Semaphores:

Semaphore	V	Resource	Wait Operation	Signal Operation
receptionistAvailable	1	Receptionist is available	Patient waits for the receptionist to be available.	Patient signals that the receptionist is now available for the next patient.
nurseAvailable[doctor id]	1	Each nurse is available {1,1,1} There should be 3	Patient waits for the nurse to be available	Patient signals that the nurse is now available for the next patient.
doctorAvaiable [doctor id] i.e. 3	1	Each doctor is available {1,1,1} There should be 3	Patient waits for the doctor to be available.	Patient signals that the doctor is now available for the next patient.
patientWaiting	0	Patients waiting in the waiting room	Receptionist waits for a patient to be in the waiting room.	Receptionist signals that a patient is in the waiting room.

All Threads:

```
//Receptionist Thread

while (true)
{
    wait(patientWaiting);
    registerPatient();
    signal(receptionistAvailable);
}
```

```
//Nurse Thread (one per doctor)

while (true)
{
    wait(receptionistAvailable);
    notifyPatient();
    leadPatientToDoctorOffice();
    signal(nurseAvailable[doctorId]);
}
```

```
//Doctor Thread (one per doctor)

while (true)
{
    wait(nurseAvailable[doctorId]);
    visitPatient();
    advisePatient();
    signal(doctorAvailable[doctorId]);
}
```

```
//Patient Thread (up to 15 patients)
void Patient
{
enterClinic();
wait(receptionistAvailable);
registerWithReceptionist();

wait(nurseAvailable[randomDoctorId]);
followNurse();

wait(doctorAvailable[randomDoctorId]);
listenToDoctorsAdvice();

leaveClinic();
}
```

Java Pseudocode: need to import (import java.util.concurrent.*;)

```
// Define the semaphores
Semaphore receptionistAvailable = new Semaphore(1);
Semaphore[] doctorAvailable = new Semaphore[3];
Semaphore[] nurseAvailable = new Semaphore[3];
Semaphore patientWaiting = new Semaphore(0);

class Receptionist extends Thread {
    public void run() {
        while (true) {
            try {
                patientWaiting.acquire();
                registerPatient();
                receptionistAvailable.release();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

class Nurse extends Thread {
    int doctorId;
    public Nurse(int doctorId) {
        this.doctorId = doctorId;
    }
    public void run() {
        while (true) {
            try {
                receptionistAvailable.acquire();
                notifyPatient();
                leadPatientToDoctorOffice();
                nurseAvailable[doctorId].release();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

class Doctor extends Thread {
    int doctorId;
    public Doctor(int doctorId) {
        this.doctorId = doctorId;
    }
    public void run() {
        while (true) {
            try {
                nurseAvailable[doctorId].acquire();
                visitPatient();
                advisePatient();
                doctorAvailable[doctorId].release();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

class Patient extends Thread {
    int doctorId;
    public Patient(int doctorId) {
        this.doctorId = doctorId;
    }
    public void run() {
        enterClinic();
        try {
            receptionistAvailable.acquire();
            registerWithReceptionist();
            nurseAvailable[doctorId].acquire();
            followNurse();
            doctorAvailable[doctorId].acquire();
            listenToDoctorsAdvice();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        leaveClinic();
```