Your bug is a classic concurrency problem in your Mailbox class that causes a deadlock.

Here’s a breakdown of what's happening and how to fix it.

💥 Bug 1: Deadlock in Mailbox.java

The program will encounter a deadlock due to an issue within Mailbox.java.

You are using **two** different locking mechanisms at the same time:

1. The synchronized keyword on the put() and get() methods.
2. The Semaphore mutex inside those methods.

This is redundant and causes a deadlock. Here is the scenario:

1. Producers fill the mailbox (capacity 5).
2. The emptySlots semaphore now has 0 permits.
3. A 6th producer (let's call it Producer-A) calls put().
4. Because put() is synchronized, Producer-A **acquires the object lock** for the Mailbox instance.
5. Inside the method, it calls emptySlots.acquire(). This call **blocks** (waits) because there are no empty slots.
6. **Crucially, Producer-A is now blocked *while still holding the Mailbox object lock*.**
7. A Consumer (which is now working after Fix 1) tries to call get().
8. Because get() is *also* synchronized, it must wait to **acquire the object lock** for the Mailbox instance.
9. But it can't! Producer-A is holding that lock and will never release it because it's stuck waiting for emptySlots.acquire().

This is a **classic deadlock**. The producer has the lock and is waiting for the semaphore, while the consumer is waiting for the lock so it can (eventually) release the semaphore.

✅ The Fix for Bug 2

You only need one locking mechanism for the critical section (the part where you item.add() and item.remove()). Your mutex semaphore is already doing this job correctly. You should **remove the synchronized keyword**.

Change the method signatures in Mailbox.java:

```java
// FROM:
public synchronized void put(int value){
// TO:
public void put(int value){
```

```java
// FROM:
public synchronized int get(){
// TO:
public int get(){
```

Your mutex.acquire() and mutex.release() calls will correctly protect the item list all by themselves.