Pseudocode and solution details:

```
1- Semaphore serviceQueue = new Semaphore(1)  //to put readers and writers in queues and prevent starvation
2-Semaphore rmutex = new Semaphore(1)  //to protect the read counter from modifying its value by more than one user
3- Semaphore resource = new Semaphore(1)   //the readers or writer
4- int readCount = 0        //the read count with default val = 0
```

- Reading:

```
    Try{
    //Acquire Section
    acquire serviceQueue     // wait in queue to be serviced
    acquire rmutex
     readCount++
    if (readCount == 1) {
    //if the second, third, fourth,….,n threads are read so they will go and read without acquiring
                acquire resource   // request exclusive access to resource
                         }

    release serviceQueue     // let the next thread be serviced and start reading
    release rmutex

    //Reading section
   Sleep thread for 1500 ms
```

```
//after 1500 ms the thread will have finished
 reading process


 //Releasing section
acquire rmutex
readCount--

          if(readCount == 0) {
            release resource   //ensure that there is
           no threads want to read
          }
          release rmutex
 }
 catch error
```

- Writing:

```
 Try {
 acquire serviceQueue       // wait in queue to be
 serviced
  acquire resource          // request exclusive
 access to resource
  release serviceQueue    // let the next thread be
 serviced and start writing

 //Writing section
  Sleep thread for 2500 ms
  //after 2500 ms the thread will have finished
 writing process
 release resource
```

```
    }
    catch error
```

2- deadlock Example:

When 2 threads want to book tickets each want to book 2 tickets and the number of remaining tickets on the app is 2 tickets
thread1 booked one ticket and while it’s going to book the second one, thread2 interrupts the process and booked one ticket.
So deadlock happens and each thread will wait for the second ticket to reserve.

Solution on it:

use the acquire and release to solve this problem so when thread1 start booking, thread2 can’t interrupt it and waits in queue until thread1 finishes booking.

3-starvation example:

writer starvation:

1-Semaphore rmutex = new Semaphore(1)
2- Semaphore resource = new Semaphore(1)
3- int readCount = 0

- Reading:

```
    Try{
    acquire rmutex
     readCount++
    if (readCount == 1) {
        acquire resource
                            }
     release rmutex

    Sleep thread for 1500 ms
    acquire rmutex
    readCount--

               if(readCount == 0) {
                 release resource
               }
               release rmutex
     }
catch error
```

- Writing:

```
Try {
acquire resource
Sleep thread for 2500 ms
release resource
}
catch error
```

//The priority in this example is to reader where if there is a reader reads then writer waits in queue and other readers came after writer they will enter and read so writer will wait until all readers finish

Reader starvation:

```
private static final int queueNumber  = 0

  Semaphore rmutex = new Semaphore(1)
  Semaphore wmutex = new Semaphore(1)
  Semaphore resource = new Semaphore(1)
  Semaphore readTry = new Semaphore(1)
  int readCount = 0
  int writecount = 0
```

- Reading:

```
Try{
        acquire readTry
      acquire rmutex
      readCount++
      if (readCount == 1) {
         acquire resource
      }
      release rmutex
      release readTry
      Sleep thread for 1500 ms
 acquire rmutex
 readCount--
if(readCount == 0) {
   release resource
}
release rmutex
}
catch error
```

- Writing

```
acquire wmutex
writecount++
if (writecount == 1)
acquire readTry
release wmutex
acquire resource
```

```
            Sleep thread for 2500 ms
            release resource
            acquire wmutex
            writecount--
                if (writecount == 0)
                    release readTry
                release wmutex
              }
        Catch error
        //The priority in this example is to writer
        where if there is a writer writes then reader waits in
        queue and other writers came after reader, they will
        enter each in its turn, so many readers will wait until
        writer finishes
```

To solve this problem we need to add serviceQueue semaphore and acquire it as shown in p1

The semaphore is Used to put readers and writers in queue. Where only the first reader need to be acquired if the rest are readers. They will read without waiting in the queue (Multi-reading is allowed).
And the writer will wait until all readers finishes reading

If the writer acquired first then no other writers or readers can go and they all wait until the reader finishes