Take-Home Assignment 2:

Class Diagram

Individual Contributions:

Ayushman Tripathy(19114018) - Various notations used in a class diagram and the class relationships.

Jitesh Jain(19114039)- Components of our class diagram- the classes, variables and functions

Shashank Aital(19114076) - UML Diagram(class relationships) and Flow of Model.

Shreyas Dodamani(19114079)- UML Diagram(structure, classes, variables and functions) and Flow of Model

Piyush Dagdia(19114025) - Flow of Model

Notations Used In a Class Diagram:

Various Class-Relationships Used in a UML class diagram are :

Inheritance:

This relationship allows a new class to extend features of an existing class. The original class is called the base class(or superclass/parent class) and the new class obtained is the derived class(or subclass/child class). It can be either single or multiple inheritance, ie. a subclass may inherit from one or more base classes.

The inheritance relationship is also called an ***'is a'*** relation.

Since each derived class can be assumed to be a specialisation of the base class, as it modifies or extends properties of it, so, the inheritance relationship can be viewed as a **generalisation-specialisation** relationship.

The advantages of using inheritance relationships in programming are **code reuse** and **simplicity** of program design.

Association:

An association represents a structural relationship that connects two classifiers. In an association relation, two classes take each other's help (ie. invoke each other's methods) to serve user requests. It is a ***'using'*** relationship.

In an association relation between two classes, the corresponding objects are associated and **know each others' ids** (identities or references). In this way functions of corresponding classes are invoked.

This relationship can be either **bidirectional** or **unidirectional**. Even a class may only have association with itself (called **recursive** or **unary** association).

When two classes are associated, the relationship between their objects is called a **link**. Thus, an association relationship represents a group of similar

links between pairs of objects belonging to two classes. (A link can be considered as an instance of an association relation).

Aggregation:

It is a type of association where the classes concerned have a **whole-part relationship** between them. The aggregate object doesn't only know the ids of the parts , but it also has **responsibility for creating and destroying its parts** ie. The aggregate can invoke methods of its objects and also in turn can add more objects or delete them as and when required.

Here, the parts can exist independently.

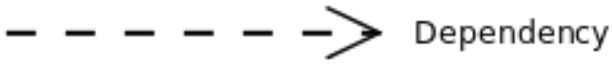
Composition:

It is a **stricter form of aggregation**, in which each of the parts is **existence-dependent** on the whole. The lifeline of the whole and part are identical, ie. the parts are created when the whole is formed and these cannot be added or deleted later on, and both get destroyed together.

To implement composition in a class model, the components in the constructor of the composite object. Thus, objects are by default created when the composite object is formed. No other methods are present in it that can do the work of addition or deletion of objects.

*Composition/aggregation relationship is also known as **'has a'** relationship.*

Dependency:



In a dependency relationship, any change made to the independent class would require the corresponding change made to the dependent class.
Dependency arises in two cases:

- A method of one class takes the object of another class as an argument.
- A class implements an interface class
- A class has object of another class as local variable

In the first and third case, changes made to the argument-object class would directly imply a change to the method-containing class. Similarly, changes to interface would directly change the class implementing it.

Other notations used:

Class:
A class is represented by a rectangular box. It contains many compartments. Class name is represented at the top. Then attributes are present in a compartment, these represent the current state of the object. Then the last compartment consists of the member functions/methods that help us set the state and utilise the member variables.

Class Modifiers:
These represent the accessibility of the attributes and functions in a given class. There are 4 types:

- Public: +
- Private: -
- Protected: #
- Package: ~

Multiplicity:
On each side of a relationship like association,composition,aggregation the multiplicity is denoted as a number or a range that denotes how many objects of the class participate in the relationship.
A range is represented as *x...y* and if we are to represent multiple objects, we use an asterisk(*).

Our class diagram contains the following elements:

Classes

- Person (Abstract Class)
    - Patient **(Sub-Class)**
    - Employee **(Sub-Class, Abstract)**
        - Staff **(Sub-Class)**
            - Doctors **(Sub-Class)**
            - Nurses **(Sub-Class)**
            - Cleaners **(Sub-Class)**
        - Director **(Sub-Class)**
        - Receptionist **(Sub-Class)**

- Hospital
    - Department **(Sub-Class, Abstract)**
        - Pharmacy **(Sub-Class)**
            - Medicine **(Sub-Class)**
        - Laboratory **(Sub-Class)**

Member Variables

- Person: name **[public]**, age **[public]**, sex **[public]**
    - Patient **(Sub-Class):** patient_id **[public]**, illness **[private]**
    - Employee **(Sub-Class):** employee_id **[public]**, salary **[private]**
        - Staff **(Sub-Class):** department **[public]**
            - Doctors **(Sub-Class):** medicinePracticingId **[private]**
            - Nurses **(Sub-Class):** medicinePracticingId **[private]**
        - Director **(Sub-Class):** headOfDepartment **[private]**
        - Receptionist **(Sub-Class):** managesDepartment **[public]**

- Hospital: name **[public]**, address **[public]**, contact_number **[public]**
    - Department **(Sub-Class):** strength **[public]**
        - Pharmacy **(Sub-Class):** count_of_medicines **[private]**, open_time **[public]**, close_time **[public]**
            - Medicine **(Sub-Class):** name **[public]**, recommended_dose **[public]**, price **[public]**, amount_in_stock **[public]**
        - Laboratory **(Sub-Class):** chemicals **[private]**, equipment **[private]**

Member Functions

- Person: talk() **[private]**, move() **[private]**, breathe() **[private]**
    - Patient **(Sub-Class):** registerPersonalInfo() **[public]**, describesSymptoms() **[public]**, giveContactInfo() **[public]**, payFees() **[public]**
    - Employee **(Sub-Class):** doJob() **[public]**
        - Staff **(Sub-Class):** reportHours() **[public]**
            - Doctors **(Sub-Class):** diagnosePatient() **[public]**, accessPatientInfo() **[public]**
            - Nurses **(Sub-Class):** administerMedicine() **[public]**
            - Cleaners **(Sub-Class):** cleanHospital() **[private]**
        - Director **(Sub-Class):** removeAndAddStaff() **[public]**, modifyFees() **[public]**, modifySalaries() **[public]**, accessPatientInfo() **[public]**
        - Receptionist **(Sub-Class):** registerPatientDetails() **[public]**, addFeesToPortal() **[public]**, directPatientsToDoctors() **[public]**, accessPatientPortal() **[public]**

- Hospital:
    - Department **(Sub-Class):** open() **[private]**, close() **[private]**
        - Pharmacy **(Sub-Class):** provideMedicines() **[public]**, orderMedicines() **[private]**
            - Medicine **(Sub-Class):** administer() **[public]**
        - Laboratory **(Sub-Class):** bloodTest() **[private]**, sugarTest() **[private]**, xRay() **[private]**

Flow of Events :

1. First, the patient enters the hospital:

Events related to the patient:

- registerPersonalInfo(): through this event, the patient initially provides information about him/herself for the hospital's records.
- describeSymptoms(): after the patient is taken into the examination room, he will describe his symptoms to the doctor/nurse for treatment.
- payFees(): After the user has availed the hospital's services, he will have to pay the hospital fees

2. Next, the receptionist interacts with the patient to get basic information:

Events related to the receptionist:

- registerPatentInfo(): When a patient initially enters the hospital, they must first tell their personal information. The receptionist is responsible for uploading it to the hospital's database.
- addFeesToPortal(): Along with the patient's information, the receptionist must also maintain track of what the patient must be charged with when they leave the hospital.

- accessPatientPortal(): In order for the receptionist to modify the patient details, the person must be able to access the portal.
- directPatientsToDoctors(): After taking patient information, the receptionist tells them which examination room to enter, and matches them up with a doctor that does not have any other engagements at that time.

3. In the examination room, the doctor examines the patient:

Events related to the doctors:

- accessPatientInfo(): before meeting the patient, the doctor will see the preliminary information about the patient to get a better perspective on the case (knowing the medical history of the patient can often provide crucial hints to diagnosing the patient).
- diagnosePatient(): the doctor will listen to the symptoms of the patient and conclude what the illness is. He will then instruct the nurses on what procedures to follow and what medications to administer in order to treat the patient.

4. The doctor makes his diagnosis and instructs the nurse on treatment of patient:

Events related to the nurse:

- administerMedicine(): The job of the nurse is to give the medicines to the patients as instructed as by the doctor.

5. Meanwhile, the cleaning staff must keep doing their work:

Events related to the cleaning staff:

- cleanHospital(): The job of cleaning the bathrooms, hallways, and taking the garbage out of the patient rooms and operating theatres, to maintain the sanitation of the hospital.

6. The directors parallelly manage the big picture events of the hospital:
Events related to the Director:

- removeAndAddStaff(): The director must make sure that qualified nurses and doctors are hired to work in the hospital
- modifySalaries(): The director must ensure that the hospital staff is being paid the right amount so that they have enough money left over for equipment, and that the staff are also happy.
- modifyFees(): In order to keep the hospital in profit, the director must make sure they are charging the correct amount for their services.
- accessPatientInfo(): In case an incident (or some major event) occurs in the hospital, the director must also have access to the patient files to know what had happened in the hospital.

Events related to the Department:

- open(): Starting the operations in each department
- close(): Ending the operations in each department

Events related to the Pharmacy:

- provideMedicines(): Provide medicines to the patients as advised by the doctors
- orderMedicines(): Order medicines when scarcity sets in

Events related to the Laboratory:

- bloodTest(): Carry out blood test of the patients
- sugarTest(): Carry out sugar measurements in the patients' blood
- xRay(): Carry out xRay of the probable fractured part of the body

Events related to the Medicine:

- administer(): Give patients medicines as per prescription