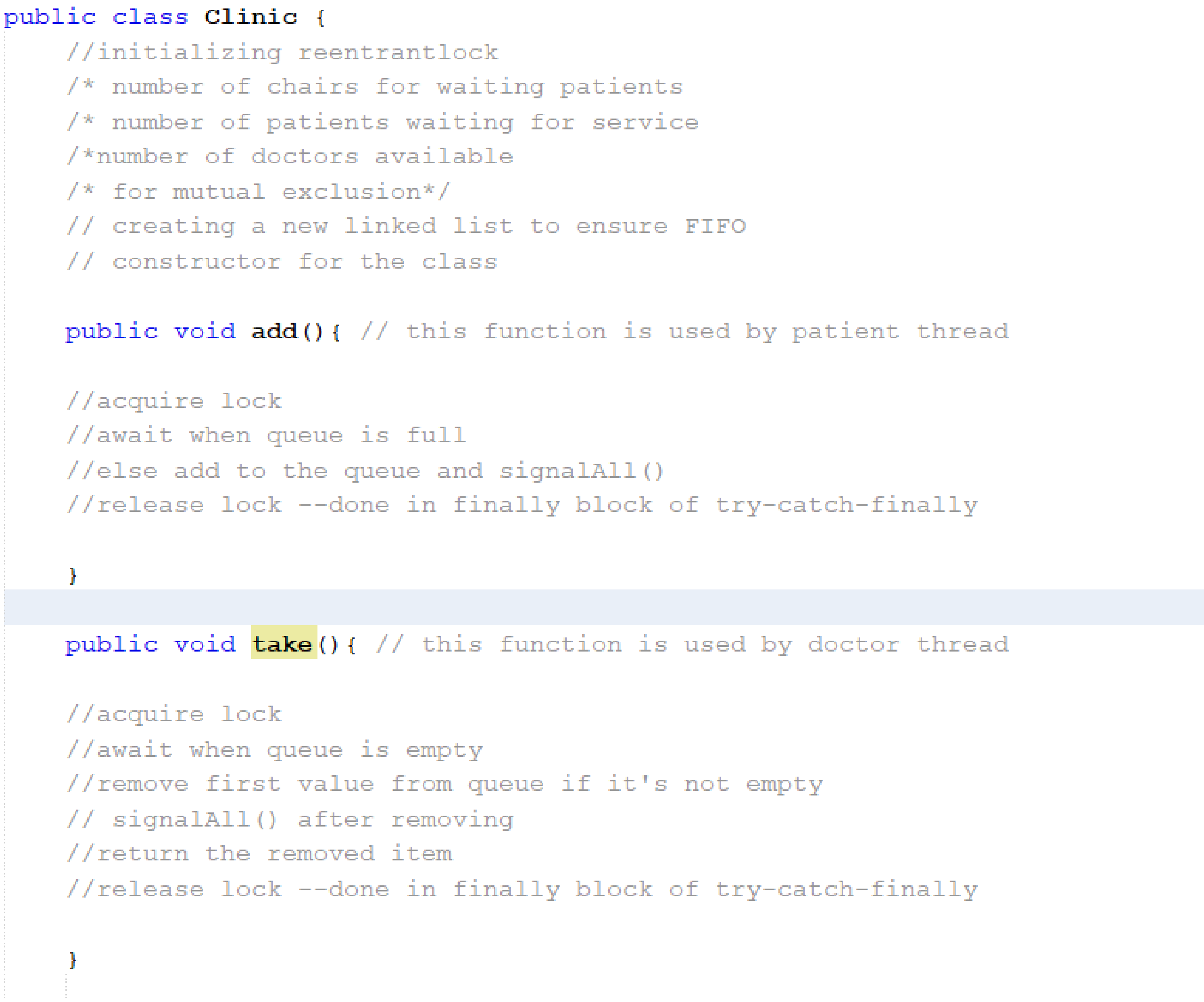
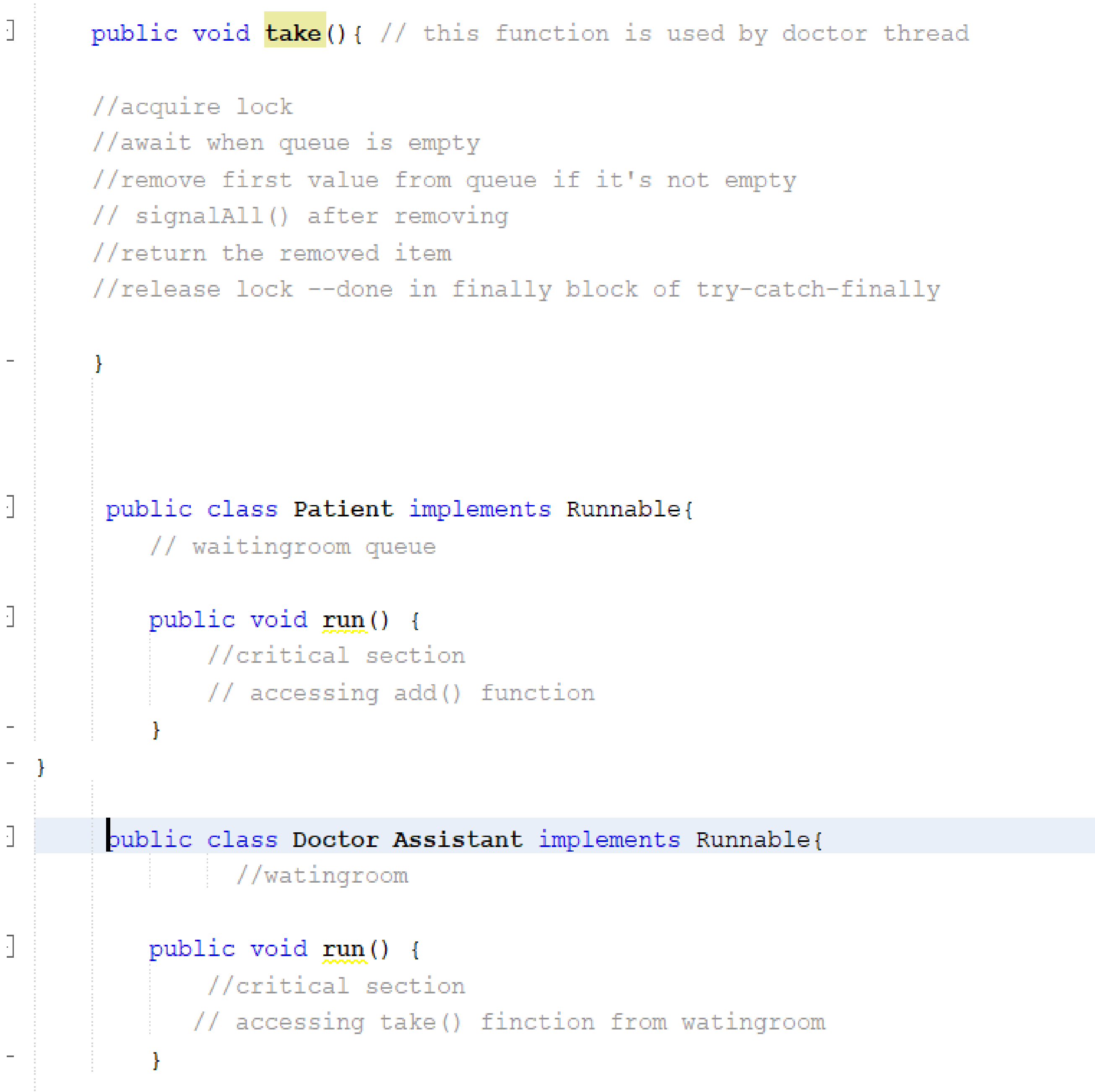
Solution pseudocode:

```
public class Clinic {
            initializing reentrantlock
             number of chairs for waiting patients
              number of patients waiting for service
             number of doctors available
              for mutual exclusion*/
              creating a new linked list to ensure FIFO
              constructor for the class

             public void add(){   // this function is used by patient thread

                    acquire lock
                    await when queue is full
                   else add to the queue and signalAll()
                   release lock --done in finally block of try-catch-finally

             }

             public void take(){ // this function is used by doctor thread

                     acquire lock
                     await when queue is empty
                     remove first value from queue if it's not empty
                      signalAll() after removing
                     return the removed item
                    release lock --done in finally block of try-catch-finally

             }
     public class Patient implements Runnable{
                    waitingroom queue

              public void run() {
                     critical section
                      accessing add() function
                 }
         }
```

```
public class Doctor_Assistant implements Runnable{
          waitingroom
      public void run() {
         critical section
         accessing take() function from waitingroom
      }



   }
```



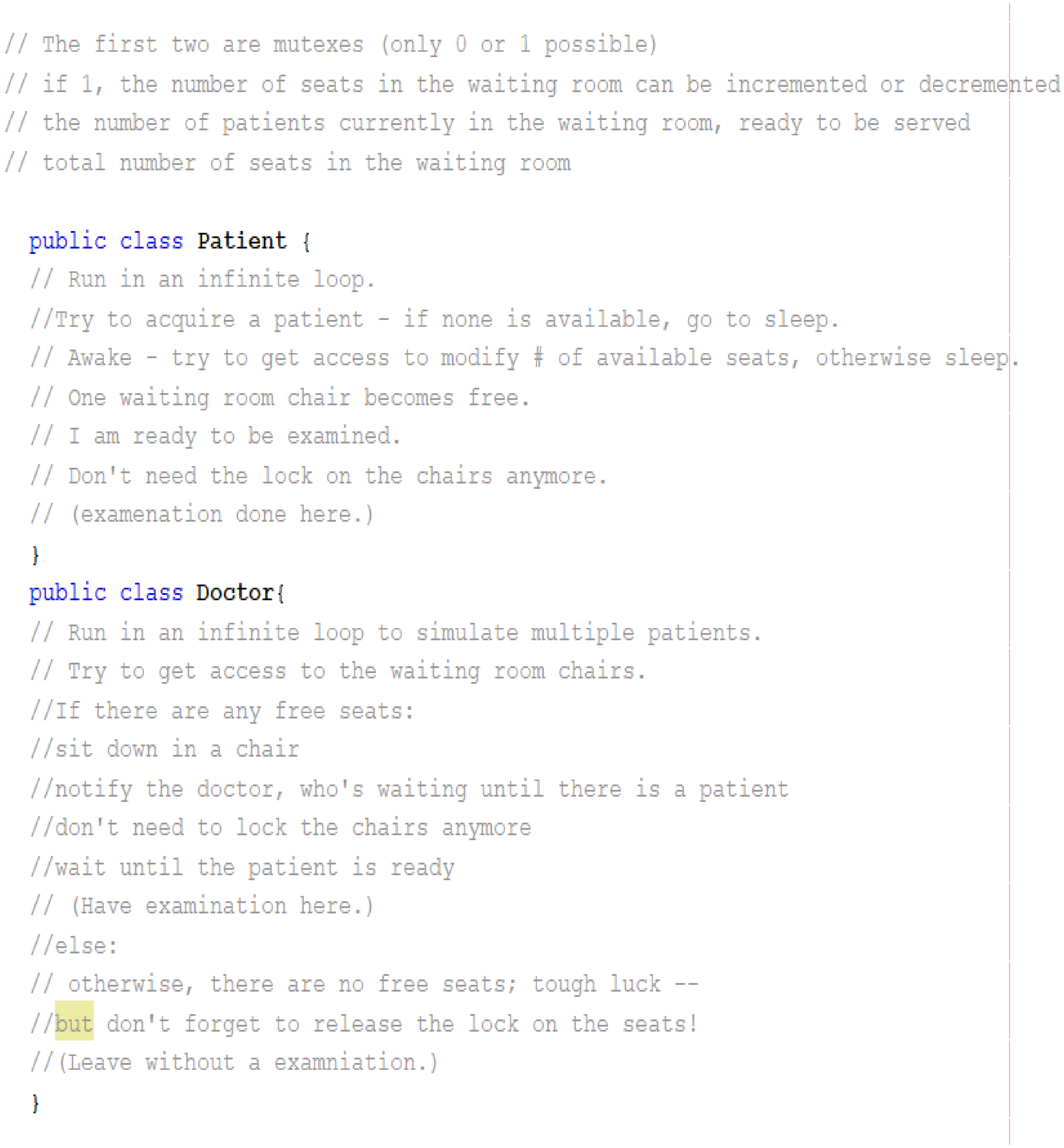
```
public class Clinic {
    //initializing reentrantlock
    /* number of chairs for waiting patients
    /* number of patients waiting for service
    /*number of doctors available
    /* for mutual exclusion*/
    // creating a new linked list to ensure FIFO
    // constructor for the class

    public void add(){ // this function is used by patient thread

    //acquire lock
    //await when queue is full
    //else add to the queue and signalAll()
    //release lock --done in finally block of try-catch-finally

    }

    public void take(){ // this function is used by doctor thread

    //acquire lock
    //await when queue is empty
    //remove first value from queue if it's not empty
    // signalAll() after removing
    //return the removed item
    //release lock --done in finally block of try-catch-finally

    }
```



```
    public void take(){ // this function is used by doctor thread

    //acquire lock
    //await when queue is empty
    //remove first value from queue if it's not empty
    // signalAll() after removing
    //return the removed item
    //release lock --done in finally block of try-catch-finally

    }



     public class Patient implements Runnable{
        // waitingroom queue

        public void run() {
            //critical section
            // accessing add() function
        }
}

     public class Doctor Assistant implements Runnable{
            //watingroom

        public void run() {
            //critical section
           // accessing take() finction from watingroom
        }
```

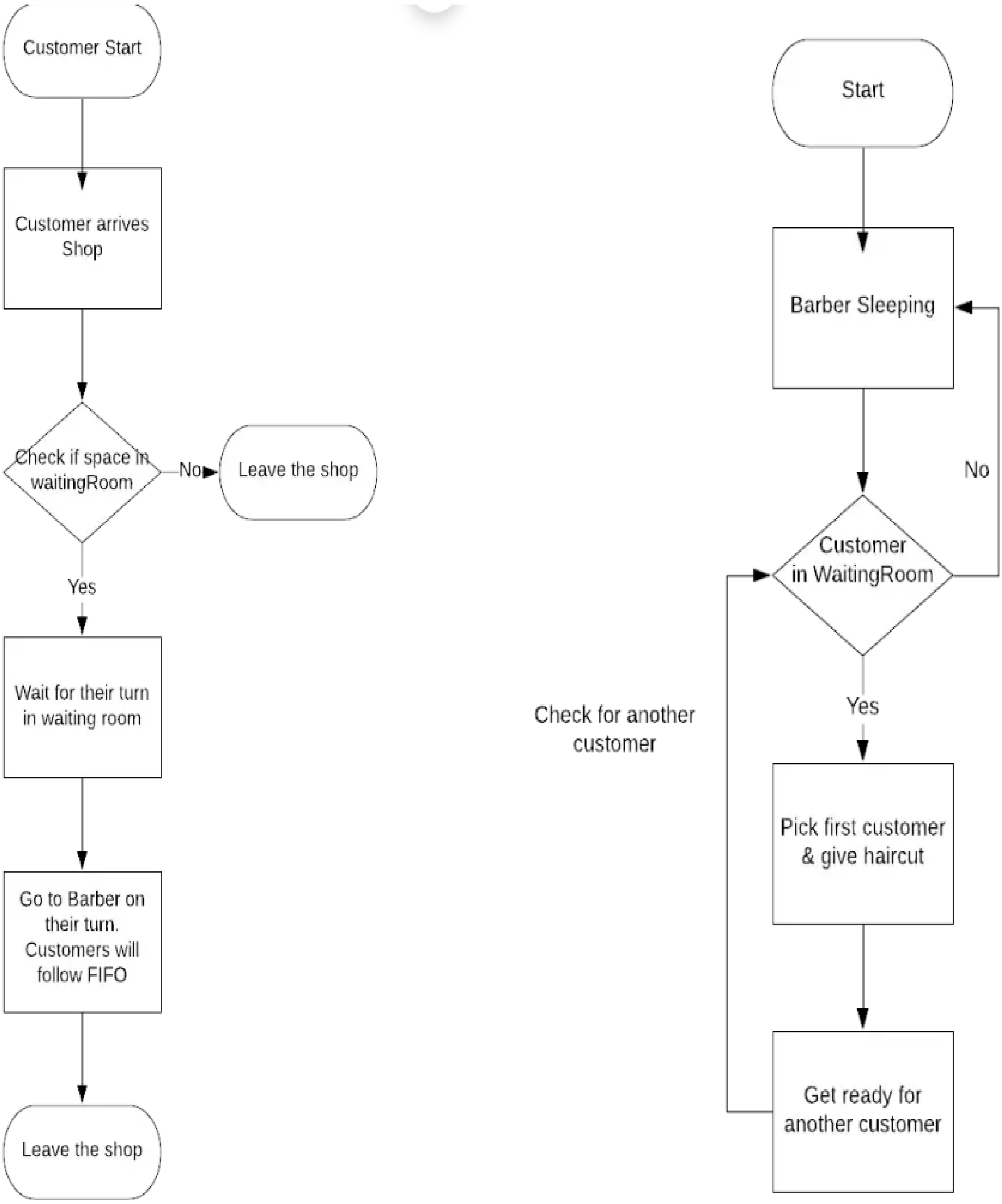
pseudocode -2

```
    // The first two are mutexes (only 0 or 1 possible)
    //  if 1, the number of seats in the waiting room can be incremented or decremented
    //   the number of patients currently in the waiting room, ready to be served
    // total number of seats in the waiting room

public class Patient {
        // Run in an infinite loop.
        //Try to acquire a patient - if none is available, go to sleep.
        // Awake - try to get access to modify # of available seats, otherwise sleep.
        // One waiting room chair becomes free.
        // I am ready to be examined.
        // Don't need the lock on the chairs anymore.
        // (examination done here.)
        }
public class Doctor{
        // Run in an infinite loop to simulate multiple patients.
        // Try to get access to the waiting room chairs.
        //If there are any free seats:
        //sit down in a chair
        //notify the doctor, who's waiting until there is a patient
        //don't need to lock the chairs anymore
        //wait until the patient is ready
        // (Have examination here.)
        //else:
        // otherwise, there are no free seats; tough luck --
        //but don't forget to release the lock on the seats!
        //(Leave without a examination.)
        }
```



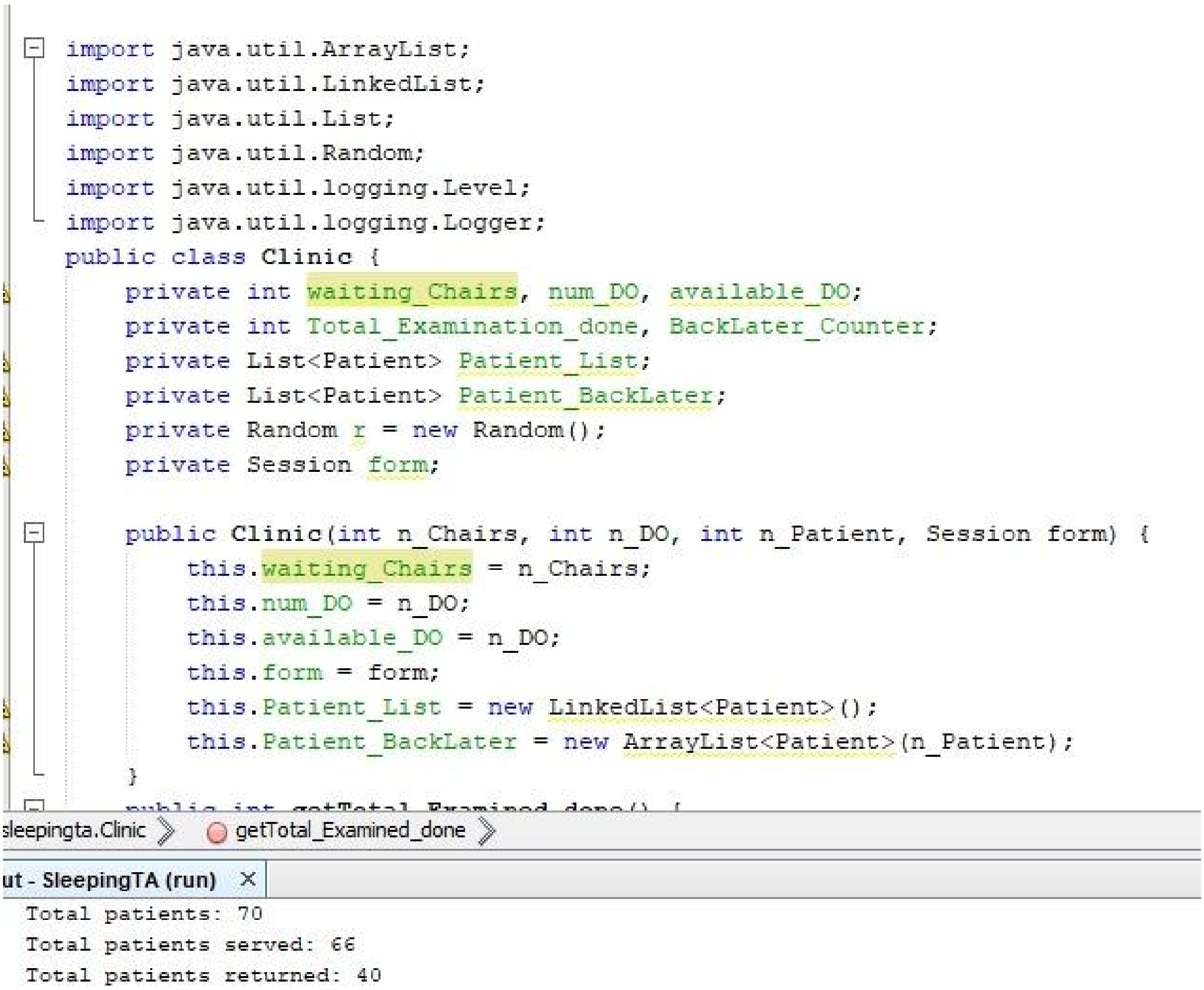
```
// The first two are mutexes (only 0 or 1 possible)
// if 1, the number of seats in the waiting room can be incremented or decremented
// the number of patients currently in the waiting room, ready to be served
// total number of seats in the waiting room

  public class Patient {
  // Run in an infinite loop.
  //Try to acquire a patient - if none is available, go to sleep.
  // Awake - try to get access to modify # of available seats, otherwise sleep.
  // One waiting room chair becomes free.
  // I am ready to be examined.
  // Don't need the lock on the chairs anymore.
  // (examenation done here.)
  }
  public class Doctor{
  // Run in an infinite loop to simulate multiple patients.
  // Try to get access to the waiting room chairs.
  //If there are any free seats:
  //sit down in a chair
  //notify the doctor, who's waiting until there is a patient
  //don't need to lock the chairs anymore
  //wait until the patient is ready
  // (Have examination here.)
  //else:
  // otherwise, there are no free seats; tough luck --
  //but don't forget to release the lock on the seats!
  //(Leave without a examniation.)
  }
```

Barber and Customer Thread Designs


Customer Start
Customer arrives Shop
Check if space in waitingRoom
No
Leave the shop
Yes
Wait for their turn in waiting room
Go to Barber on their turn. Customers will follow FIFO
Leave the shop
Start
Barber Sleeping
No
Customer in WaitingRoom
Yes
Check for another customer
Pick first customer & give haircut
Get ready for another customer

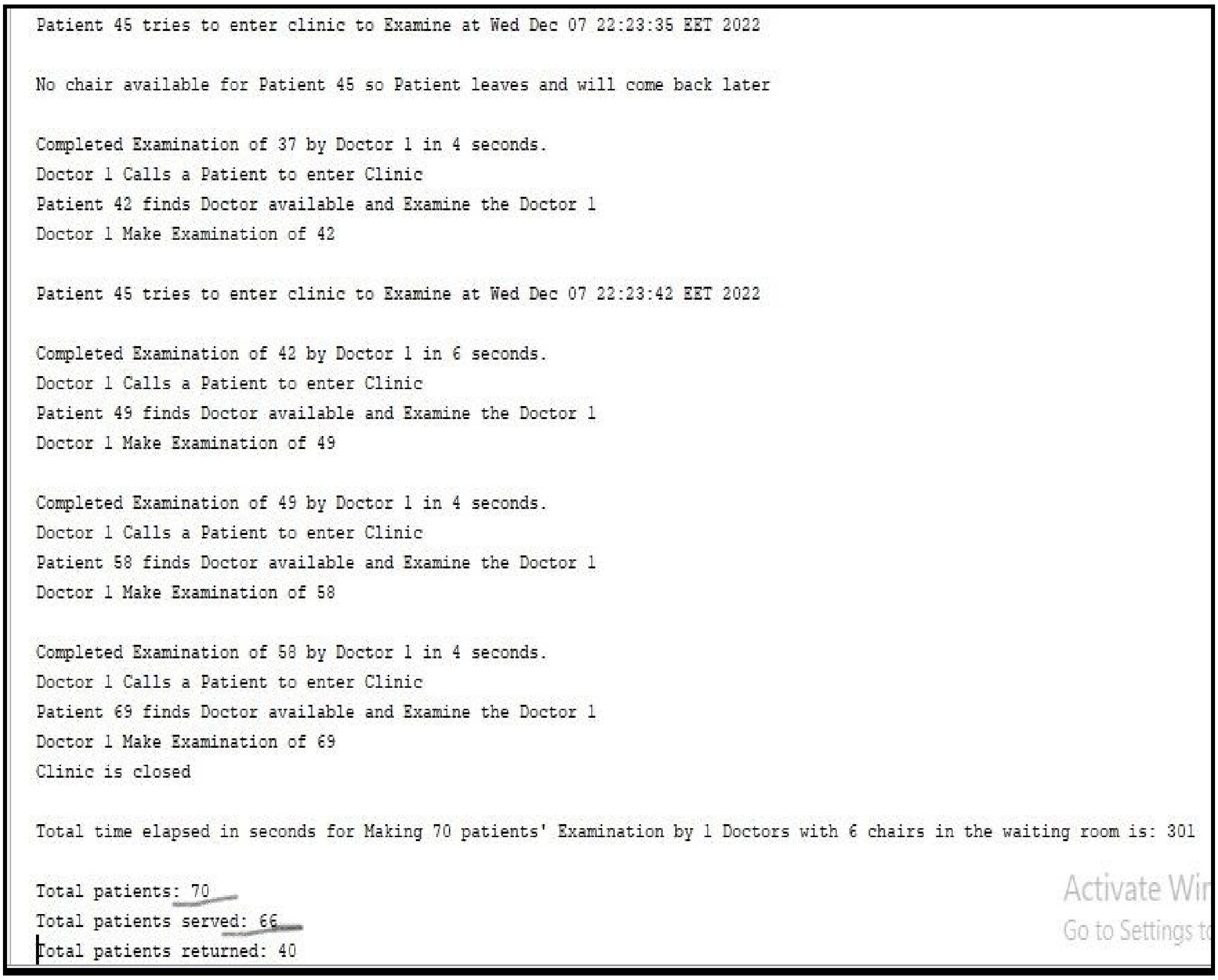
Examples of Deadlock

the deadlock will occur if the patient ends up waiting for the doctor and the doctor ends up waiting for the patient to arrive



```
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;
import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;
public class Clinic {
    private int waiting_Chairs, num_DO, available_DO;
    private int Total_Examination_done, BackLater_Counter;
    private List<Patient> Patient_List;
    private List<Patient> Patient_BackLater;
    private Random r = new Random();
    private Session form;

    public Clinic(int n_Chairs, int n_DO, int n_Patient, Session form) {
        this.waiting_Chairs = n_Chairs;
        this.num_DO = n_DO;
        this.available_DO = n_DO;
        this.form = form;
        this.Patient_List = new LinkedList<Patient>();
        this.Patient_BackLater = new ArrayList<Patient>(n_Patient);
    }
    public int getTotal_Examined_done() {
```

```
Total patients: 70
Total patients served: 66
Total patients returned: 40
```



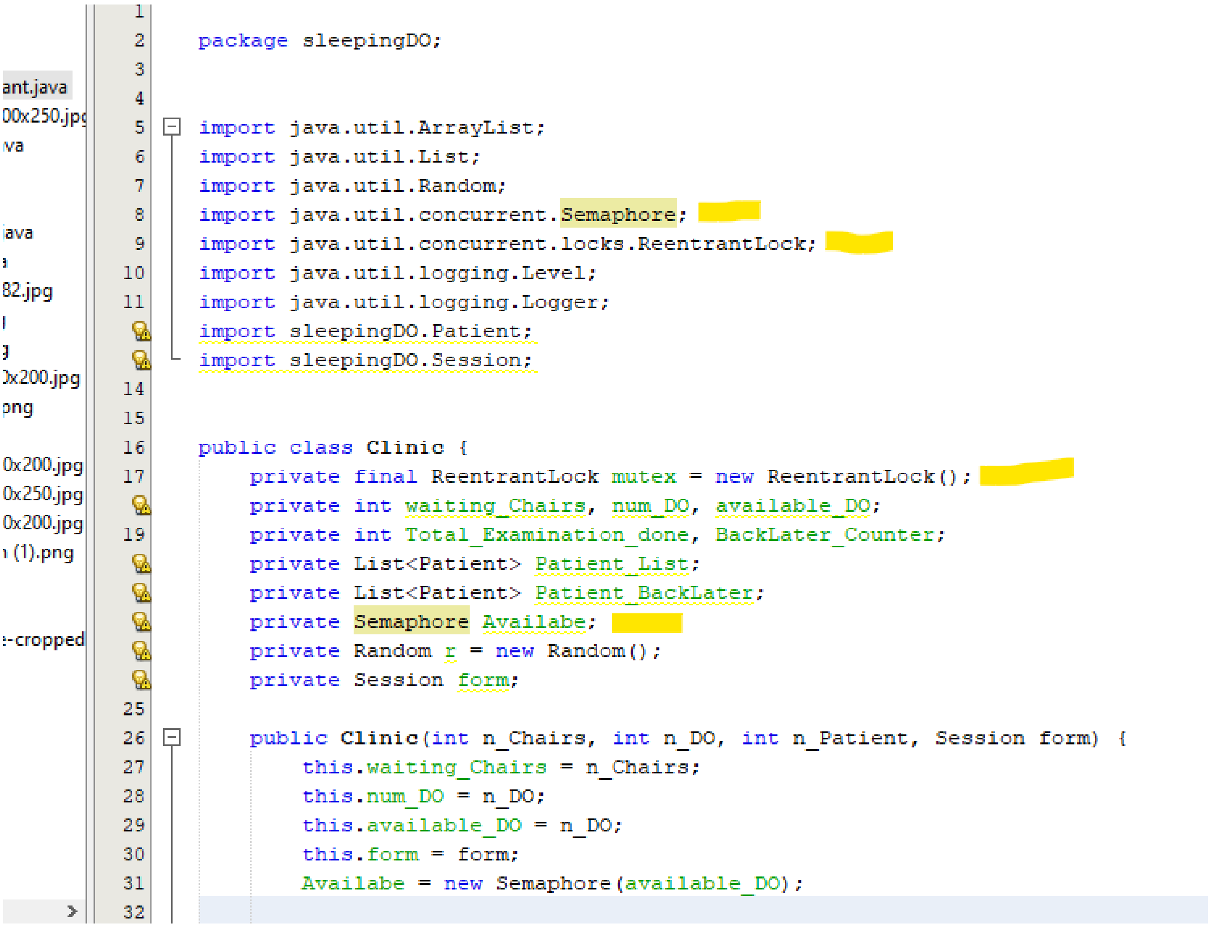
```
Patient 45 tries to enter clinic to Examine at Wed Dec 07 22:23:35 EET 2022

No chair available for Patient 45 so Patient leaves and will come back later

Completed Examination of 37 by Doctor 1 in 4 seconds.
Doctor 1 Calls a Patient to enter Clinic
Patient 42 finds Doctor available and Examine the Doctor 1
Doctor 1 Make Examination of 42

Patient 45 tries to enter clinic to Examine at Wed Dec 07 22:23:42 EET 2022

Completed Examination of 42 by Doctor 1 in 6 seconds.
Doctor 1 Calls a Patient to enter Clinic
Patient 49 finds Doctor available and Examine the Doctor 1
Doctor 1 Make Examination of 49

Completed Examination of 49 by Doctor 1 in 4 seconds.
Doctor 1 Calls a Patient to enter Clinic
Patient 58 finds Doctor available and Examine the Doctor 1
Doctor 1 Make Examination of 58

Completed Examination of 58 by Doctor 1 in 4 seconds.
Doctor 1 Calls a Patient to enter Clinic
Patient 69 finds Doctor available and Examine the Doctor 1
Doctor 1 Make Examination of 69
Clinic is closed

Total time elapsed in seconds for Making 70 patients' Examination by 1 Doctors with 6 chairs in the waiting room is: 301

Total patients: 70
Total patients served: 66
Total patients returned: 40
```

How did solve deadlock

The solution to this problem includes three Semaphore.First is for the patient which counts the number of patients present in the waiting room (patient in the doctor chair is not included because he is not waiting). Second, the doctor 0 or 1 is used to tell whether the doctor is idle or is working, And the third mutex is used to provide the mutual exclusion which is required for the process to execute.

and use the ReentrantLock→mutex

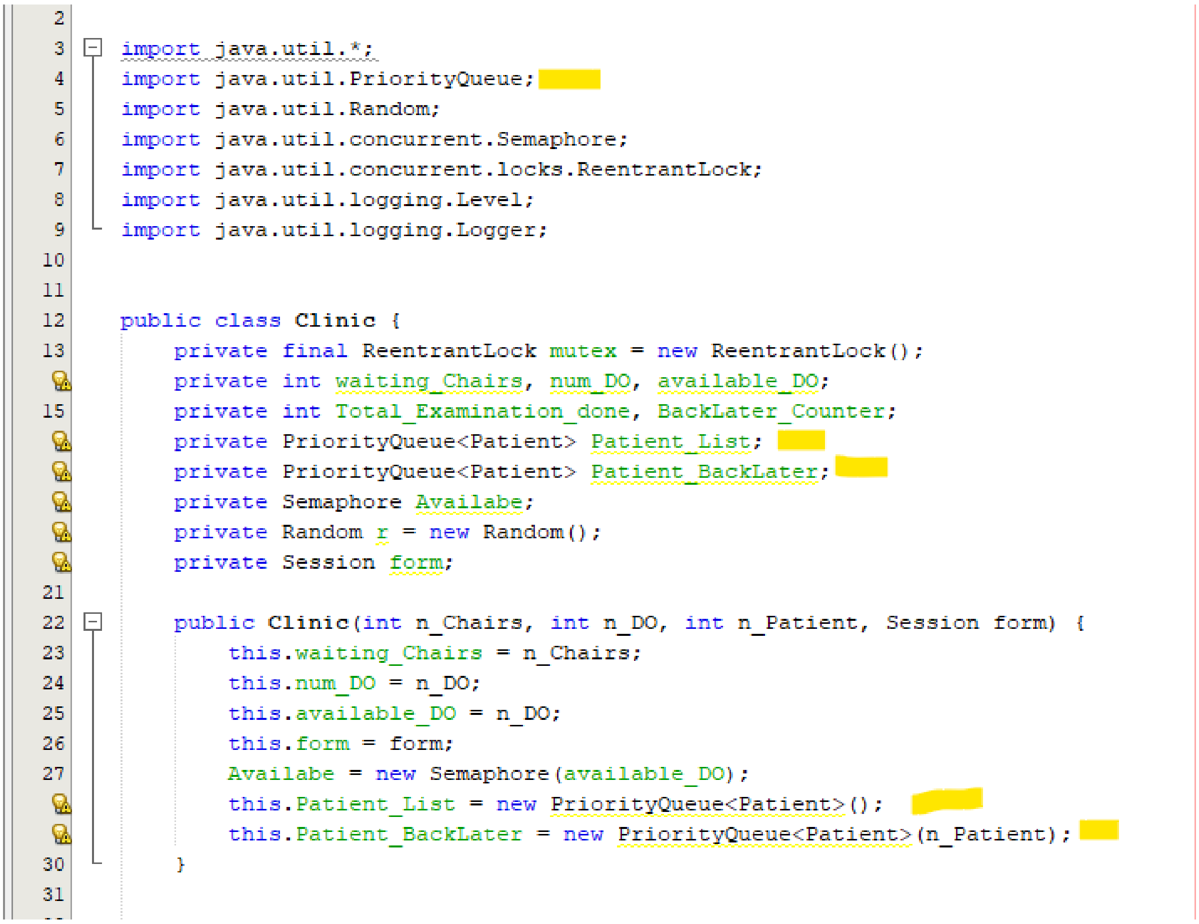
A mutex provides mutual exclusion,one patient can have the key (mutex) and proceed with their work



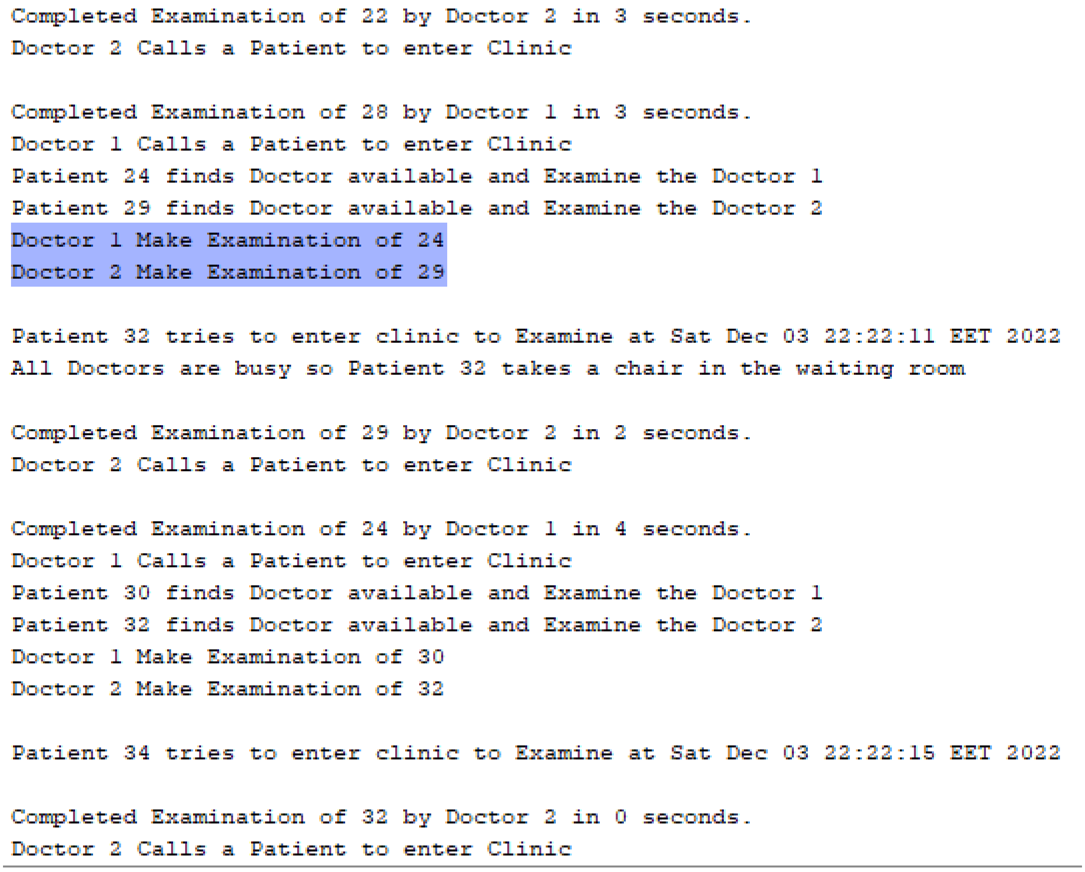
```

package sleepingDO;


import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import java.util.concurrent.Semaphore;
import java.util.concurrent.locks.ReentrantLock;
import java.util.logging.Level;
import java.util.logging.Logger;
import sleepingDO.Patient;
import sleepingDO.Session;


public class Clinic {
    private final ReentrantLock mutex = new ReentrantLock();
    private int waiting_Chairs, num_DO, available_DO;
    private int Total_Examination_done, BackLater_Counter;
    private List<Patient> Patient_List;
    private List<Patient> Patient_BackLater;
    private Semaphore Availabe;
    private Random r = new Random();
    private Session form;

    public Clinic(int n_Chairs, int n_DO, int n_Patient, Session form) {
        this.waiting_Chairs = n_Chairs;
        this.num_DO = n_DO;
        this.available_DO = n_DO;
        this.form = form;
        Availabe = new Semaphore(available_DO);

```

Examples of starvation

problem of starvation will occur if the patients don’t follow any order for getting a examination , as some won’t get a examination even though even after waiting for a long time



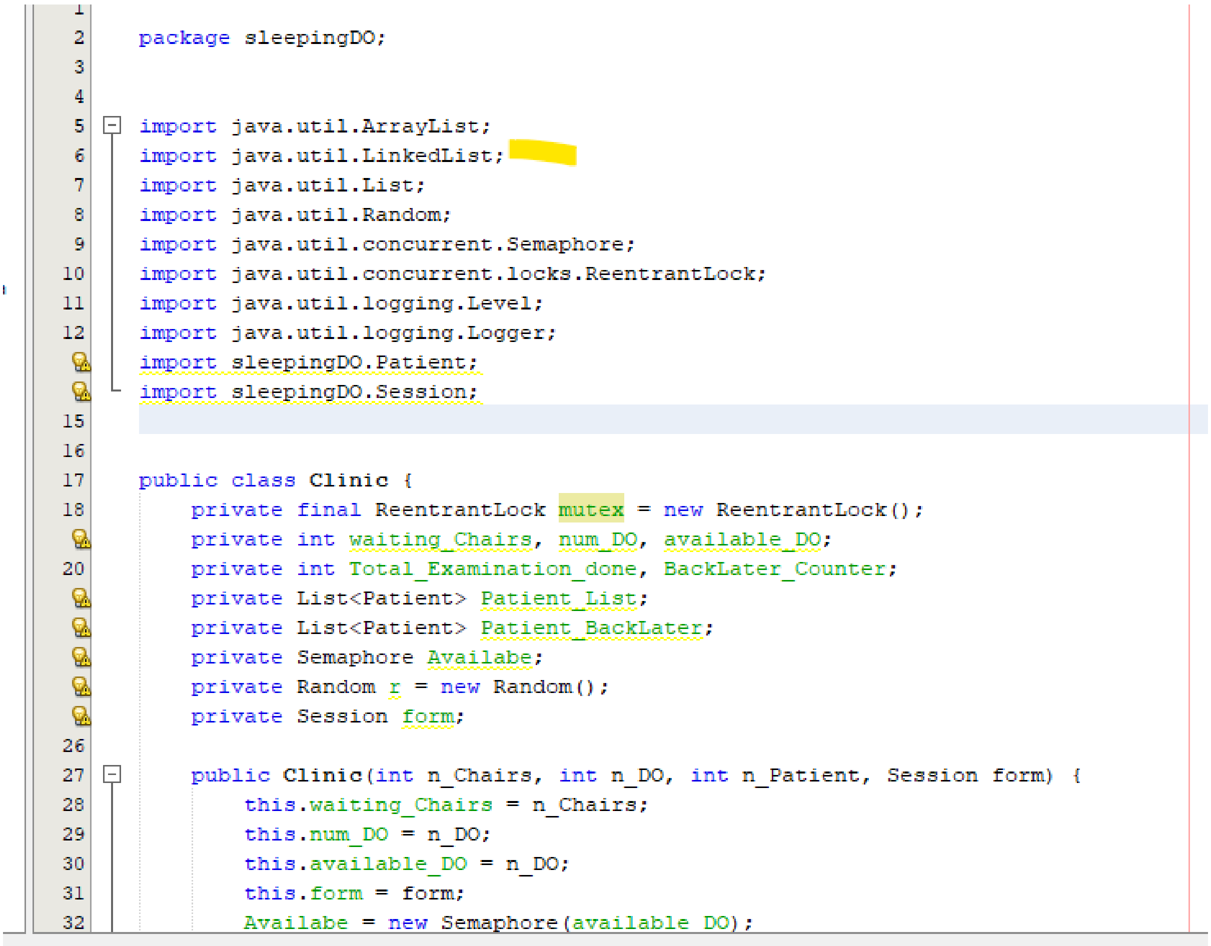
```

import java.util.*;
import java.util.PriorityQueue;
import java.util.Random;
import java.util.concurrent.Semaphore;
import java.util.concurrent.locks.ReentrantLock;
import java.util.logging.Level;
import java.util.logging.Logger;


public class Clinic {
    private final ReentrantLock mutex = new ReentrantLock();
    private int waiting_Chairs, num_DO, available_DO;
    private int Total_Examination_done, BackLater_Counter;
    private PriorityQueue<Patient> Patient_List;
    private PriorityQueue<Patient> Patient_BackLater;
    private Semaphore Availabe;
    private Random r = new Random();
    private Session form;

    public Clinic(int n_Chairs, int n_DO, int n_Patient, Session form) {
        this.waiting_Chairs = n_Chairs;
        this.num_DO = n_DO;
        this.available_DO = n_DO;
        this.form = form;
        Availabe = new Semaphore(available_DO);
        this.Patient_List = new PriorityQueue<Patient>();
        this.Patient_BackLater = new PriorityQueue<Patient>(n_Patient);
    }

```



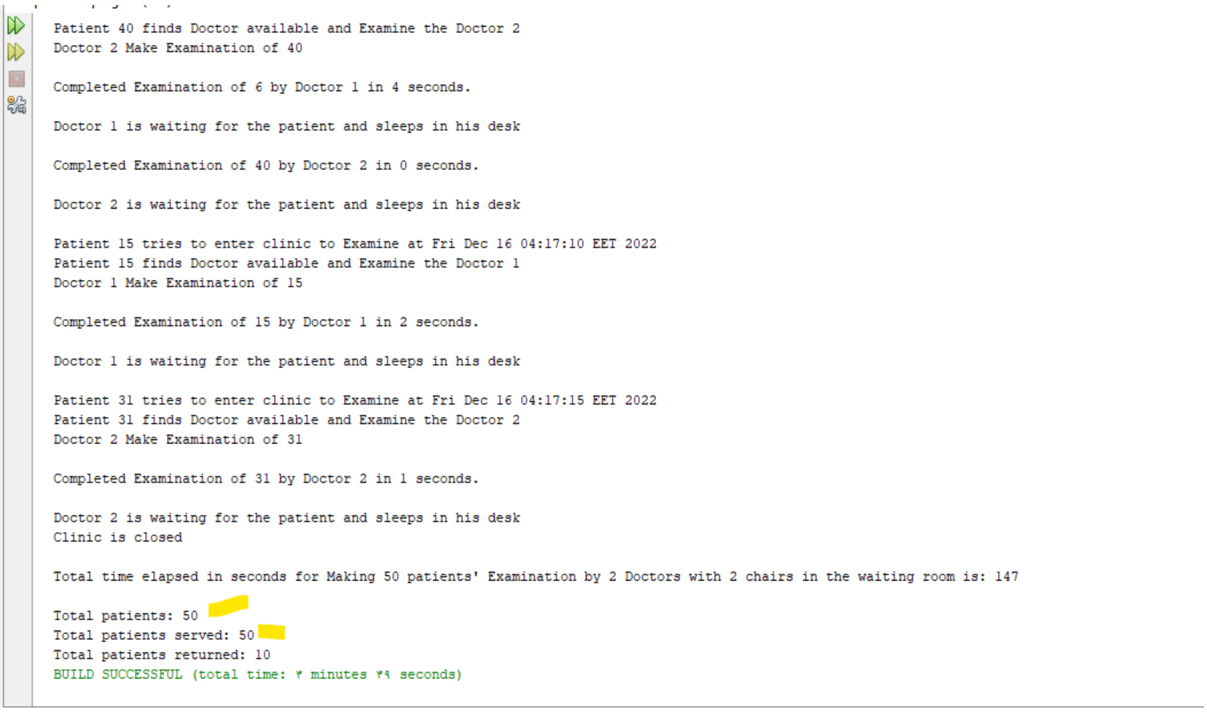
```
Completed Examination of 22 by Doctor 2 in 3 seconds.
Doctor 2 Calls a Patient to enter Clinic

Completed Examination of 28 by Doctor 1 in 3 seconds.
Doctor 1 Calls a Patient to enter Clinic
Patient 24 finds Doctor available and Examine the Doctor 1
Patient 29 finds Doctor available and Examine the Doctor 2
Doctor 1 Make Examination of 24
Doctor 2 Make Examination of 29

Patient 32 tries to enter clinic to Examine at Sat Dec 03 22:22:11 EET 2022
All Doctors are busy so Patient 32 takes a chair in the waiting room

Completed Examination of 29 by Doctor 2 in 2 seconds.
Doctor 2 Calls a Patient to enter Clinic

Completed Examination of 24 by Doctor 1 in 4 seconds.
Doctor 1 Calls a Patient to enter Clinic
Patient 30 finds Doctor available and Examine the Doctor 1
Patient 32 finds Doctor available and Examine the Doctor 2
Doctor 1 Make Examination of 30
Doctor 2 Make Examination of 32

Patient 34 tries to enter clinic to Examine at Sat Dec 03 22:22:15 EET 2022

Completed Examination of 32 by Doctor 2 in 0 seconds.
Doctor 2 Calls a Patient to enter Clinic
```

How did solve starvation

To handle this problem in my code I have inserted the patients in a linked list which follows the first in first out property. So, every time a patient sits in a waiting room, they will be selected by the doctor in first come first serve basis. We could have also used other data structures like a stack, but the linked list seems like the best choice for this scenario



```java

package sleepingDO;


import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;
import java.util.Random;
import java.util.concurrent.Semaphore;
import java.util.concurrent.locks.ReentrantLock;
import java.util.logging.Level;
import java.util.logging.Logger;
import sleepingDO.Patient;
import sleepingDO.Session;


public class Clinic {
    private final ReentrantLock mutex = new ReentrantLock();
    private int waiting_Chairs, num_DO, available_DO;
    private int Total_Examination_done, BackLater_Counter;
    private List<Patient> Patient_List;
    private List<Patient> Patient_BackLater;
    private Semaphore Availabe;
    private Random r = new Random();
    private Session form;

    public Clinic(int n_Chairs, int n_DO, int n_Patient, Session form) {
        this.waiting_Chairs = n_Chairs;
        this.num_DO = n_DO;
        this.available_DO = n_DO;
        this.form = form;
        Availabe = new Semaphore(available_DO);
```



```
Patient 40 finds Doctor available and Examine the Doctor 2
Doctor 2 Make Examination of 40

Completed Examination of 6 by Doctor 1 in 4 seconds.

Doctor 1 is waiting for the patient and sleeps in his desk

Completed Examination of 40 by Doctor 2 in 0 seconds.

Doctor 2 is waiting for the patient and sleeps in his desk

Patient 15 tries to enter clinic to Examine at Fri Dec 16 04:17:10 EET 2022
Patient 15 finds Doctor available and Examine the Doctor 1
Doctor 1 Make Examination of 15

Completed Examination of 15 by Doctor 1 in 2 seconds.

Doctor 1 is waiting for the patient and sleeps in his desk

Patient 31 tries to enter clinic to Examine at Fri Dec 16 04:17:15 EET 2022
Patient 31 finds Doctor available and Examine the Doctor 2
Doctor 2 Make Examination of 31

Completed Examination of 31 by Doctor 2 in 1 seconds.

Doctor 2 is waiting for the patient and sleeps in his desk
Clinic is closed

Total time elapsed in seconds for Making 50 patients' Examination by 2 Doctors with 2 chairs in the waiting room is: 147

Total patients: 50
Total patients served: 50
Total patients returned: 10
BUILD SUCCESSFUL (total time: ٢ minutes ٢٩ seconds)
```

Explanation for real world application and how did apply the problem

-the real word application is a clinic

-Problem : The analogy is based upon a hypothetical clinic with one doctor. There is a clinic which has one doctor, one doctor chair, and n chairs for waiting for patients if there are any to sit on the chair.

- **If there is no patient , then the doctor sleeps in his own chair.**
- **When a patient arrives, he has to wake up the doctor.**
- **If there are many patients and the doctor examines the patient , then the remaining patients either wait if there are empty chairs in the waiting room or they leave if no chairs are empty.**

-Solution : The solution to this problem includes three semaphores .First is for the patient which counts the number of patients present in the waiting room (patient in the doctor chair is not included because he is not waiting). Second, the doctor 0 or 1 is used to tell whether the doctor is idle or is working, And the third mutex is used to provide the mutual

exclusion which is required for the process to execute. In the solution, the patient has the record of the number of patients waiting in the waiting room if the number of patients is equal to the number of chairs in the waiting room then the upcoming patient leaves the clinic .

When the doctor shows up in the morning, he executes the procedure doctor , causing him to block on the semaphore patients because it is initially 0. Then the doctor goes to sleep until the first patient comes up.

When a patient arrives, he executes patient procedure the patient acquires the mutex for entering the critical region, if another patient enters thereafter, the second one will not be able to anything until the first one has released the mutex. The patient then checks the chairs in the waiting room if waiting patients are less then the number of chairs then he sits otherwise he leaves and releases the mutex.

If the chair is available then patient sits in the waiting room and increments the variable waiting value and also increases the patient 's semaphore this wakes up the doctor if he is sleeping.

At this point, patient and doctor are both awake and the doctor is ready to give that patient an examination . When the examination is over, the patient exits the procedure and if there are no patients in waiting room doctor sleeps.