Lumanlan, Almea S.

Villanueva, Ian Mc. Coy B.

Midterm Paired Task 1.

Object Oriented Analysis and Design

1. **Following the OO workflow as discussed in class,** you are task to design the OO Model of the given problem (use draw.io) of the scenario below:

Problem Statement. Tiny Hospital keeps information on **patients** and **hospital rooms**. The system assigns each patient a patient ID number. In addition, the patient's name and date of birth are recorded. Some patients are resident patients (they spend at least one night in the hospital) and others are outpatients (they are treated and released). Resident patients are assigned to a room. Each room is identified by a room number. The **Tiny hospital system** also stores the room type (private or semi-private) and room fee. Overtime, each room will have many patients who stay in it. Each resident patient will stay in only one room. The hospital system has features that can view patient information and view whether a room is occupied or not. Both patient and room entities must have features that allows adding, updating and searching of records.

STEP1. IDENTIFY all the necessary **OBJECT** within the problem domain

- Patients
- Resident Patient
- Out Patient
- HospitalRoom
- TinyHospitalSystem

STEP 2. IDENTIFY all the properties and methods/behaviors in the

PATIENT

Properties:	Behaviors:
▪ PatientID	▪ addPatient()
▪ Name	▪ updatePatient()
▪ dateofbirth	▪ searchPatient()
▪ patientType	▪ viewPatient()

RESIDENT PATIENT

Properties:	Behaviors:
▪ roomNumber	▪ assignRoom()
	▪ Discharge()

OUT PATIENT

Properties:	Behaviors:
▪ visitDate	▪ recordVisit()

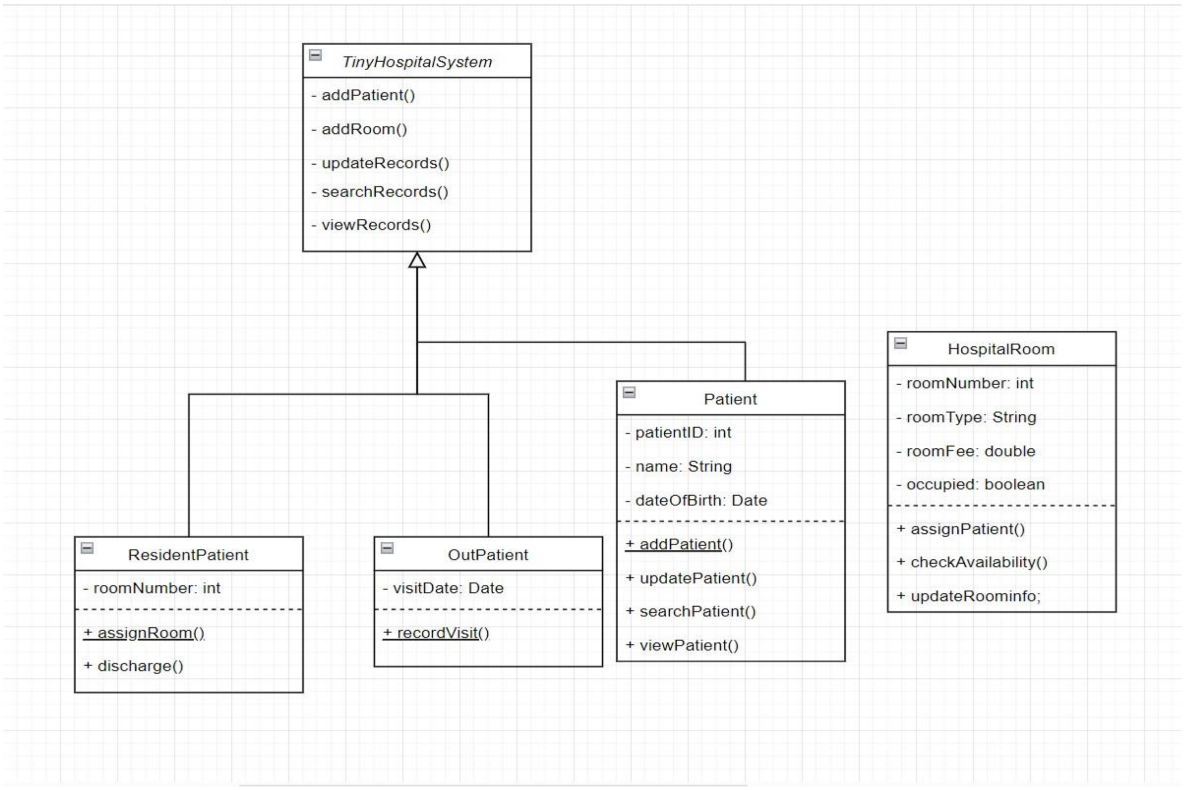
HOSPITAL ROOM

Properties:	Behaviors:
▪ roomNumber	▪ assignPtients()
▪ roomType	▪ checkAvailability()
▪ roomFee	▪ updateRoomInfo()
▪ occupied	

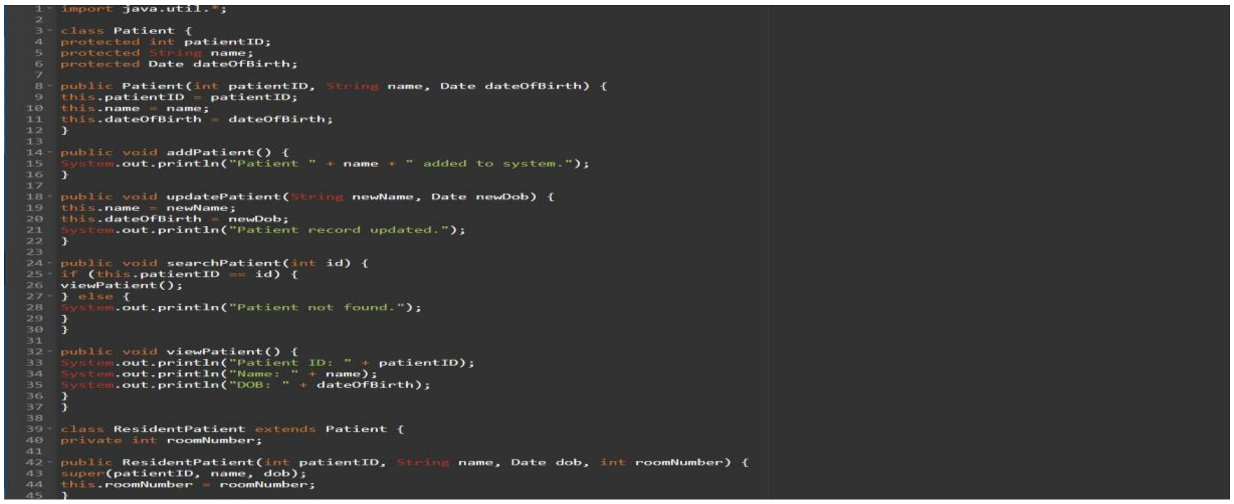
TINY HOSPITAL SYSTEM

Behaviors:
▪ addPatient()
▪ addRoom()
▪ updateRecord()
▪ searchRecord()
▪ viewRecord()

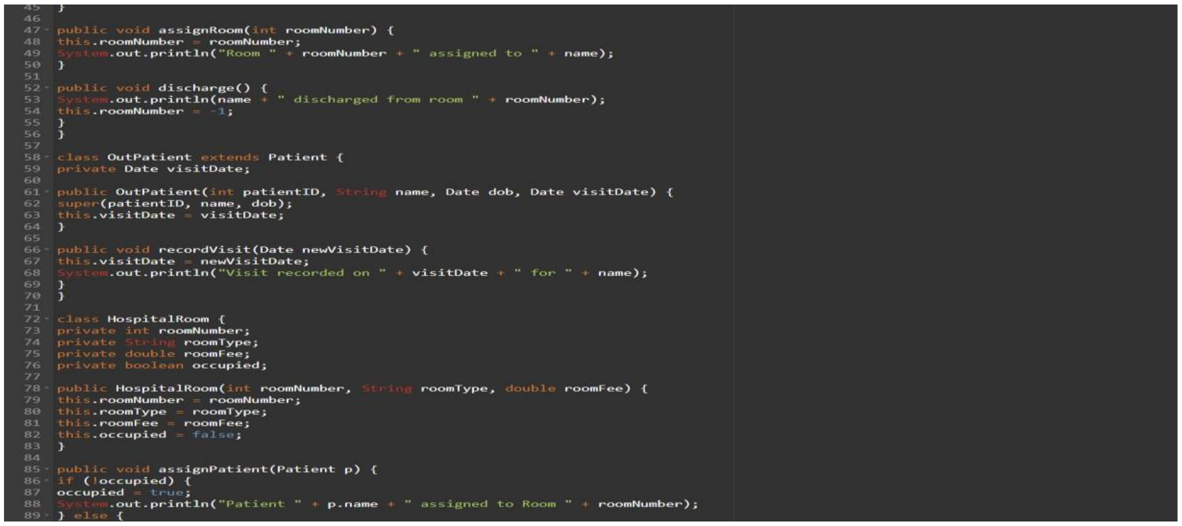
STEP 3. Design the MODEL using a Class Diagram (You may use draw.io to represent the Blueprint of all the class that you need to create)



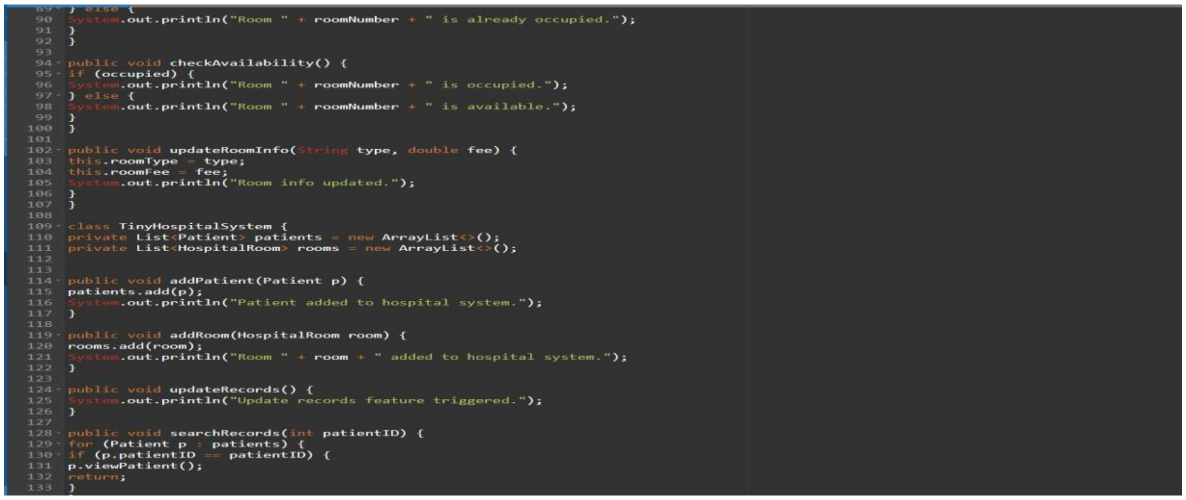
STEP 4. Implement the **class using Java code** construct of each interacting entities that you have identified.



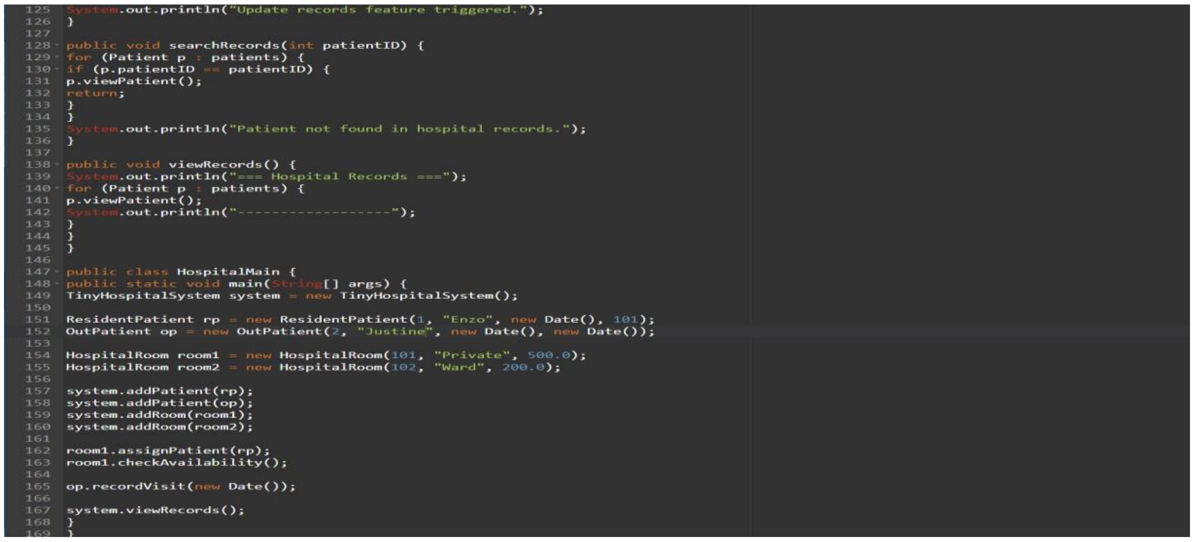
```java
import java.util.*;

class Patient {
protected int patientID;
protected String name;
protected Date dateOfBirth;

public Patient(int patientID, String name, Date dateOfBirth) {
this.patientID = patientID;
this.name = name;
this.dateOfBirth = dateOfBirth;
}

public void addPatient() {
System.out.println("Patient " + name + " added to system.");
}

public void updatePatient(String newName, Date newDob) {
this.name = newName;
this.dateOfBirth = newDob;
System.out.println("Patient record updated.");
}

public void searchPatient(int id) {
if (this.patientID == id) {
viewPatient();
} else {
System.out.println("Patient not found.");
}
}

public void viewPatient() {
System.out.println("Patient ID: " + patientID);
System.out.println("Name: " + name);
System.out.println("DOB: " + dateOfBirth);
}
}

class ResidentPatient extends Patient {
private int roomNumber;

public ResidentPatient(int patientID, String name, Date dob, int roomNumber) {
super(patientID, name, dob);
this.roomNumber = roomNumber;
}
```



```java
}

public void assignRoom(int roomNumber) {
this.roomNumber = roomNumber;
System.out.println("Room " + roomNumber + " assigned to " + name);
}

public void discharge() {
System.out.println(name + " discharged from room " + roomNumber);
this.roomNumber = -1;
}
}

class OutPatient extends Patient {
private Date visitDate;

public OutPatient(int patientID, String name, Date dob, Date visitDate) {
super(patientID, name, dob);
this.visitDate = visitDate;
}

public void recordVisit(Date newVisitDate) {
this.visitDate = newVisitDate;
System.out.println("Visit recorded on " + visitDate + " for " + name);
}
}

class HospitalRoom {
private int roomNumber;
private String roomType;
private double roomFee;
private boolean occupied;

public HospitalRoom(int roomNumber, String roomType, double roomFee) {
this.roomNumber = roomNumber;
this.roomType = roomType;
this.roomFee = roomFee;
this.occupied = false;
}

public void assignPatient(Patient p) {
if (!occupied) {
occupied = true;
System.out.println("Patient " + p.name + " assigned to Room " + roomNumber);
} else {
```



```java
} else {
System.out.println("Room " + roomNumber + " is already occupied.");
}
}

public void checkAvailability() {
if (occupied) {
System.out.println("Room " + roomNumber + " is occupied.");
} else {
System.out.println("Room " + roomNumber + " is available.");
}
}

public void updateRoomInfo(String type, double fee) {
this.roomType = type;
this.roomFee = fee;
System.out.println("Room info updated.");
}
}

class TinyHospitalSystem {
private List<Patient> patients = new ArrayList<>();
private List<HospitalRoom> rooms = new ArrayList<>();


public void addPatient(Patient p) {
patients.add(p);
System.out.println("Patient added to hospital system.");
}

public void addRoom(HospitalRoom room) {
rooms.add(room);
System.out.println("Room " + room + " added to hospital system.");
}

public void updateRecords() {
System.out.println("Update records feature triggered.");
}

public void searchRecords(int patientID) {
for (Patient p : patients) {
if (p.patientID == patientID) {
p.viewPatient();
return;
}
```



```java
System.out.println("Update records feature triggered.");
}

public void searchRecords(int patientID) {
for (Patient p : patients) {
if (p.patientID == patientID) {
p.viewPatient();
return;
}
}
System.out.println("Patient not found in hospital records.");
}

public void viewRecords() {
System.out.println("=== Hospital Records ===");
for (Patient p : patients) {
p.viewPatient();
System.out.println("------------------");
}
}
}

public class HospitalMain {
public static void main(String[] args) {
TinyHospitalSystem system = new TinyHospitalSystem();

ResidentPatient rp = new ResidentPatient(1, "Enzo", new Date(), 101);
OutPatient op = new OutPatient(2, "Justine", new Date(), new Date());

HospitalRoom room1 = new HospitalRoom(101, "Private", 500.0);
HospitalRoom room2 = new HospitalRoom(102, "Ward", 200.0);

system.addPatient(rp);
system.addPatient(op);
system.addRoom(room1);
system.addRoom(room2);

room1.assignPatient(rp);
room1.checkAvailability();

op.recordVisit(new Date());

system.viewRecords();
}
}
```

Note: Highlight all the outputs following the example from STEP 1 to STEP 4 as shown