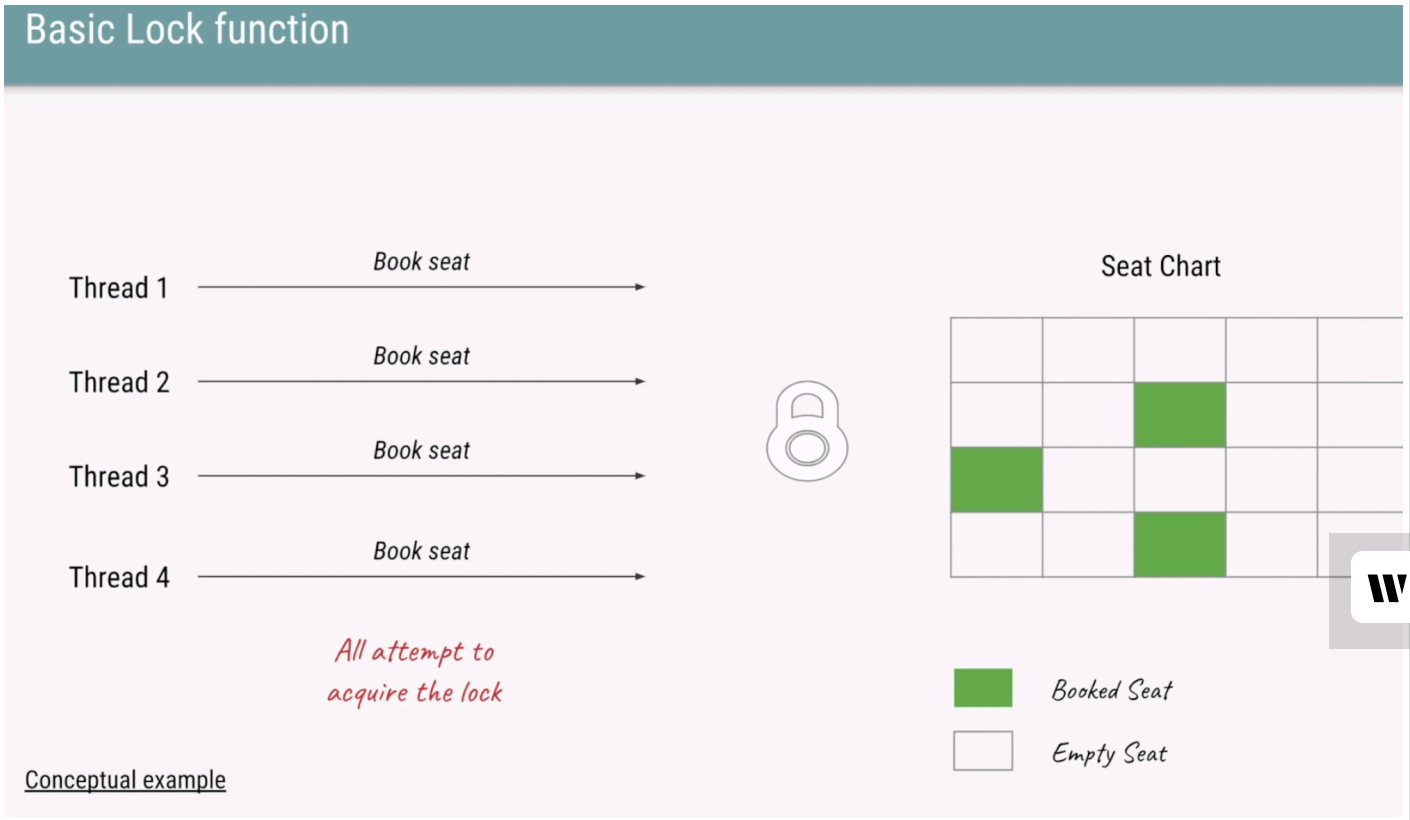
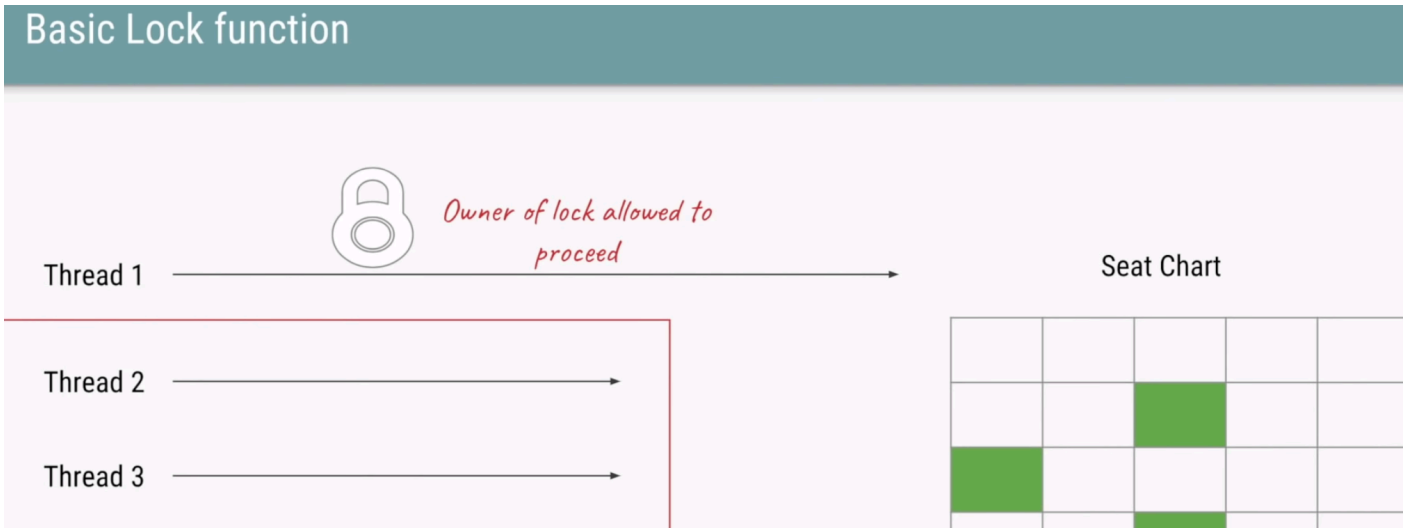
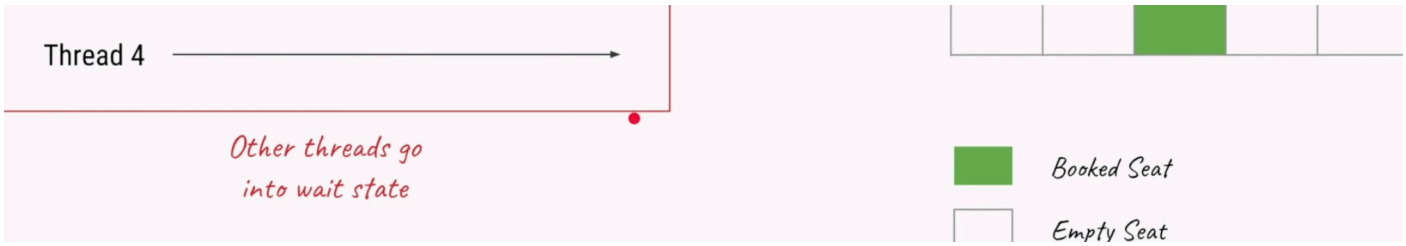
Multithreading: Locks: ReentrantLock class



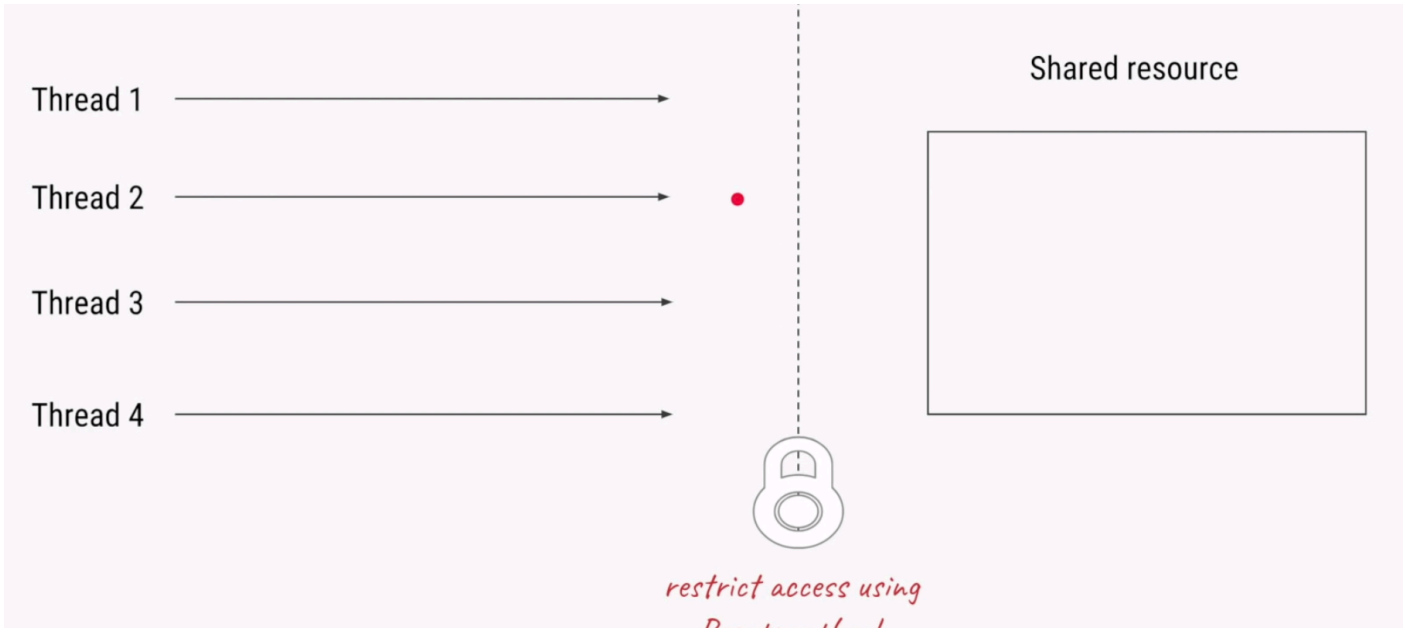
→ Here we allow to only one thread at a time to get a lock and enter in window.





→ Once a thread release the lock then waiting thread could take the lock.

Lock allowed restrict the shared resource such that only one thread can access the resource.



Code

```
private static ReentrantLock lock = new ReentrantLock();

private static void accessResource() {

    lock.lock();

    // access the resource
```

```
    lock.unlock();
}

public static void main(String[] args) {

    Thread t1 = new Thread(() -> accessResource()); t1.start();
    Thread t2 = new Thread(() -> accessResource()); t2.start();
    Thread t3 = new Thread(() -> accessResource()); t3.start();
    Thread t4 = new Thread(() -> accessResource()); t4.start();
```

Same as synchronized but with more features

```
private void accessResource() {

    synchronized (this) {          <-- lock.lock()
        // access the resource
    }                              <-- lock.unlock()
}

private void accessConcurrently() {
    Thread t1 = new Thread(() -> accessResource()); t1.start();
    Thread t2 = new Thread(() -> accessResource()); t2.start();
    Thread t3 = new Thread(() -> accessResource()); t3.start();
    Thread t4 = new Thread(() -> accessResource()); t4.start();
}
```

- Locks are explicit
- Locks allow locking/unlocking in any scopes and in any order
- Ability to tryLock, and tryLock(timeout)

Difference b/w synchronized and ReentrantLock is:

1. Synchronised is implicit but lock is explicit
2. We can aquire a lock in one method and release on other method but synchronized we have all this in one block only
3. Lock provided many advanced facility as well

Better safe than sorry

```
private static ReentrantLock lock = new ReentrantLock();
```

```
private static void accessResource() {

    lock.lock();
    try {
        // access the resource
    } finally {
        lock.unlock();
    }
}

public static void main(String[] args) {

    Thread t1 = new Thread(() -> accessResource()); t1.start();
    Thread t2 = new Thread(() -> accessResource()); t2.start();
    Thread t3 = new Thread(() -> accessResource()); t3.start();
    Thread t4 = new Thread(() -> accessResource()); t4.start();
```

Why this is called ReentrantLock ?

Bcoz it allow us to lock multipal times without calling unlock.

Why the name - Reentrant

```
private static void accessResource() {

    lock.lock();
    lock.lock();

    int number = lock.getHoldCount();

    lock.unlock();
    lock.unlock();
}
```

Number of times lock called..
Without calling unlock

Reentrant - More practical example

```
private static ReentrantLock lock = new ReentrantLock();

private static void accessResource() {

    lock.lock();
```

In recursive call lock count is increased as trying to access same resource which he has already access so only getHeldCount is increased.