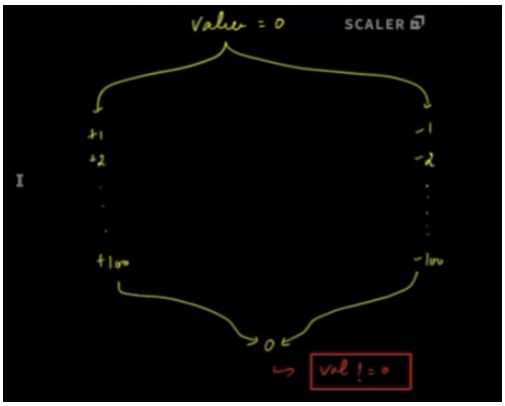
Synchronization 1

Synchronization Problem

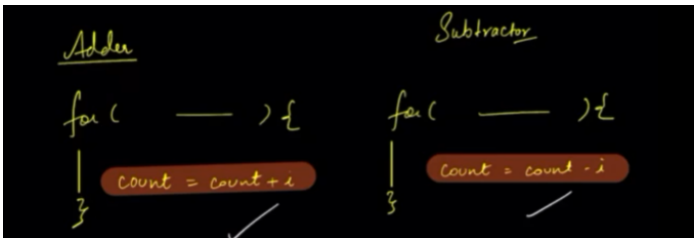
When does the Synchronization Problem happen ?

-> Happens because of Synchronization Issues.



Why ?

- **Critical Section:** Part of the code where we are working on shared data.



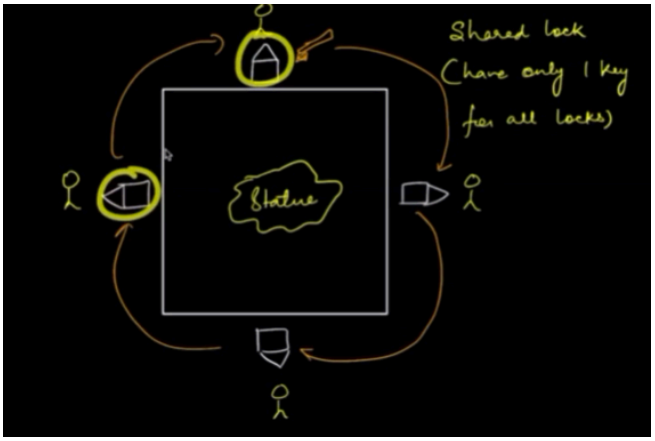
- **Race Condition:** Two threads kind of Race to complete the task.
  I.e When more than one thread tries to enter the critical section at the same time.

- **Pre Emptiness:** When we move from one task to another task, Context switching happens. In this case the last task can be in a partially completed state.

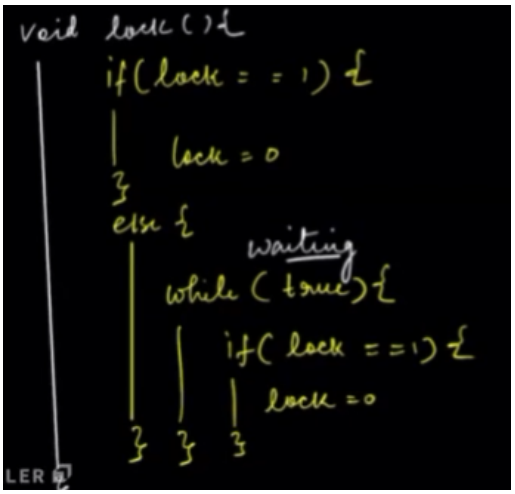
Solution: We have to allow only one thread to enter the critical section at a point of time.

Mutex / Mutual Exclusion

- Allows only one thread to enter the critical section at any point of time.



- But what if lock is never releasing the resource. At the end that process goes into the deadlock state and other process keeps on waiting

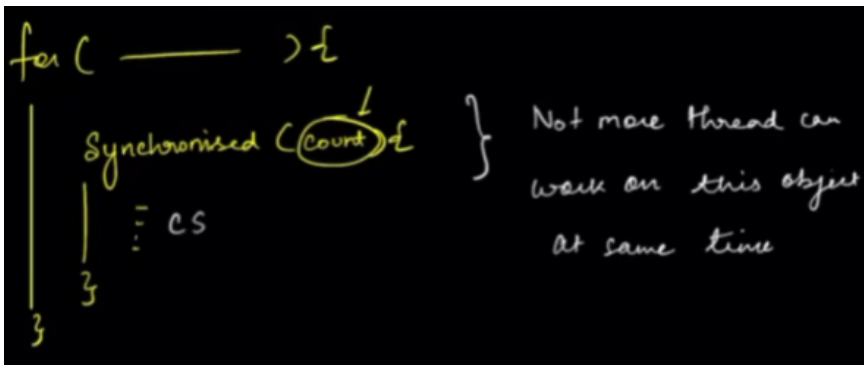


Solution:

So another way to handle this situation is using Synchronization.

Synchronized And Sync Methods

- In java, every object has an implicit lock.
  Ex:

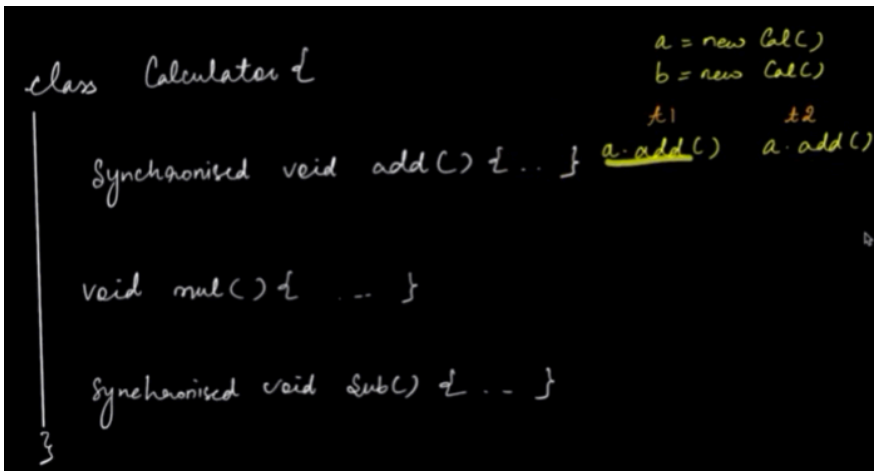


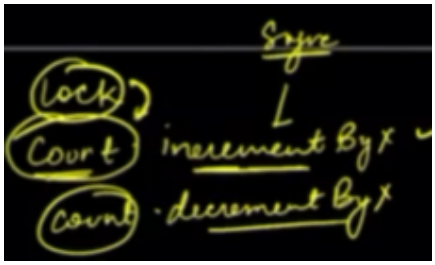
- Simply use **synchronized(_variable)** to add the synchronization in the critical section.

Sync Method

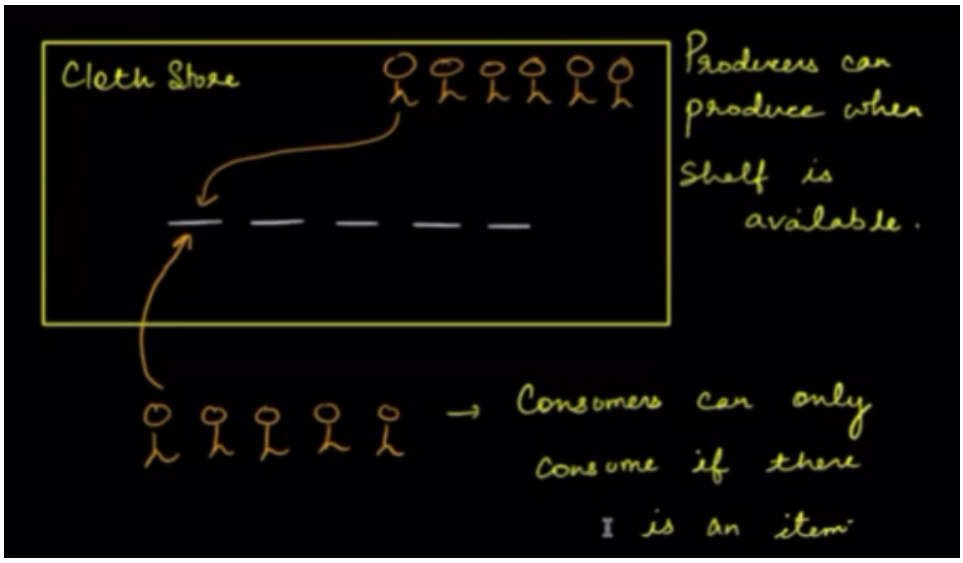
- Lock is taken on the object which calls the sync method.

  Example of Sync Method:





Producer Consumer Problem



Structure Of Store

- maxSize (No of shelf)
- List<Items>
- Operations
    - AddItems()
    - RemoveItems()

Producer add()

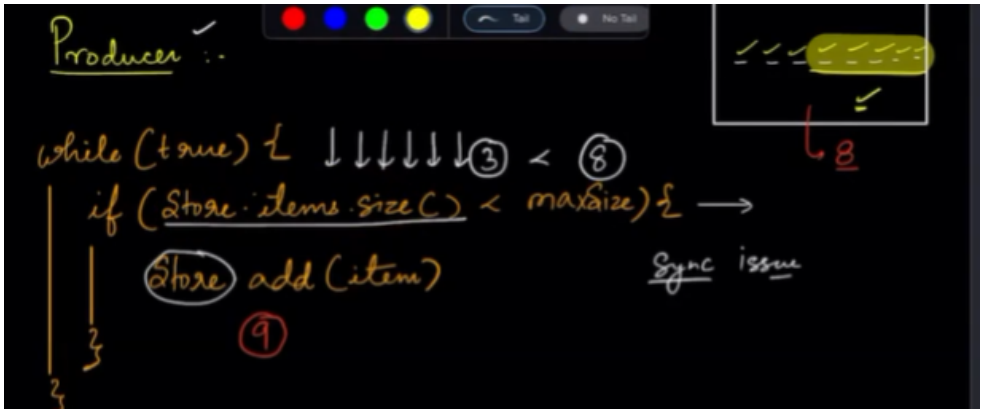
```
while(true){
   if(store.items.size() < maxSize()){
        store.add()   // Critical Section
   }
}
```

Consumer remove()

```
while(true){
   if(store.items.size() > 0){
        store.remove();   // Critical Section
   }
}
```

Problem:

- Let's say there is a max size 8. And there are 6 producers who want to produce. All at the same time.
- Let's say three shelves are filled, 5 are empty. Since 5 are empty all 6 producers will be allowed to enter the critical section.
- But after 5th item for 6th item issue will occur.



So the last producer should not be allowed.

- Similarly the same issue can happen with consumers.
- Let's say 5 Selves are there, only two selves have the product. 3 consumers enter at the same time. So for one consumer there will be a synchronization issue.

Questions

Q1. Introduction to Synchronization - Mutex, synchronized keyword, Atomic Data Types, Concurrent DS -1

Which of the following statements is true about the synchronized keyword?

It can only be applied to instance methods
It can be applied to both instance methods and static methods
It can only be applied to static methods
It cannot be used in a multithreaded application

Q2. Introduction to Synchronization - Mutex, synchronized keyword, Atomic Data Types, Concurrent DS -3

Which of the following is true regarding the differences between Hashtable and ConcurrentHashMap in Java?

HashTable allows multiple threads to access different parts of the map at the same time, while ConcurrentHashMap does not.
ConcurrentHashMap is synchronized, while Hashtable is not.
Hashtable is synchronized, ConcurrentHashMap Hashtable is not.
ConcurrentHashMap allows multiple threads to access different parts of the map at the same time, while Hashtable does not.

Q3. Introduction to Synchronization - Mutex, synchronized keyword, Atomic Data Types, Concurrent DS -5
Solved
What is the name of the algorithm used by Atomic Data types for thread-safety?

Mark and Sweep
Binary Search
Compare and Swap
None of the above

Q4. Introduction to Synchronization - Mutex, synchronized keyword, Atomic Data Types, Concurrent DS -7

Solved

Consider the following code

public class A {

```
synchronized static void fun1(){

}

synchronized  void fun2(){

}

void fun3(){

}
```

}

And the following code

public class Client {

```
public static void main(String[] args) {

    A obj1 = new A();

    A obj2 = new A();

}
```

}

Which of the following can't execute concurrently on two separate threads

obj1.fun1 and obj1.fun2
obj1.fun1 and obj2.fun1
obj2.fun2 and obj2.fun3
obj1.fun2 and obj2.fun2