
**Department of Computer Science and Mathematics**

**CSC490: Software Engineering**

# Hospital Management System

**By:**

**GHEEDA CHOUCAIR**

**RANIA DAKROUB**

**REINE EL FEREKH**

**HASSAN MSHAWRAB**

**Coordinate/Leader Project Phase 2: Rania Dakroub**

**19 April 2021**

# Contents

# I. Classes Description:

## 1. Hospital Services Management:

- **Patient:**

Includes all the patient personal and medical information needed by the hospital to store them in the database.

***Attributes:***

- **Patient_ID:** A numeric identifier of 9 numbers representing the patient's ID.
- **First_Name:** String datatype, represents the first name of the patient.
- **Last_Name:** String datatype, represent the last name of the patient.
- **Gender:** Char datatype, represent the gender of the patient in which the letter "F" represents a female, and the letter "M" represents a male.
- **Date_of_Birth:** String datatype, represents the date of birth of a patient.
- **Financial_Status:** String datatype, represent the type of insurance that a patient has.
- **Phone_Number**: int datatype, represents the patient personal phone number.
- **Address:** String datatype, represent the address that a patient resides in.
- **Diagnosis:** String datatype, represents a brief description of the diagnosis that is given to the admitted patient.
- **Date_oF_Entry**: String datatype, represents the date that the patient was admitted to the hospital.
- **Allergies:** String datatype, an array list that include the type of allergy that a patient has.
- **Reason_oF_Entrance:** String datatype, represent the reason of patient entrance to the hospital.

- **Admitting_Dr_Name:** String datatype, represents the name of the doctor who is responsible for the patient treatment.
- **Date_of_Exit:** String datatype, it included the date of which the patient exits the hospital.
- **PatientList:** HashTable<Integer, Patient>, where the patientID will be the key and the patient object are the value, and it is used to store the patient information.
- **PatientEntry:** HashTable<Integer, String>, where the patientID will be the key and the Date_oF_Entry is the value.

***Functions needed in the class:***

- **Patient (personal information, medical information):** the constructor of the class that is called to fill all the medical and personal patient information that are needed and should be used in the process of admitting the patient to the hospital.
- **getPatientList():** gets the HashTable of the patient list.
- **getPatientEntry():** gets the Hashtable of the patient entry.

## ➢ Patient Admission:

This class is responsible for the admission process of the patient who needs to enter the hospital.

***Functions needed in the class:***

- **Patient_Admission():** the constructor of the class.
- **Create_New_Patient_Id():** a new patient who is admitted to the hospital needs to have a unique ID that represent him. The ID will be of 9 numbers, the first 5 numbers are randomly generated, and the last 4 numbers represent the year in which the patient is admitted in.

Afterward, the randomly generated number will be concatenated with the year and the method will check if this ID does not exist in the database. If the generated ID exists in the database, the method will generate another new number of 5 digits and concatenate it with the year, and if the number does not exist in the database, it will be given to the patient.

- **Add_New_Patient():** this function will be used to add a new patient to the patients list. The function will first check if there are available vacancies in the hospital and according to that the admission procedure will be carried on. The previous function "Create_New_Patient_Id()" will be called to create a new unique ID for the patient upon adding him to the list of patients. Then, the this will call the function "getAvailableRooms()" from the class "Rooms" to choose a bed for the patient from the available beds in the hospital. This function will scan all the personal and medical information of the patient and create an object from the class "Patient" to add these scanned attributes to it. Afterwards, the function will add the patient to a HashTable representing the list of patients in the hospital.
- **Display_Patient(patient_ID):** this method will search for the patient according to the ID inserted as a parameter. If the ID is valid, the method will return all detailed personal and medical information of the registered patient. If the ID is not valid, a request to enter a valid value will be issued **"PLEASE ENTER A VALID ID"**.
- **Edit_Patient(patient_ID):** this method will first search if the entered patient ID exists in the database of the hospital. If the ID exists, the user will then be asked to enter the attribute that he/she needs to change. Afterwards, the user will be asked to enter the new attribute value. Finally, the method will return **"UPDATED SUCCESSFULLY".**

- **Electronic_Health_Record:**

This class is responsible of storing the patient information needed for the electronic health record. This class inherits attributes from the class patient.

***Attributes:***

- **Medication**: String datatype, includes the set of medication that a patient might be undergoing. This value can be set to null.
- **Immunization:** String datatype, represents the names of all the vaccines that a patient took from his/her birth. This value can be set to null.
- **PatientEHR:** HashTable<Integer, Electronic_Health_Record >, where the patientID will be the key and the Electronic_Health_Record is the value.

***Functions needed in the class:***

- **Electronic_Health_Record(attributes of Patient Class, Medication, Immunization):** it is the constructor of the class.
- **EHR_Display(ID):** this method will take the ID as a parameter to be used for searching for the electronic medical health records in the database. If the ID does not exist in the database, a statement "**THE SEARCH YOU PROVIDED IS UNAVAILABLE"** will be issued. Otherwise, the patient information will be displayed.
- **Edit_EHR(ID):** this method will take the patient ID as a parameter and if the ID is valid, it will allow the user to request changes and updates in the EHR of the patient. Otherwise, the method will return "**PLEASE ENTER A VALID ID"** if the ID that was entered does not exist in the database.

- **Patients Appointment Management:**

This class is responsible for all the data associated with scheduling and displaying the patient appointments in the hospital.

***Attributes:***

- **DrName:** String datatype, represents the name of the doctor that the appointment will be carried on with.
- **Table**: String datatype, it is a double array that includes the days and times of the appointments sheet and the available vacancies to be reserved by the patient.

***Functions needed in the class:***

- **Patient_Appointment_Management(String DrName):** the constructor of the class, it take the name of the doctor who the patient will take an appointment with.
- **Display_Appointment():** This method will allow to display a sheet that include the dates and times of the available appointments for each doctor.
- **takeAppointment(date information, patient name):** this method will be used by the authorized user to allow the process of reserving appointment. The method will check if the date requested is available, if not, the method will return **"THE DATE AND/OR TIME YOU ENTERED IS NOT CORRECT!".** If the date and time are correct, the method will check if this appointment was previously reserved. If the appointment was previously reserved, the method will return **"THIS APPOINTMENT IS RESERVED, PLEASE CHOOSE ANOTHER TIMING!",** otherwise, the patient will be able to take the appointment and the

method will display **"APPOINTMENT RESERVED SUCCESSFULLY!".** Afterwards, the user will be asked to enter information needed for the patient and this info will be stored in the patient database.

- **DeleteAppointment(date information, patient name):** this method will take from the patient the date and time information of the appointment he/she wants do delete. If the patient took an appointment at this time, the appointment will be deleted and a statement **"APPOINMENT DELETED SUCCESSFULLY"** will be displayed. Otherwise, a statement **"NO SUCH APPOINTMENT EXISTS"** will be displayed. If the time and date entered is not valid **"THE DATE AND/OR TIME YOU ENTERED IS NOT CORRECT!"** statement will be displayed.

- **Surgery Scheduling:**

This class is responsible for keeping track of the medical surgeries that are taking place at the hospital.

***Attributes:***

- **equipmentSurgery:** String datatype, that represents the surgery's equipment.
- **name:** String datatype, that represents the name of the staff that are responsible for the surgery.
- **Table:** String datatype, represents a 2D array that contains the dates and times of the appointment sheet and the available vacancies.

***Functions needed in the class:***

- **isAvailableequipmentSurgery():** This method will return a Boolean value "true" if the required material is available in the hospital. Otherwise, the method will return "false".
- **isAvailableName():** This method will return a Boolean value "true" if the staff are available for the surgery at the hospital. Otherwise, the method will return "false".
- **DisplaySurgerySchedule():** this method will be used to return a table that includes all the available surgeries with the dates and times.
- **EditAppointment(Date information):** this method will allow to add, edit, or delete the appointment of surgery.

**Pharmacy Services Management:**

- **Medicine Class:**

The Medicine class represents the information about medicines for the patient.

***Attributes:***

- **name:** String datatype, that represents the medicine's name.
- **Cost:** Int datatype, that represents the medicine's cost.
- **id:** Int datatype, that represents the medicine's id.
- **expirationDate:** Int datatype, that represents the medicine's expiration Date.

***Functions needed in the class:***

- **getName():** The user will be able to access the name of the medicine.
- **setName():** The user will be able to update and modify the name of the medicine.

- **getCost():** The user will have access to the cost of the medicine.
- **setCost():** The user will be able to update and modify the cost of the medicine.
- **getId():** The user will have access to the medicine's id.
- **setId():** The user will be able to update and modify the medicine's id.
- **getExpirationDate():** The user will be able to know the expiration date of the medicine.
- **setExpirationDate():** The user will be able to modify and update the expiration date of the medicine.

➢ **Med:**

***Attributes:***

- **m**: LinkedList <Medicine>, includes all the medicines in the hospital.

***Functions needed in the class:***

- **searchMedicine(name):** the method will allow to search for a certain specified medicine. If the medicine is found in the system, the method will display to the user all the information that are related to the searched medicine. If the medicine does not exist in the system, the method will issue a statement **"THIS MEDICINE DOES NOT EXIST".**
- **addMedicine(name, quantity):** this method will be used to add a certain medicine to the system. The method will search if the name of the medicine already exists in the system. If yes, the quantity will be added to the initial quantity of the medicine. If the name of the medicine is not available in the system, a new medicine with a given quantity will be added to the system.
- **removeMedicine(name, quantity):** the method will check if the name of the medicine exists in the linked list, if it does, the method will check for the quantity available. If the quantity greater

than the requested number of the medicine, the medicine will be removed, else, the system will issue a statement **"THIS QUANTITY IS NOT AVAILABLE, PLEASE ENTER AVAILABLE QUANTITY".** If the name of the medicine is not available in the database, the system will issue a statement **"THIS MEDICINE IS NOT AVAILABLE".**

# 2. Hospital Resource Management:

- **Room:**

This class is used to create the IDs of each room and to keep track of their availability status.

***Attributes:***

- **RoomID:** An integer that represents the room ID, where each room has an ID of 3 integers, the first number will represent the room floor and the rest 2 numbers will represent the count of this room in this floor.
- **Availability:** A Boolean variable that indicates the availability of each room in the hospital. If the availability of a specific room is true then this room is available, and if it is false then this room is not available.

***Functions needed in the class:***

- **Room(roomID, availability):** this method is the constructor of the class, and it will be used in the Rooms class to create the rooms in the hospital with their ID's and the status of availability.
- **getAvailability():** getter of the variable availability.

- **setAvailability(availability):** setter of the variable availability, where it will be used to set the status of availability of each room (true, false).

- **Rooms:**

This class is used to create the rooms in the hospital with their IDs and to keep track of all the available rooms and non-available rooms. It also allows the user to swap rooms between patients whenever needed and to check if a *specific* room is available or not.

***Attributes:***

- **rooms:** ArrayList of type Room, where all the rooms with their IDs and status of availability will be saved in it.
- **AvailableRooms:** ArrayList of type Room, where all the available rooms will be saved in it.
- **NotAvailable:** ArrayList of type Room, where all non-available rooms will be saved in it.
- **map:** HashMap<Integer, Integer>, where the roomID will be the key and the patientID is the value, and it is used in the swapRooms function.

***Functions needed in the class:***

- **CheckAvailableRoom(RoomID):** this function will take specific room ID to check whether this room is available or not.
- **getAvailableRooms ():** The system should display the user all the available rooms, thus this method is created to return the available rooms; that are not full of patients, the method first will loop over all the rooms in the hospital, if the availability status for a room is true, then the room is available, thus it will be added to the list of available rooms and then this list will be returned.

- **getNotAvailableRooms():** The system should display the user all the rooms that are not available, thus this method is created to return the Not available rooms; that are full with patients, the method first will loop over all the rooms in the hospital, if the availability status for a room is false, then the room is not available, thus it will be added to the list of non-available rooms and then this list will be returned.
- **swapRooms(roomID1, patientID1, roomID2, patientID2):** Sometimes, the user will need to swap rooms between the patients, thus this method is created to do this job. The user will specify the two rooms with their corresponding patient's IDs they want to swap. Then, the method will use a hashmap, first it will remove roomID1 (mapped to patientID1) from the hashmap and it will add it again to the hashmap but with mapping it to patientID2. After that, roomID2 (mapped to patientID1) will be removed from the hashmap and re-added to the hashmap but with mapping it to patientID1.

➢ **InventoryManagement:**

This class is used to manage the work of the inventory department, where it will allow the user to add, remove, and search for a product in addition to displaying all the products with their quantity.

***Attributes:***

- **product:** Hashmap<String, Integer>, where all the products with their relative quantity will be saved in it.

***Functions in this class:***

- **add_product(productName, quantity):** this function will take the product name with its quantity that the user wants to add, and it will add it to the system with updating the quantity number of this product.
- **remove_product(ptoduct_to_remove, remove_quantity):** this function will the take name of the product with its relative quantity the user wants to remove (use), and it will remove the relative quantity of the product the user wants to remove, and if the quantity the user wants to remove is greater than the available quantity, the system will print to the user the statement **"THE QUANTITY YOU REQUESTED IS GREATER THAN THE QUANTITY AVAILABLE"** and if the product is not available the system will print **"PRODUCT NAME"** + **"IS UNAVAILABLE".**
- **search(itemToSearch):** this method will take the name of the product the user wants to search for, and it will return the quantity available of this product and if the quantity is equal to 0 the method will ask the user if he/she wants to buy from this product and add it to the system.
- **displayProducts():** this method will display all the products with their relative quantity for each product whenever the user request the list of available products with their relative quantities.

## ➢ Laboratory:

This class will be responsible for managing all the patient information related to the tests results and appointments.

***Attributes:***

- **TestResult:** Boolean datatype, defines whether a test results has done "true" or not "false".

- **Test:** Hashmap<String, Integer>, where it stores the name of the test as a key value and the price of test as the value.

***Functions needed in the class:***

- **Laboratory ():** represents the constructor of the class.
- **DisplayTestAppointmentSheet():** displays the available and non-available appointments of each test.
- **TakeAppointment(Patient Name, Patient Last Name, Patient Phone Number, Date, Time):** this function will allow the user to take appointment for the patient. If the date and time entered are not reserved, the appointment will be reserved, and a statement **"APPOINTMENT RESERVED SUCCESSFULLY"** will be generated. Otherwise, the statement **"THIS APPOINTMENT IS RESERVED"** will be generated. If the entered date and time does not exist in the schedule, the statement **"THE DAY AND/OR TIME ENTERED DOES NOT EXIST".**
- **GenerateReciept(TestType):** this method will take the type of test to be performed for the patient. The method will search in a hashmap for the price of each test and then return the price.

# 3. Billing and Financial Management:

- **FinancialFlows:**

The financial flows class represents all the data about the financial flows.

***Attributes:***

- PatientPayments: Double datatype, represents the payments of the patient to the hospital.
- ProductsPaymnets: Double datatype, represents the amount of money paid by the hospital to buy products needed by the hospital.

***Functions needed in the class:***

- **FinancialFlows():** represents the constructor of the class.
- **CalculateProfits():** this method will calculate the total amount of payments paid by all the patients in the hospital and sums up the total amount of payments. The total payments obtained will be considered as the profit of the hospital.
- **CalculateTotalWages():** this method will calculate the total salaries of all the employees in the hospital and return the total salaries obtained. The total salaries obtained will be considered as the wages paid by the hospital.
- **CalculateNetProfits():** this method will calculate the total net profit of the hospital by calling the previous two methods and decrementing their values. Then it will return the final value.

## ➢ Invoice:

The Invoice class represents an invoice document that is delivered to the patient after satisfying all the medications that the patient needs.

***Functions needed in the class:***

- **SearchPatientInfo(int ID, String name):** this function access the patient table and search for the ID provided. If the ID is valid, the method will check if the name provided by the patient is

the same name associated to the given ID in the database, if yes, the method will check for some attributes needed and then display them for the users. If the name provided is not related to the ID in the database, the method will return "**THE NAME/ID YOU ENTERED ARE NOT CORRECT, PLEASE ENTER CORRECT INFORMATION".** If the patient ID is not available in the database, the method will return **"ID IS NOT AVAILABLE".**

# 4. Employee Management:

## ➢ EmployeeGData:

Employee General Data class is responsible for collecting all personal information related to the employee to be able to enroll in the hospital.

***Attributes:***

- **FName**: String datatype, represents the First name of the employee.
- **MName:** String datatype, represents the Middle name of the employee.
- **LName**: String datatype, represents the Last name of the employee.
- **Gender:** Character datatype, represent the gender of the employee in which the letter "F" represents a female, and the letter "M" represents a male.
- **PhoneNum:** Integer datatype, represents the phone number of the employee.
- **Nationality:** String datatype, represents the Nationality of the employee.
- **Residence:** EmployeeResidence datatype (discussed below), represents the address info of the employee.
- **DOB (Date of Birth):** Date datatype, represents the birth date of the employee.

- **Speciality**: String datatype, represents what the employee special of.
- **Degree:** String datatype, represents the employee education status.
- **HiringDate:** LocalDate datatype, represents the hiring date employee hired in the hospital.
- **EmpID:** Integer datatype, represents the employee unique ID.

***Functions needed in the class:***

- **EmployeeGData (FName, MNAme, LName, gender, phoneNum, nationality, residence, Date DOB, speciality, degree, tracker)**: this is the constructor method is used to create an object of this class. It takes most of the defined attributes, knowing that tracker is of type EmployeeIDTracker (disscussedbelow).
- **EmployeeGData(EmployeeGData employee):** this is another construct for the same class, it is used to be implemented in the EmployeeHData class.
- **SetEmpID(tracker):** this method will assign automatically new ID for the hired employee. (Tracker is of type EmployeeIDTracker discussed below).
- **getFName ():** this method will get back the First name of the employee.
- **getMName ():** this method will get back the Middle name of the employee.
- **getLName ():** this method will get back the Last name of the employee.
- **getEmpFullName ():** this method will return the employee's full name.
- **getNationality ():** this method will return the employee's nationality.
- **getDOB ():** this method will return the employee's date of birth.
- **getHiringDate ():** this method will return the employee's hired date.
- **getEmpID ():** this method will return the employee's unique ID.
- **getGender ():** this method will return the employee's gender type.
- **setGender (char Gender):** this method will set the employee gender.

- **getPhoneNum ():** this method will return the employee's phone number.
- **setPhoneNum (int phoneNum):** this method will set the employee phone number.
- **getResidence ():** this method will return the employee's address.
- **setResidence (EmployeeResidence residence**): this method will set the employee address.
- **getSpecialty ():** this method will return the employee's specialty.
- **setSpeciality (String specialty):** this method will set the employee specialty.
- **getDegree ():** this method will return the employee's educational degree.
- **setDegree (String degree):** this method will set the employee educational degree.
- **toString ():** this method will return the employee's info in a specified form.

➢ **EmployeeHData:**

Employee Hospital Data class is responsible for collecting and assigning all information for the enrolled employee.

***Attributes:***

- **EmpID:** Integer datatype, represents the employee unique ID.
- **Position:** String datatype, represents the employee position in the hospital.
- **Department:** String datatype, represents the employee working department.
- **SupervisorID:** Integer datatype, represents the employee supervisor unique ID.
- **Salary:** Double datatype, represents the employee salary.

***Functions needed in the class:***

- **EmployeeHData (EmployeeGData employee, Position, Department, SupervisorID, Salary:Double):** this is the constructor method, it is used to create an object of this class.

- **searchEmployeeByFName(Hashtable<Integer, EmployeeHData> table, String FName):** this method search the data structure that saves all employee info by employee first name, and return list of all employees with the specified FName.
- **getSalary ():** this method will return the employee's salary.
- **getEmpID ():** this method will return the employee's unique ID.
- **getPosition ():** this method will return the employee's position.
- **setSalary ():** this method will set the employee salary.
- **setPosition ():** this method will set the employee position in the hospital.
- **getDepartment ():** this method will return the employee's working department.
- **setDepartment ():** this method will set the employee's working department.
- **getSupervisorID ():** this method will return the employee's supervisor's unique ID.
- **setSupervisorID ():** this method will set the employee's supervisor's unique ID.
- **toString ():** this method will return the employee's info in a specified form.

## ➢ EmployeeResidence:

Employee Residence class gather employee addresses assigned to their ID.

***Attributes:***

- **empID:** Integer datatype, represents the employee unique ID.
- **streetAddress:** String datatype, represents the employee street address.
- **cityName:** String datatype, represents the employee city name.
- **countryName:** String datatype, represents the employee country residence name.

***Functions needed in the class:***

- **EmployeeResidence (empID, streetAddress, cityName, countryName):** this is the constructor method, it is used to create an object of this class.
- **getEmpID ():** this method will return the employee's unique ID.
- **getStreetAddress ():** this method will return the employee's street address.
- **setStreetAddress (String streedAddress):** this method will set the employee street address.
- **getCityName ():** this method will return the employee's city name.
- **setCityName (String cityName):** this method will set the employee city name.
- **getCountryName ():** this method will return the employee's country name.
- **setCountryName (String countryName):** this method will set the employee country name.
- **toString ():** this method will return the employee's info in a specified form.

- **EmployeeIDTracker:**

Employee ID Tracker class is for keeping all employees ID tracked with no ID repetition to avoid system confusion and for employee security.

***Attributes:***

- **IDTable:** HashTable datatype, represents the employees ID with their full name.
- **lastNumber:** int datatype, represents the last 4-digits was concatenated to the employee ID and then inserted to the IDTable, so that we can add the new ID based on the last one to avoid any collision in the hash table.
- **Date:** LocalDate datatype, represents the date data is inserted to the database, and it is used in generating the employee ID.

***Functions needed in the class:***

- **EmployeeIDTracker (IDTable, lastID, Date):** this is the constructor method, it is used to create an object of this class.
- **generateID (String hiringDate, String fullName):** this method will generate a unique ID for every employee in the hospital and it is done by getting the hiring date that the employee gets hired to the hospital and concatenating to it 4-digits that are unique for one day depending on lastIDigit defined before. (i.e., HiringDate "20210512" lastDigit "0013" than the ID will be empID "202105120014").

➢ **EmployeeShiftScheduling:**

Employee shift scheduling class will be implemented to help the hospital management in setting dates and time to all employees without any time conflict and confusion.

***Attributes:***

- **empID:** Integer datatype, represents the employee unique ID.
- **department:** String datatype, represents the employee's working department.
- **shiftType:** String datatype, represents the employee's shifting type (i.ie., dayshift, …)
- **needWorkers:** Boolean datatype, represents if the department need workers at a specific shift.
- **covidDep:** DepartmentWorkerNeeded datatype (discussed below), represents the department neediness of workers during shift.
- **admDep:** DepartmentWorkerNeeded datatype (discussed below), represents the department neediness of workers during shift.
- **kitDemp:** DepartmentWorkerNeeded datatype (discussed below), represents the department neediness of workers during shift.

- **covidDepTable:** Hashtable datatype, represent all employee scheduled to the covid department.
- **admDepTable:** Hashtable datatype, represent all employee scheduled to the admission department.
- **kitDempTable:** Hashtable datatype, represent all employee scheduled to the kitchen department.

***Functions needed in the class:***

- **EmployeeShiftScheduling (empID, department, shiftType):** this is the constructor method, it is used to implement an object of this class.
- **setSchedule ():** this method set schedule for a specific employee based on his ID and working department by assigning him/her to another method such as setScheduleCoviDep().
- **setScheduleCovidDep ():** this method is private and only accessed by the class itself, it assigns employee with the needed shift and save his/her data in the hash table defined before. This method will not allow same employee to be assign more than once for same day. Also, this method will help in managing department workers by assigning only the needed worker to it to exceeding nor having less.
- **N.B:** all other department will have same method as the covid one (i.e., setSchedulCoviDep()).

# 5. Emergency Covid – 19 Department:

- **Affected_Patient:**

The Affected_Patient class represents the personal and Covid – 19 information about the affected patient.

***Attributes:***

- **Patient_ID:** A numeric identifier of 9 numbers representing the employee's ID.
- **First_Name:** String datatype, represents the first name of the patient.
- **Last_Name**: String datatype, represent the last name of the patient.
- **Gender:** Char datatype, represent the gender of the patient in which the letter "F" represents a female, and the letter "M" represents a male.
- **Date_of_Birth:** String datatype, represents the date of birth of a patient.
- **Financial_Status:** String datatype, represent the type of insurance that a patient has. It might me set to null if the patient does not have any insurance type.
- **Phone_Number**: int datatype, represents an 8 digits number that specifies the patient personal number.
- Address: String datatype, represent the address that a patient resides in.
- **PCR_Results:** string datatype, represents the PCR results of the patient who did the PCR test. The result will be either "Positive" or "Negative".
- **Day_of_Affection:** String datatype, represents the date in which the affected patient was affected in.
- **CT_Level:** int datatype, represent the CT level of the affected patient.
- **Date_of_Entry:** String datatype, represents the date that the patient was admitted to the hospital.
- **Allergies**: String datatype, an array list that include the type of allergy that a patient has. Initially the list will be empty and on confirmation of allergy existence, the user will add elements to the list.

- **Equipment's:**

The Equipment's class represents the required equipment's for the carrying disease.

***Attributes:***

- **Oxygen_Machines:** int datatype, represents the number of available oxygen machines.
- **Sanitizers:** int datatype, represents the number of the sanitizers available in the hospital.
- **PCR_Equipment:** int datatype, represents the number of the PCR equipment's available in the hospital.

***Functions in the class:***

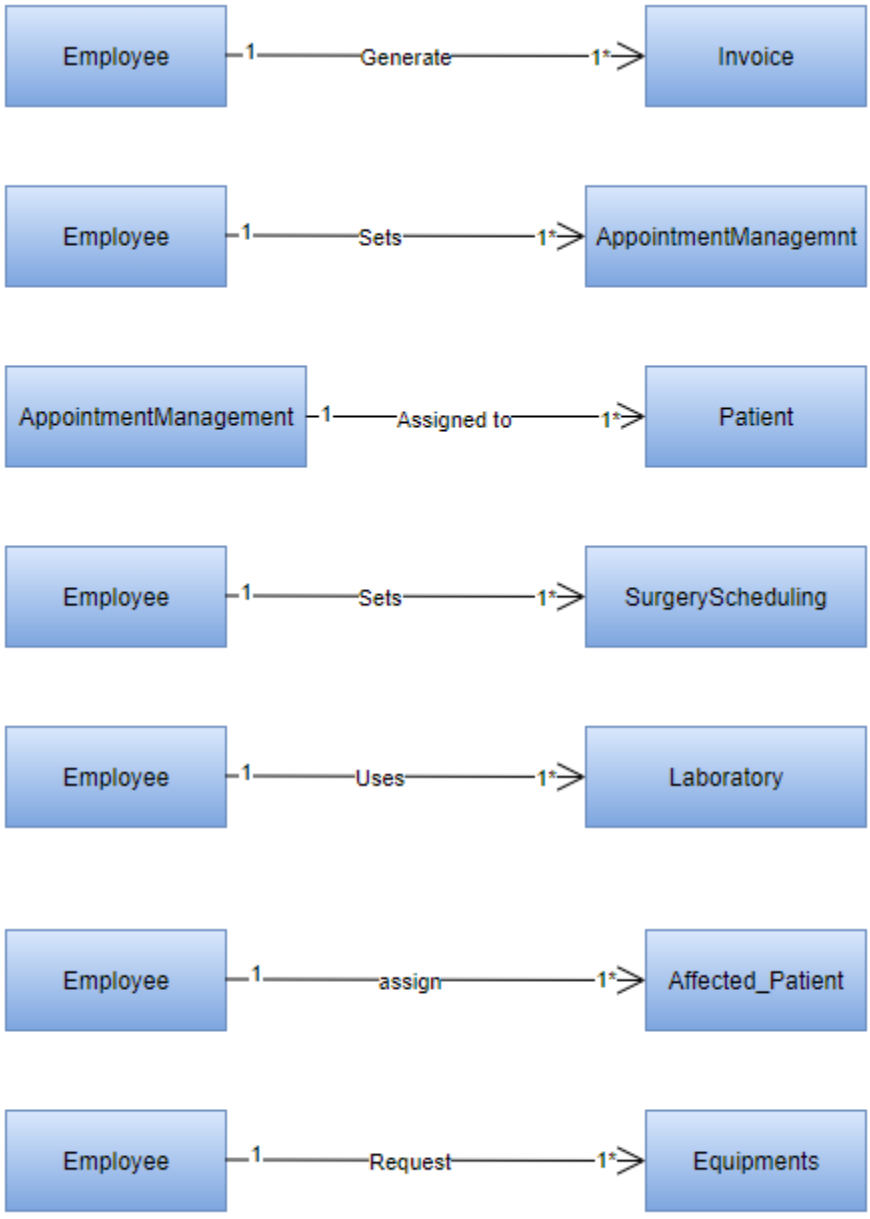
- **Is_Available_Oxygen():** This function returns a Boolean value "true" if there are available oxygen machines in the hospital. Otherwise, the method will return false.
- **setOxygen_Machines(int Oxygen_Machines):** int datatype, it sets the quantity of the oxygen machines available in the hospital.
- **getOxygen_Machines():** int datatype, it gets the number of the available oxygen machines in the hospital.
- **Is_Available_Sanitizers():** This function returns a Boolean value "true" if there are available sanitizers in the hospital. Otherwise, the method will return false.
- **getSanitizers():** int datatype, it gets the number of the available sanitizers in the hospital.
- **setSanitizers(int sanitizers):** int datatype, it sets the quantity of sanitizers available in the hospital.
- **Is_Available_PCR_Equi():** This function returns a Boolean value "true" if there are available PCR equipment's in the hospital. Otherwise, the method will return false.
- **getPCR_Equipments():** int datatype, it gets the number of the available PCR equipment's in the hospital.

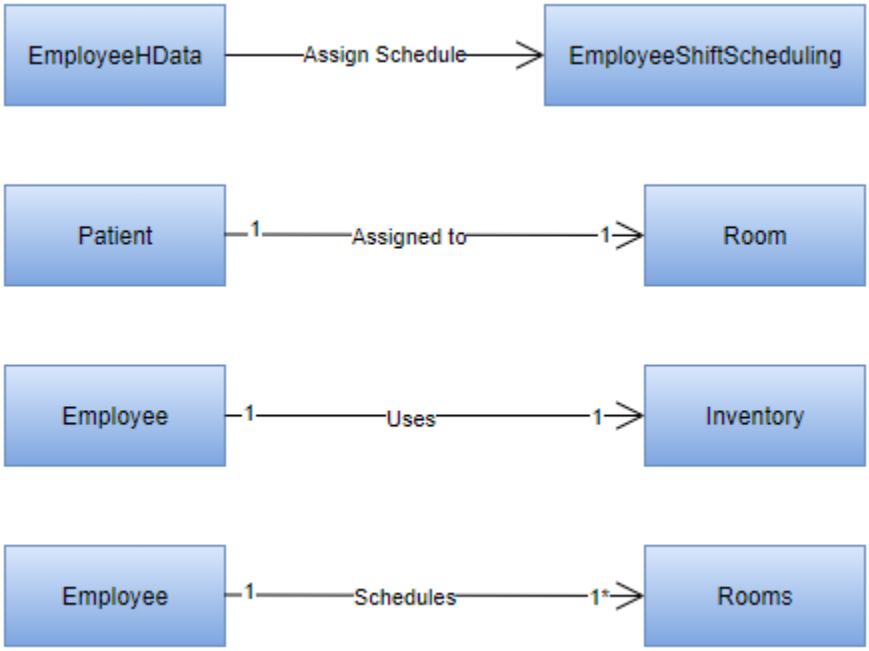
- **SetPCR_Equipments(int PCR_Equipments):** int datatype, it sets the quantity of PCR Equipment's available in the hospital.

# II. Class Relationships:

## 1. Association:

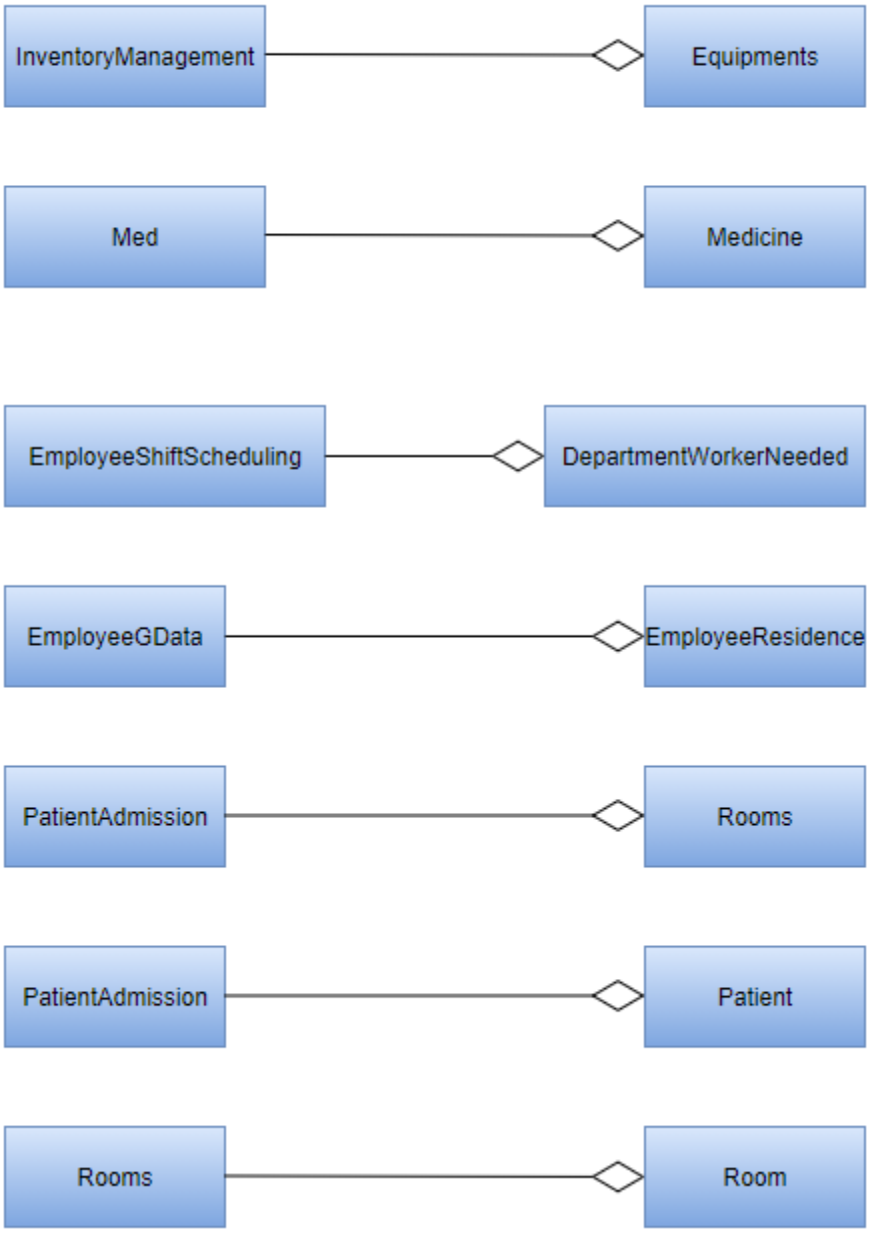
Association defines the multiplicity between objects. Association can be one-to-one, one-to-many, many-many, and one-to-many.

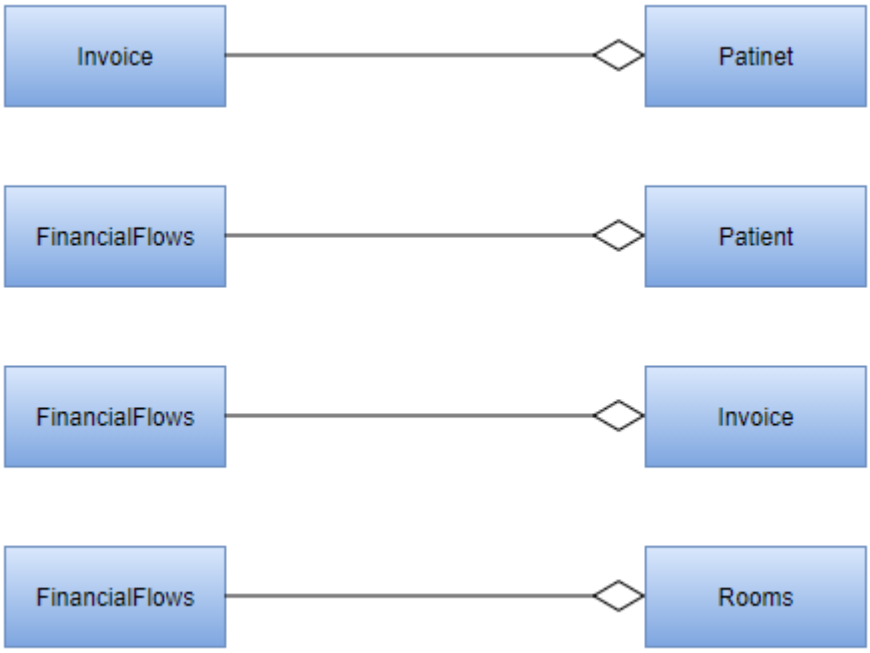



EmployeeHData
Assign Schedule
EmployeeShiftScheduling
Patient
1
Assigned to
1
Room
Employee
1
Uses
1
Inventory
Employee
1
Schedules
1*
Rooms

## 2. **Aggregation:**

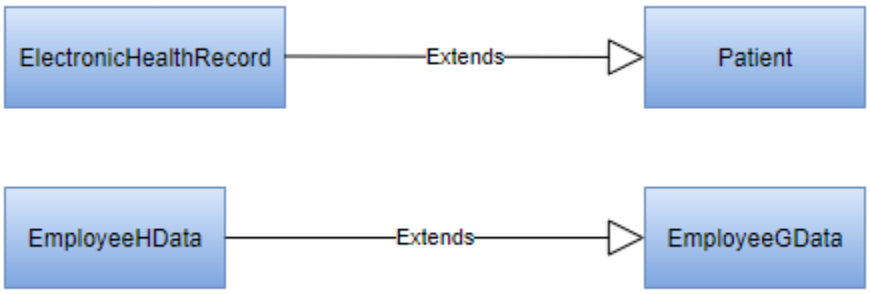
Aggregation is a type of association. It is the case where an object ‘has-a’ another object.





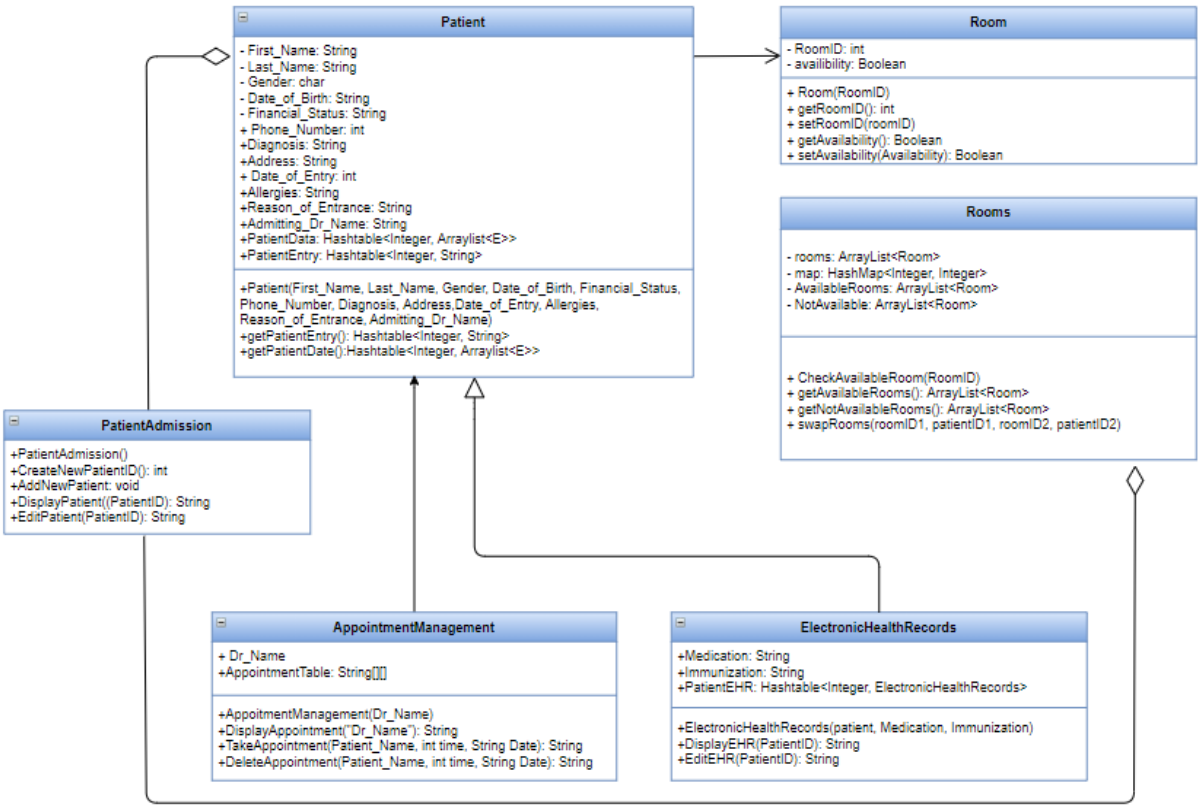

## 3. Generalization:

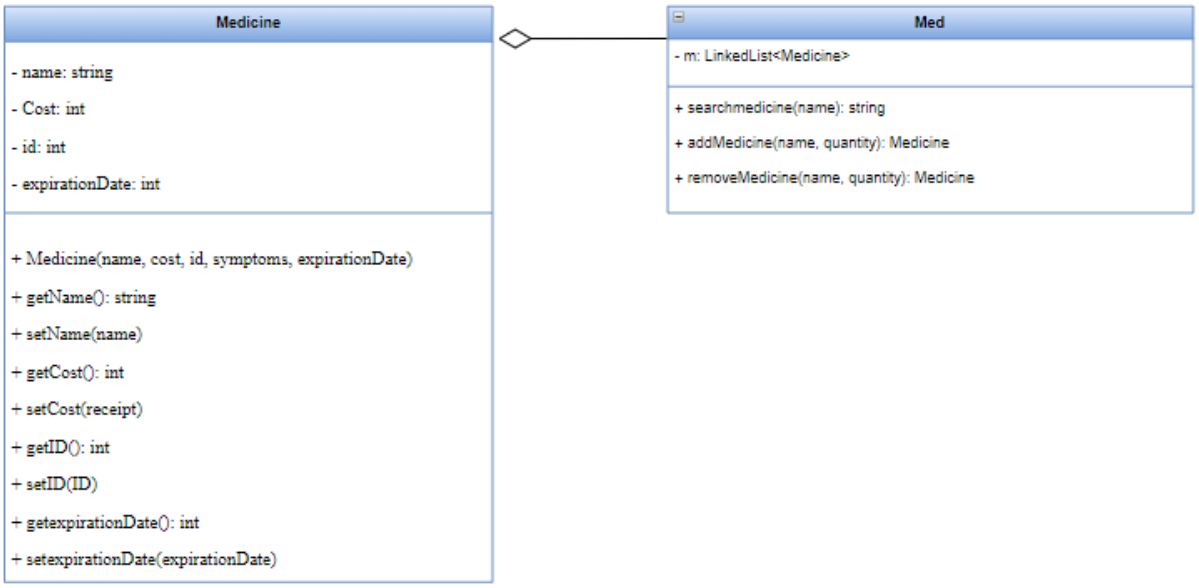
Generalization is defined when one class acquires the methods and fields from another class. One class will be super class and the other will be subclass.



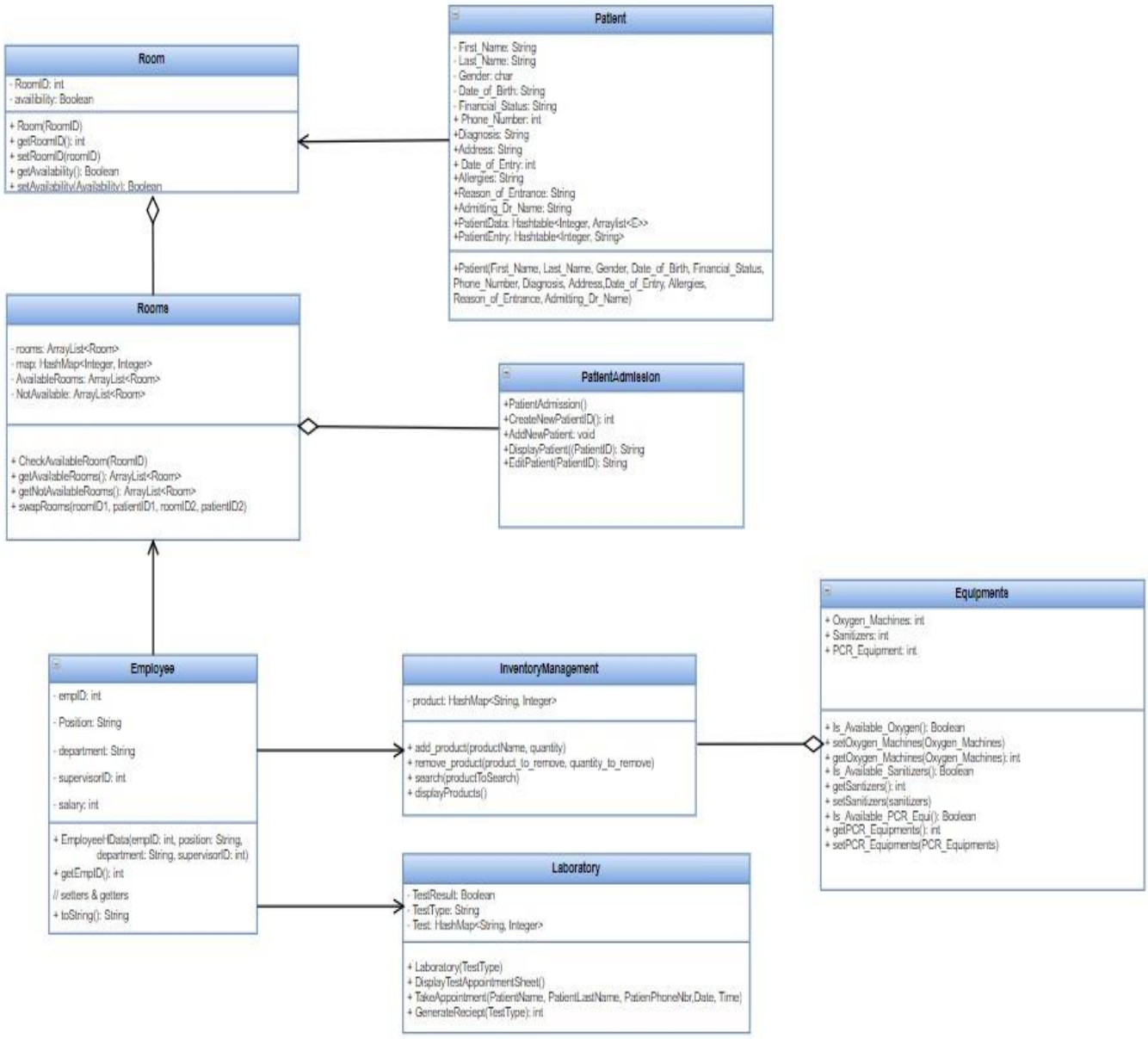
# III. Class Diagram:

## 1. Hospital Services Management:

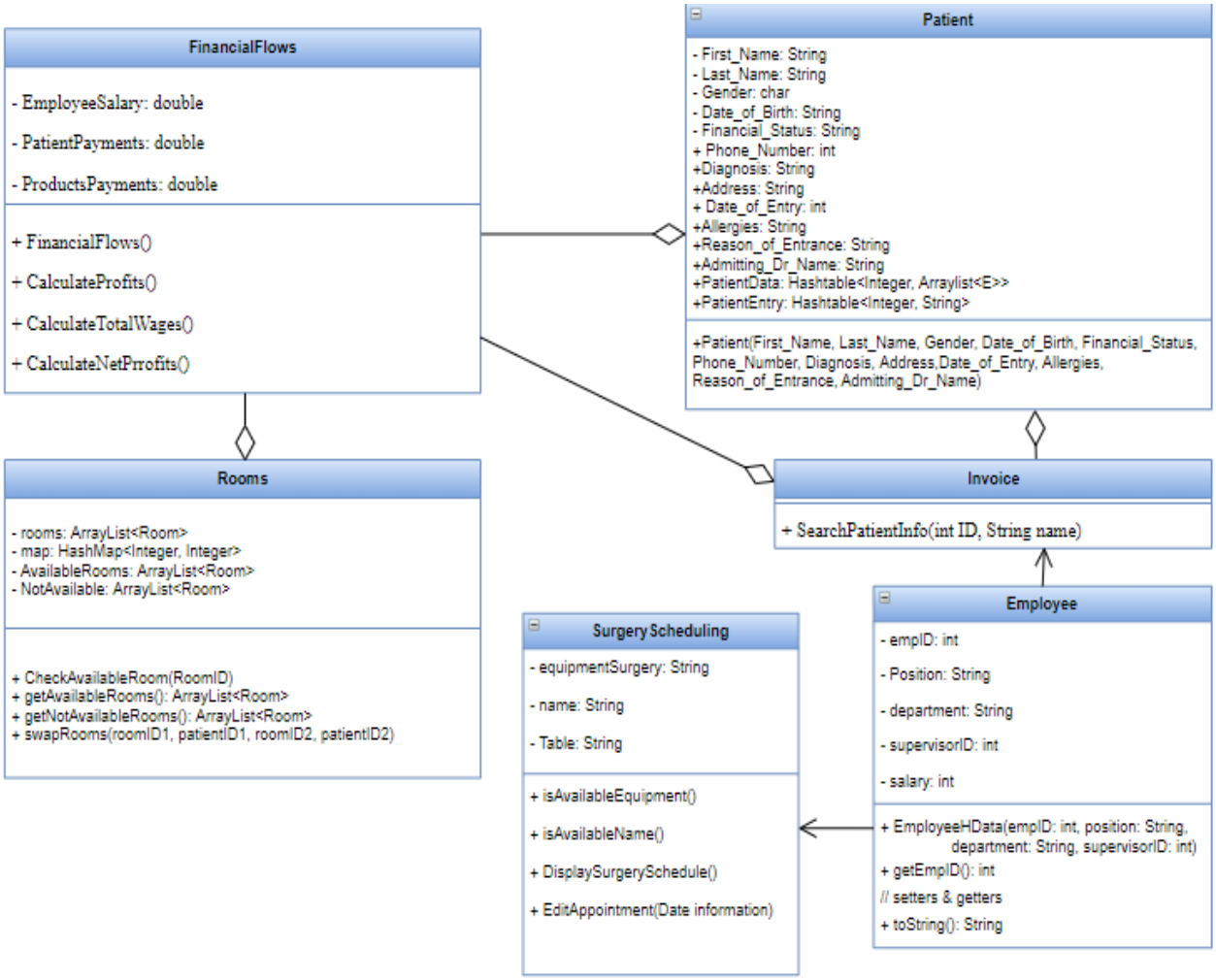



Medicine
- name: string
- Cost: int
- id: int
- expirationDate: int
+ Medicine(name, cost, id, symptoms, expirationDate)
+ getName(): string
+ setName(name)
+ getCost(): int
+ setCost(receipt)
+ getID(): int
+ setID(ID)
+ getexpirationDate(): int
+ setexpirationDate(expirationDate)
Med
- m: LinkedList<Medicine>
+ searchmedicine(name): string
+ addMedicine(name, quantity): Medicine
+ removeMedicine(name, quantity): Medicine

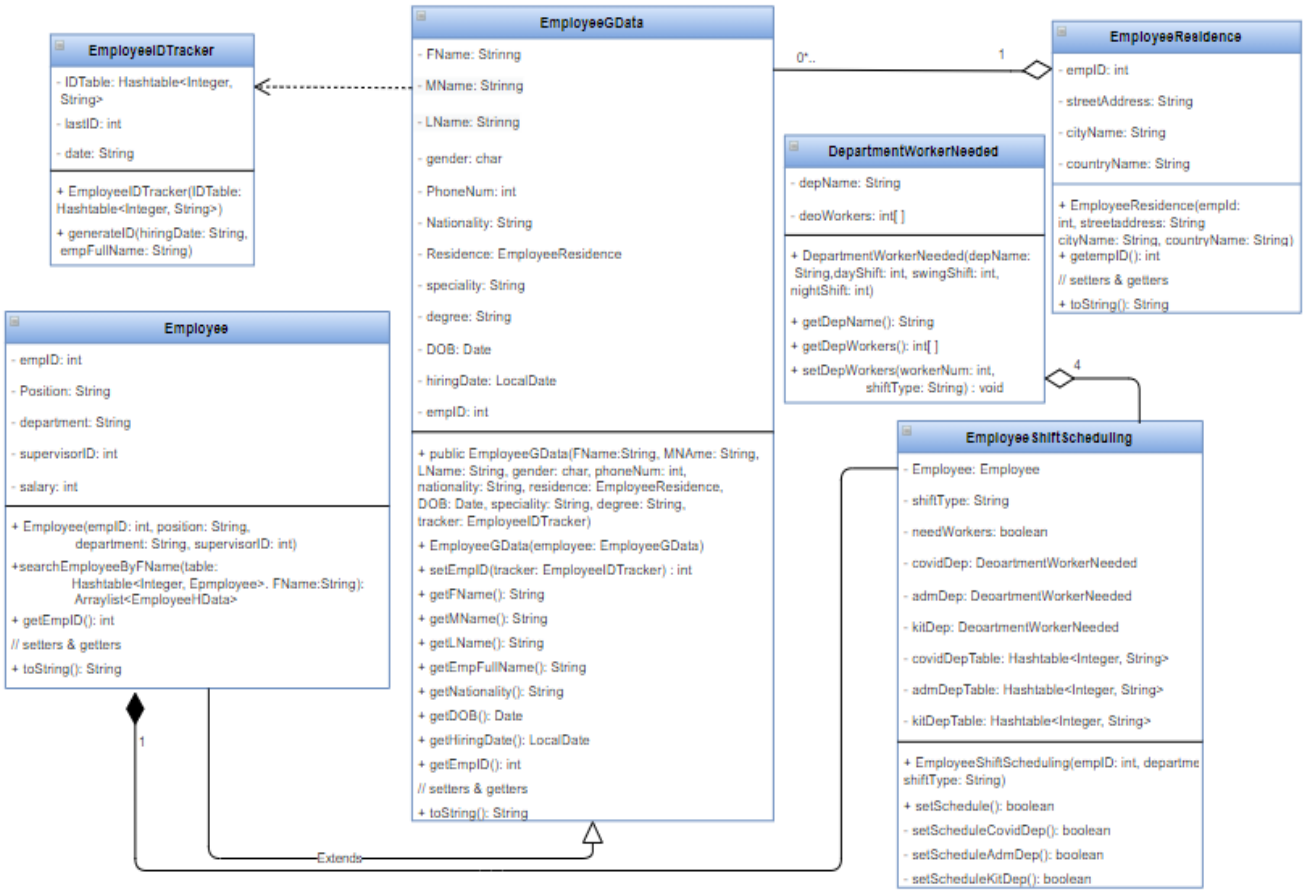
## 2. Hospital Resource Management:



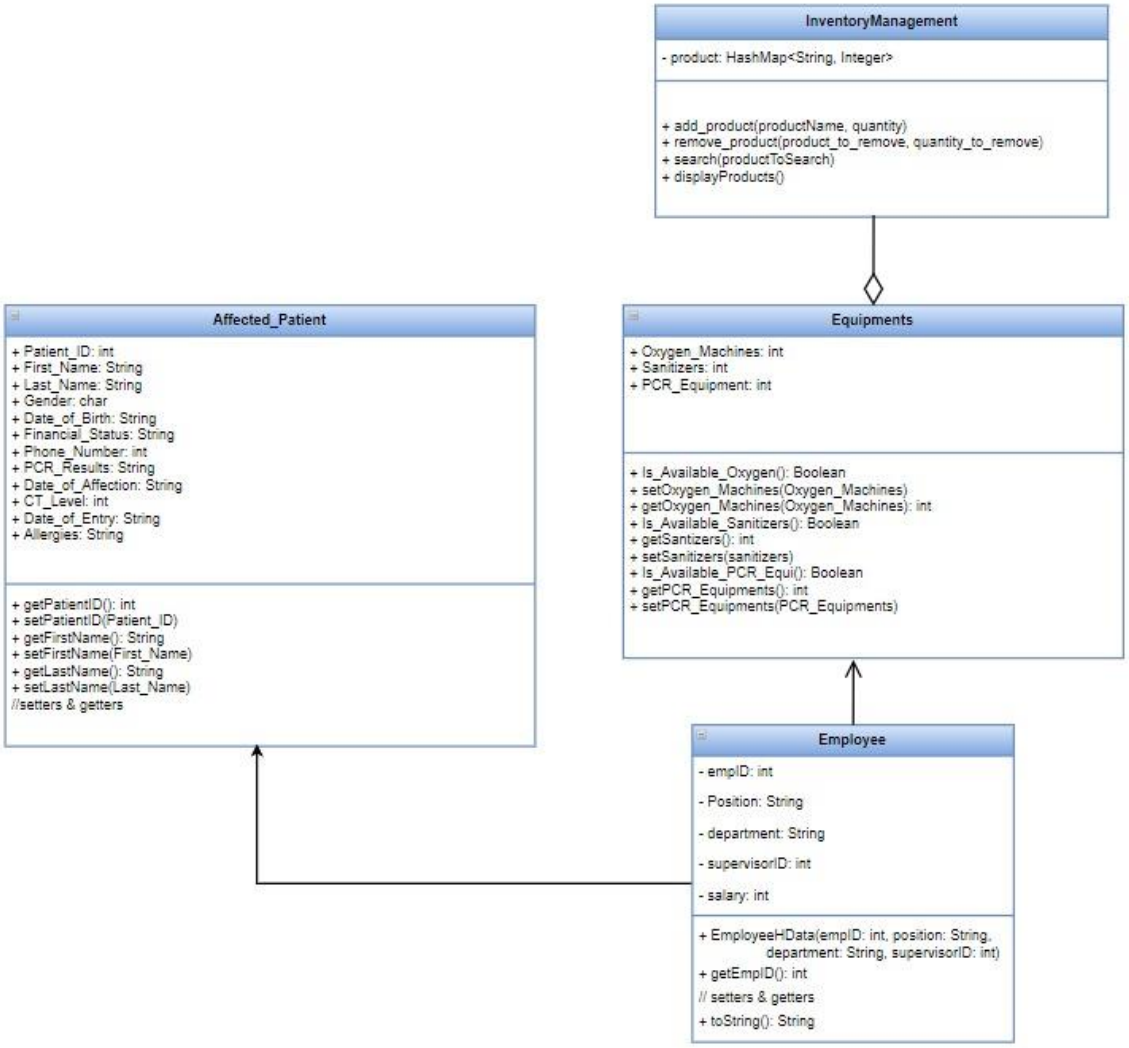
# 3. Billing and Financial Management:



# 4. Employee Management:



# 5. Emergency Covid – 19 Department:



# IV. Algorithms:

## 1. Hospital Services Management:

### a. Patient:

```
public class Patient{


Patient personal information: First_Name, Last_Name, Gender, Date_of_Birth,
Financial_Status, Phone_Number, Diagnosis, Adress

Patient Medical Information: Date_of_Entry, Allergies, Reason_of_Entrance,
Admitting_Dr_Name


public HashTable<Integer, Patient> PatientList;

public HashTable<Integer, String> PatientEntry;


//constructor

      public Patient (personal information, medical information){

       this.personal Information = personal Information;

       this.Medical Information = Medical Information;

      }


getPatientList(){…}

getPatientEntry(){…}


}
```

**b. PatientAdmission:**

```
Public class PatientAdmission{

Calendar calendar = Calendar.getInstance();
Random random = new Random();
Rooms available = new Rooms();


public int Create_New_Patient_ID() {
    //CREATE ID FOR THE NEW PATIENT
    int year = get the year from the clalander
    int num = generate a random number of 5 digits
    int ID = concatenate num with year

    if (if ID exists in the system) {

    while(ID does not exist in the system) {
            num = generate a random number of 5 digits
            ID = concatenate num with year;
            }
            return ID;
    }

public String Add_New_Patient(Patient Personal Information, Patient Medical Information,
int Patient_Entry_Date) {

    if (if there are available rooms in the hospital){
    available.setAvailability = false;

    int ID = Create_New_Patient_ID();
```

```
Patient patient = new Patient(Patient Personal Information, Patient Medical Information);

Call the HashTable "PatientData" created in Patient Class
-> add the ID as key value to the table and the patient object as a value to the table

Call the HashTable "PatientEntry" created in Patient Class
-> add the ID as key value to the table and the Patient_Entry_Date as a value to the table

return "Admission Done!";
    }
    else
        return "No Vacancies are Available";
}

public String Display_Patient(int Patient_ID){

    if(Patient is found in the HashTable "PatientData") {
        return medical and personal info of this Patient
    }
    else
    {
        return “PLEASE ENTER A VALID ID”;
    }
}

public Edit_Patient(int Patient_ID ){
    if (Patient is found in the HashTable "PatientData") {
    while (! editing is done){
        attribute = ask user for attribute to be changed
```

```
                newValue = ask user for new value for attribute
                oldValue(attribute) = newValue
        }
        return “UPDATED SUCCESSFULLY”;
} else{
        return “PLEASE ENTER A VALID ID”;
                }
}
}
```

**c. ElectronicHealthRecord:**

```
public class ElectronicHealthRecord{
public String Medication;
public String Immunization;
public HashTable<Integer, ElectronicHealthRecord> PatientEHR;

public String EHR_Display(ID){
        if(Patient_ID exists in the PatientEHR){
        return all the medical and personal information found in the EHR of
the patient;
        }
else
                return “PLEASE ENTER A VALID ID”;
}

public String Edit_EHR(ID){
        if(Patient_ID exists in the PatientEHR){
        while ( ! editing is done){
                attribute = ask user for attribute to be changed
```

```
                newValue = ask user for new value for attribute
                oldValue( attribute ) = newValue
        }
        return “UPDATED SUCCESSFULLY”;
    }
    else{
        return “PLEASE ENTER A VALID ID”;
                }
        }
    }
```

**d. AppointmentManagement:**

```
public class AppointmentManagement{
public String Dr_Name;
public String table[][] = the days of the week and the times available each day;

    //constructor
public Appointment_Management(String Dr_Name) {...}

public void DisplayAppointments() {
    Print the table using System.out.format(...);
}

public String takeAppointment(date info, patient info) {
    if (date is valid) {

    int index = 0; //change this index according to the "day" entered
    int index2 = 0;   //change this index according to the "time" entered
```

```
    if (if no appointment is in table[index][index2]) {
            table[index][index2] = reserve this appointment for the patient;
            return "APPOINTMENT RESERVED SUCCESSFULLY!";
    }
    else
            return "THIS APPOINTMENT IS RESERVED, PLEASE CHOOSE ANOTHER
TIMING!";
    }
    else
            return "THE DAY AND/OR TIME YOU ENTERED IS NOT CORRECT";
    }

public String DeleteAppointment(date information, patient information) {

    if (date entered is valid) {

    int index = 0; //change this index according to the "day" entered
    int index2 = 0;   //change this index according to the "time" entered

    if (if the patient took appointment at the entered time) {
            table[index][index2] = set to empty appointment;
            return "APPOINTMENT DELETED SUCCESSFULLY!";
    }
    else
            return "NO SUCH APPOINTMENT EXISTS";
    }
    else
            return "THE DAY AND/OR TIME YOU ENTERED IS NOT CORRECT";
    }
}
```

**e. Surgery_Scheduling:**

```
class Surgery_Scheduling{
String equipmentSurgery;
String name;
boolean isAvailableequipmentSurgery(){ //checks if the required equipments for the surgery are available
String array[];
for int i = 1 to length of array{
if(array [i] == equipmentSurgery){
return true;
}
else{
return false;
}
}
}
boolean isAvailableName(){ //checks if the employee responsible for the surgery is available
String array[];
for int i = 1 to length of array{
if(array [i] == name){
return true;
}
else{
return false;
}
}
DisplaySurgerySchedule(){
int date, time;
```

```
String Table [][] = new String [date][time];
prints the table to the user
}
EditAppointment(Date information){
if(Date information is Valid){
add, delete or update the date information
}
}
}
```

**f. Medicine:**

```
class Medicine {
String name;
Int cost;
int id;
int expirationDate;
public Medicine(String name, int cost, int id, int expirationDate){
super();
this.name = name;
this.cost = cost;
this.id = id;
this.expirationDate = expirationDate;
}
String getName() {
return name;
}
void setName(String name) {
this.name = name;
}
```

```
int getCost() {
return cost;
}
void setCostt(int cost) {
this.cost = cost;
}
int getId() {
return id;
}
void setId(int id) {
this.id = id;
}
int getExpirationDate() {
return expirationDate;
}
void setExpirationDate(int expirationDate) {
this.expirationDate = expirationDate;
}
}
```

**g. Med:**

```
class Med {
LinkedList <Medicine> m;
Medicine searchMedicine(String name) {
Iterator <Medicine> iter = m.iterator();
while (iter.hasNext()) {
Medicine m = iter.next();
if (m.getName().equals(name)) {
String med;
int medicine = 1;
```

```
switch(medicine) {
case 1: med = "Name: " + m.getName();
case 2: med = "Receipt: " + m.getReceipt();
case 3: med = "Id: " + m.getId();
case 4: med = "Symptoms: " + m.getSymptoms();
case 5: med = "Expiration Date: " + m.getExpirationDate();
break;
default: break;
}
}
else{
print to the user “THE MEDICINE DOES NOT EXIST”
}
}
return null;
}
void addMedicine(String name, int quantity) {
this.name.add(name);
while(iter.hasNext()){
Medicine m = iter.next();
if(m.getName().equals(name)){
add the quantity to the initial quantity of the medicine
}
else{
add a new medicine with a given quantity
}
}
}
void removeMedicine(String name, int quantity) {
while(iter.hasNext()){
```

Medicine m = iter.next();

if(m.getName().equals(name)){

if(isFound(quantity){

check if quantity greater than the requested medicine

this.name.remove(name);

}

else{

print to the user “THIS QUANTITY IS NOT AVAILABLE, PLEASE ENTER AVAILABLE QUANTITY”

}

}

else{

print to the user “THE MEDICINE IS NOT AVAILABLE”

}

}

}

# 2. Hospital Resource Management:

## a. InventoryManagement:

Public class InventoryManagement{

HashMap<String, Integer> *products* = new HashMap<>(); // string: name of the product, Integer: quantity of the product

public void add_product (productName, quantity){

if(productName is available in *products* hashmap){

int new quantity = initial quantity + quantity

remove productName from *products*

add productName with its new quantity to *products*

}

else{

Add productName with its quantity to products

}

}

Code: (for the previous pseudo code to make it clearer):

```
public void add_product(String productName, int quantity){
        if(products.containsKey(productName)){
            int newquantity;
            newquantity = (products.get(productName) + quantity);
            products.remove(productName);
            products.put(productName, newquantity);
        }else{
            products.put(productName, quantity);
        }

    }
public void remove_product(product_to_remove, remove_quantity){
if(quantity of the productName in products hashmap == 0){
notify the user that product_to_remove is unavailable
}else{
Calculate remained quantity = initial quantity – remove_quantity
    If(remained quantity >= 0){
    Remove product_to_remove with the initial quantity from products hashmap
    Add product_to_remove with the remained quantity to the hashmap
}
}else
if(remained quantity < 0){
    Notify the user that the quantity requested to be removed is greater than the Quantity available
```

```
    }
}
```

Code: (for the previous pseudo code to make it clearer):

```
public static void remove_product(String product_to_remove, int remove_quatity){


    if(products.get(product_to_remove) == 0){
            System.out.println(product_to_remove + "is UNAVAILABLE");
    }else{
    int remainedQuantity;
    remainedQuantity = (products.get(product_to_remove)- remove_quatity);
    if(remainedQuantity >= 0){
            products.remove(product_to_remove); // I removed the quantity totally
            products.put(product_to_remove, remainedQuantity);
    }else if(remainedQuantity < 0){
            System.out.println("The quantity you requested is greater than the quantity
available");
              }
    }
}
public void searchProduct(itemToSearch){
    if(itemToSearch is not available in the products hashmap){
            notify the user that this product is out of quantity
}else{
for each product in products{
if(the name of this product is equal to itemToSearch){
    print the product
}else{
    Notify the user that that itemToSearch is unavailable
    Asks the user if he/she wants to buy this product
    If(the user wants to buy){
```

Name = Asks the user to enter the name of the product if he/she wants to buy

quantity = asks the user to enter the quantity he/she wants to buy

add_product(Name, quantity)

}else if(the user doesn't want to buy the product){

Print to user "no buying request is issued"

}else{

Notify the user to enter a valid value "yes" or "no" to buy the product or not

}

}

Code: (for the previous pseudo code to make it clearer):

```
public void searchProduct(String itemToSearch){
    if(products.get(itemToSearch) == 0){
            System.out.println("OUT OF QUANTITY");
    }else{
            for (String name : products.keySet()) {
            if(name.equalsIgnoreCase(itemToSearch)){
                    System.out.println(name);
    }else{
    System.out.println(itemToSearch + "UNAVAILABLE");
System.out.println("Do you want to buy this product? please answer by \"yes\" or \"no\"");

String x;
    x = scan.next();
    if(x == "yes"){
    System.out.println("Please enter the product name you want to add: ");
    String nameP= scan.nextLine();
    System.out.println("Quantity: ");
    int q = scan.nextInt();
    add_product(nameP, q);
    }else if(x == "no")
```

```
                System.out.println("No buying request is issued!");
        else
                System.out.println("please enter a valid answer(yes or no)");
                        }
                }
        }
}
      }
}
}
public void displayProducts(){
Iterator<Integer> itr = products.values().iterator();
Iterator<String> itr2 = products.keySet().iterator();
while(itr2.hasNext()) {
        while(itr.hasNext()){
                System.out.println(itr2.next() + " " + itr.next());
                }
}
}
```

**b. Room:**

```
int RoomId;
Boolean availability;
getters and setters for both RoomID and Availability
public Room(int roomID, Boolean availability){
        RoomID= roomID
        this.availability = availability;
}// constructor
```

**c. Rooms:**

```
ArrayList<Room> rooms = new ArrayList<Room>();
```

```
ArrayList<Room> AvailableRooms = new ArrayList<Room>();
ArrayList<Room> NotAvailableRooms = new ArrayList<Room>();
HashMap<Integer, Integer> map = new HashMap<Integer, Integer>();
Room r1 = new Room(101, true);//create objects Room and fill them with their IDs and status of availability
public void CheckAvailableRoom(int RoomID){
    if(rooms.get(RoomID).getAvailability == false){
        System.out.println("This room is not available");
    }else{
        System.out.println("This room is available");
    }
}
 public ArrayList<Room> getAvailableRooms() {
    for (int i= 0; i< rooms.size(); i++){
        if(rooms at i is available){
            add room at i to the AvailableRooms arraylist;
            remove room at i from the NotAvailanleRooms arraylist;
        }else{
            Remove room at i from AvailableRooms arraylist;
            Add room at i to the NotAvailableRooms arraylist;
        }
    }

if(AvailableRooms arraylist is empty){
    System.out.println("there are no vacancies in the hospital");
}else{
    For each room in the AvailableRooms arraylist print the room;
}
Return AvailableRooms arraylist
}
```

**Code: (for the previous pseudo code to make it clearer):**

```
public ArrayList<Room> getAvailableRooms() {
        System.out.println("Available Rooms: ");
        for (int i = 0; i < rooms.size(); i++){
            if(rooms.get(i).getAvailability() == true){
                AvailableRooms.add(rooms.get(i));
                NotAvailableRooms.remove(rooms.get(i));
            }
            else{
                AvailableRooms.remove(rooms.get(i));
                NotAvailableRooms.add(rooms.get(i));
            }

        }
        if(AvailableRooms.size() == 0){
            System.out.println("There are no vacancies in the hospital");

        }else{
        for (Room r : AvailableRooms) {
            System.out.println(r);
         }
        }
        return AvailableRooms;
    }
public ArrayList<Room> getNotAvailableRooms() {
    for (int i= 0; i< rooms.size(); i++){
        if(rooms at i is not available){
            add room at i to the NotAvailableRooms arraylist;
            remove room at i from the AvailableRooms arraylist;
        }
```

if(NotAvailableRooms is empty){

System.out.println(“all the rooms in the hospital are available”);

}

}else{

For each room in the NotAvailableRooms arraylist print the room;

}

Return NotAvailableRooms arraylist

}

**Code: (for the previous pseudo code to make it clearer):**

```
public ArrayList<Room> getNotAvailableRooms() {
        System.out.println("Rooms Not Available: ");
        for (int i = 0; i < rooms.size(); i++){
            if(rooms.get(i).getAvailability() == false){
                NotAvailableRooms.add(rooms.get(i));
                AvailableRooms.remove(rooms.get(i));
            }

        }
        if(NotAvailableRooms.size() == 0){
            System.out.println("All rooms in the hospital are available");

        }else{
        for (Room r : NotAvailableRooms) {
            System.out.println(r);
         }
        }
        return NotAvailableRooms;
    }
public void swapRooms(int roomID1, int patientID1, roomID2, patientID2){
    remove roomID1 from the map// map.remove(roomID1);
```

add roomID1 to the map with patientID2 // map.put(roomID1, patientID2);

remove roomID2 from the map// map.remove(roomID2);

add roomID2 to the map with patientID1 // map.put(roomID2, patientID1);

}

**Note: We added the full codes to make it clearer**

# 3. Billing and Financial Management:

## a. FinancialFlows:

```
class FinancialFlows{

//constructor
public FinancialFlows() {…}
public MyExit{
String answer = scan from user

if(answer is “yes”)
System.out.println("Please enter the patient ID");

int getPatientID(){…}

if(if the patient is in the hashtable){
setAvailability() to false
}
int getDate_Of_Entry(){…}
int getDate_Of_Exit(){…}

double CalculateProfits{
```

```
int sum;
while(hashtable is not empty){
sum += arraylist();
return sum;
}
double CalculateTotalWages{
double sum = get all the salaries of the employees
return sum;
}
double CalculateNetProfits{
return CalculateProfits() - CalculateTotalWages();
}
}
```

**b. Invoice:**

```
class Invoice{
HashMap <FinancialFlows, Patient> hm;
public Invoice(){
hm = new HashMap <FinancialFlows, Patient>();
}
int searchPatientInfo(int ID, String name){
if(hm.containsValue(ID)) {
getName();
if(hm.containsKey(name).equals.getName()) {
int patient = 1;
switch(patient) {
case 1:  hm.containsKey("First Name: " + hm.containsKey(getFirstName()));
case 2:  hm.containsKey("Last Name: " + hm.containsKey(getLastName()));
case 3: hm.containsKey("Phone Number: " + hm.containsKey(getPhoneNumber()));
```

```
case 4: hm.containsKey("Address: " + hm.containsKey(getAddress()));
case 5: hm.containsKey("Patient Payments: " + hm.containsKey(getPatientPayments));
break;
default: break;
}
}
else {
if (hm.isEmpty()) {
print to the user "THE NAME/ID YOU ENTERED ARE NOT CORRECT, PLEASE ENTER CORRECT INFORMATION"
} else {
for (Patient patientName : hm.values()) {
print to the user the patientName
}
}
}
}
else{
print to the user “ID IS NOT AVAILABLE”
}
}
}
}
```

# 4. Employee Management:

## a. EmployeeGData:

pseudocode for setEmpID

```
setEmpID(EmployeeIDTracker tracker)
    // convert date to string
```

```
String date = hiringDate.toString
// partition the date into year, month, and day
String y = date.substring from 0 to 4
String m = date.substring from 5 to 7
String d = date.substring from 8 to 10

// group all above in one string
String hiringDate1 = y + m +y

// now call method generateID from EmployeeIDTracker class

int empID = generateID(hiringDate1, empFullName)
```

**b. Employee:**

```
Pseudocode for searchEmployeeByFName
searchEmployeeByFName(hashTable, FName)
    list = new Arraylist

   // iterate over hashtable

   Set setOfIDs = table.key.Set()
   for-each loop (int key : setOFIDs)
       if(table.get(key).getFName == FName)
          add employee to the list

    return list
```

**c. EmployeeIDTracker:**

```
Pseudocode for generateID

generateID(String hiringDate, String empFullName)
    date = today date

    if(date == lastDate used){

      if(last digit inserted < 10)
          // put 00 in the middle to reach 11 digits for the ID (ex. 20210419 + 00 + 2)
           String empID = hiringDate + "00" + last digit inserted
      else if(last digit inserted < 100)
          String empID = hiringDate + "0" + last digit inserted
      else if(last digit inserted < 1000)
```

```
        String empID = hiringDate + last digit inserted

      if(hashTable contains the ID already)
        print("The ID already exist")
        else
          add the ID to the hashTable
          then increment last digit inserted by one

    }else {
      lastDate used = today date
      last digit inserted = 0
      // call the method recursively, in which it reaches the date
      // used is equal to last date used
      generateID(String hiringDate, String empFullName)
    }
```

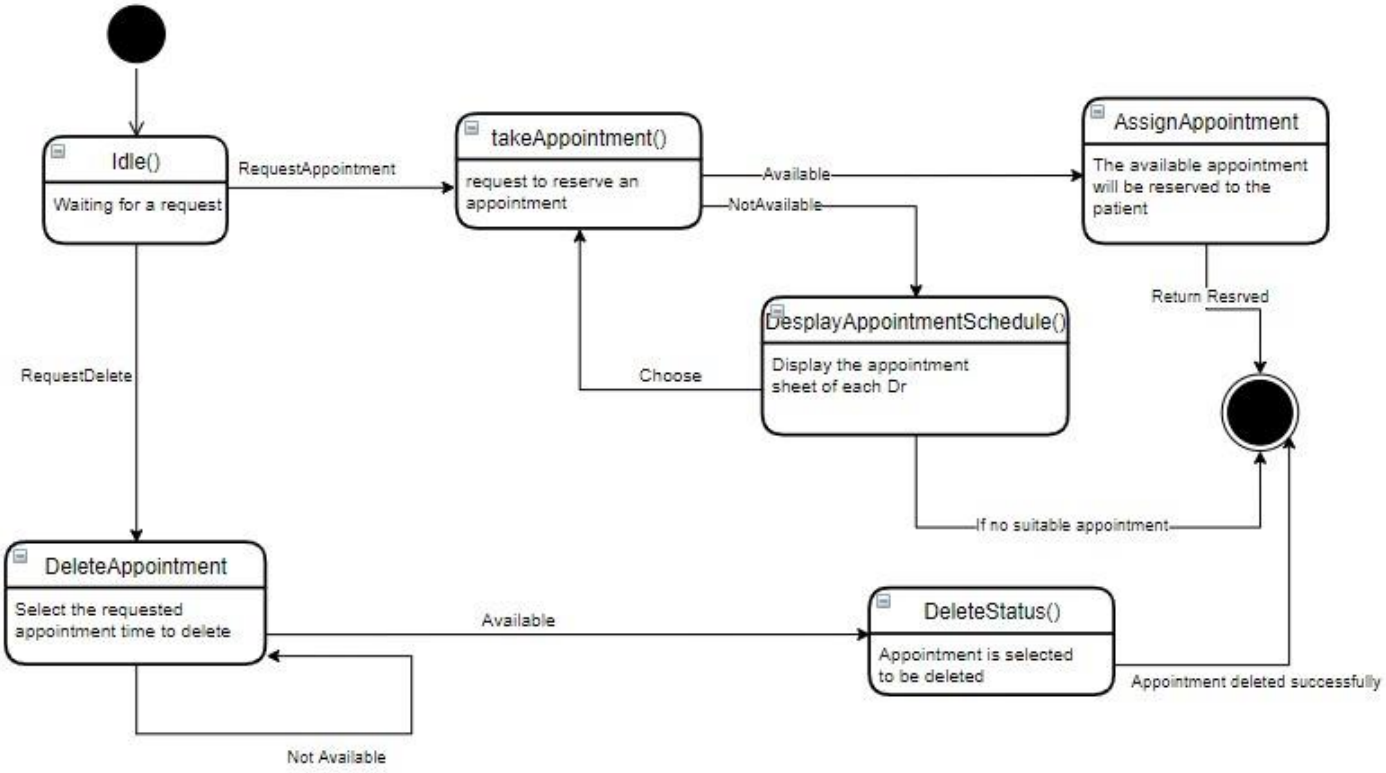
**d. EmployeeShiftScheduling:**

Pseudocode for setScheduleCovidDep

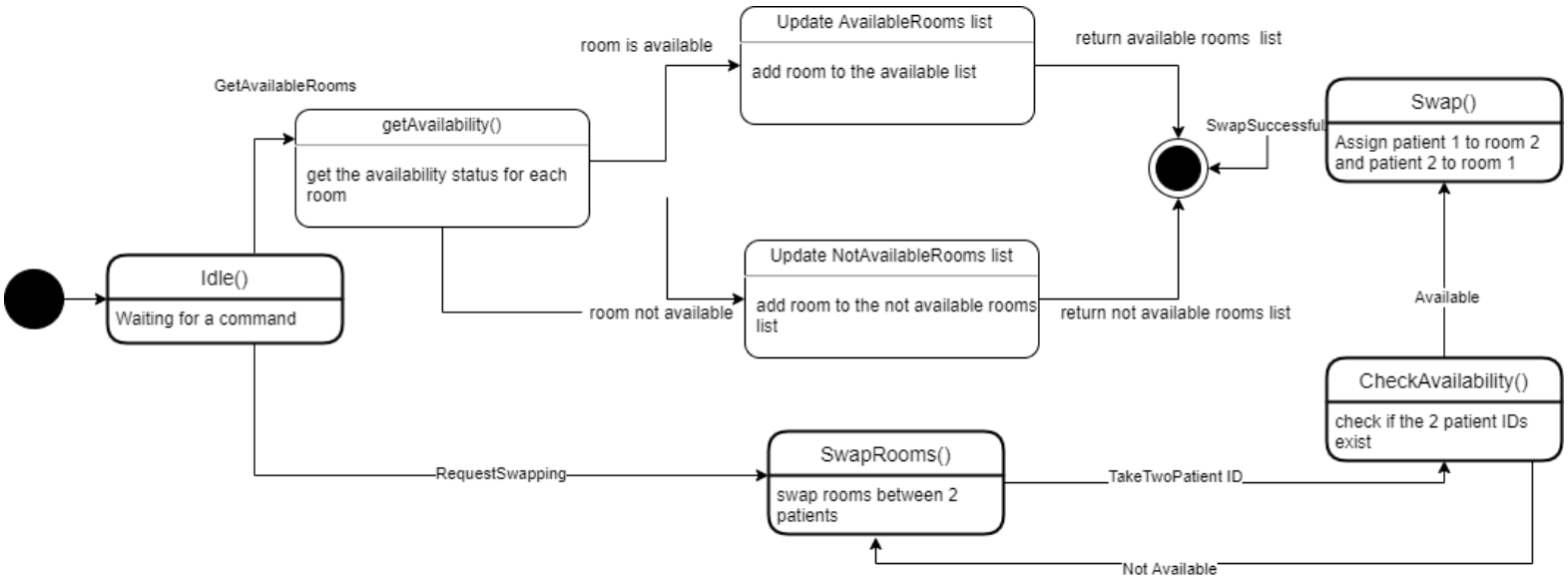
```
setScheduleCovidDep()
    // shiftType is day, swing, or night shift
    switch (shift type)
      case day
        call the DepartmentWorkerNeeded for Covid
        check if it is till need employee for day shift
        if(true)
          set employee schedule to day shift (i.e., 7:00 am - 7:00 pm)
      case swing
        call the DepartmentWorkerNeeded for Covid
        check if it is till need employee for swing shift
        if(true)
          set employee schedule to swing shift (i.e., 11:00 am - 11:00 pm)
      case swing
        call the DepartmentWorkerNeeded for Covid
        check if it is till need employee for night shift
        if(true)
          set employee schedule to night shift (i.e., 7:00 pm - 7:00 am)

    return true if the schedule was set
    return false if the schedule wasn't set
```

# V. State Machines:

## 1. Hospital Services Management:



## 2. Hospital Resource Management:

# VI. Coordinator's Report:

**Coordinator Name: Rania Dakroub**

**Meetings:**

| Date | Members Present | Duration Time |
|---|---|---|
| Saturday, April 10, 2021 | Gheeda Choucair<br>Rania Dakroub<br>Reine El Ferekh<br>Hassan Mshawrab | 4 hours |
| Monday, April 12, 2021 | Gheeda Choucair<br>Rania Dakroub<br>Reine El Ferekh<br>Hassan Mshawrab | 3 hours |
| Tuesday, April 13, 2021 | Gheeda Choucair<br>Rania Dakroub<br>Reine El Ferekh<br>Hassan Mshawrab | 4 hours |
| Wednesday, April 14, 2021 | Gheeda Choucair<br>Rania Dakroub<br>Reine El Ferekh<br>Hassan Mshawrab | 4 hours |
| Thursday, April 15, 2021 | Gheeda Choucair<br>Rania Dakroub<br>Reine El Ferekh<br>Hassan Mshawrab | 5 hours |
| Monday, April 19, 2021 | Gheeda Choucair<br>Rania Dakroub<br>Reine El Ferekh<br>Hassan Mshawrab | 8 hours |

**Who did what:**

**I. Additional Remarks:**

- We did the meetings through WEBEX and ZOOM.
- We had an interactive online chat WhatsApp group in which we discussed the project on daily basis.
- When we were facing troubles, we were always helping each other.

**Signatures:**

*Gheeda Choucair*

*Rania Dakroub*

*Reine El Ferekh*

*Hassan Mshawrab*