

## Aim:

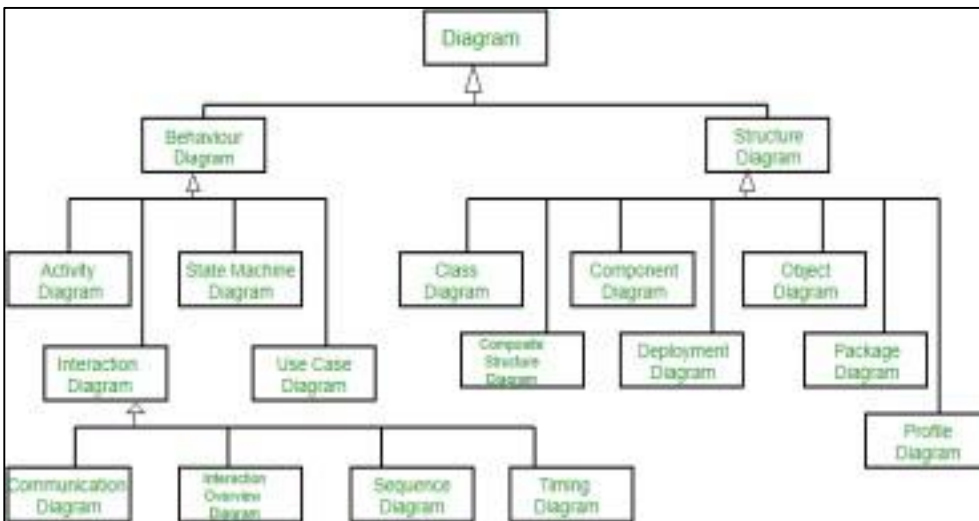
Identify scenarios & develop UML Use case and Class Diagram for the project.

## Theory:

### Brief theory about UML

- UML, short for Unified Modeling Language, is a standardized modeling language consisting of an integrated set of diagrams, developed to help system and software developers for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems.
- The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.
- The UML is a very important part of developing object-oriented software and the software development process.
- UML is not a programming language; it is rather a visual language. We use UML diagrams to portray the behavior and structure of a system.
- UML is linked with object-oriented design and analysis. UML makes the use of elements and forms associations between them to form diagrams. Diagrams in UML can be broadly classified as:
  - i. Structural Diagrams – Capture static aspects or structure of a system. Structural Diagrams include: Component Diagrams, Object Diagrams, Class Diagrams and Deployment Diagrams.
  - ii. Behavior Diagrams – Capture dynamic aspects or behavior of the system. Behavior diagrams include: Use Case Diagrams, State Diagrams, Activity Diagrams.

## Diagrams and Interaction Diagrams.



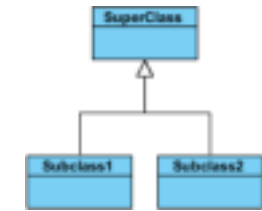

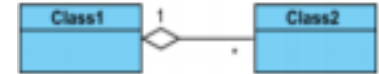
## Use Case Diagram:



- Use Case Diagram captures the system's functionality and requirements by using actors and use cases.
- Use Cases model the services, tasks, functions that a system needs to perform.
- Use cases represent high-level functionalities and how a user will handle the system.
- Use case diagrams capture the dynamic behaviour of a live system.
- It models how an external entity interacts with the system to make it work.
- It's a great starting point for any project discussion because you can easily identify the main actors involved and the main processes/functionalities of the system
- Use case diagrams describe what a system does from the standpoint of an external observer. The emphasis is on what a system does rather than how.
- Use case diagrams are closely connected to scenarios. A scenario is an example of what happens when someone interacts with the system where a scenario is a sequence of actions a system performs that yields a valuable result for a particular actor.
- Use Cases describe scenarios that describe the interaction between users of the system (the actor) and the system itself.

## Class Diagram

- Class diagram describes the attributes and operations of a class and also the constraints imposed on the system.
- The class diagrams are widely used in the modeling of object-oriented systems because they are the only UML diagrams, which can be mapped directly with object oriented languages.
- The purpose of class diagrams is to model the static view of an application.
- Class diagrams are the only diagrams which can be directly mapped with object oriented languages and thus widely used at the time of construction.

- A UML class diagram is made up of:
  - i. A set of classes and
  - ii. A set of relationships between classes
- Class diagram consists of Classes: Entities with common features, i.e., attributes and operations.
- It Represented as solid outline rectangle with compartments where these Compartments can be for name, attributes, and operations.
- Attribute and operation compartments are optional depending on the purpose of a diagram.
- A class represents a set of objects having similar attributes, operations, relationships and behavior.
- Class Relationships: A class may be involved in one or more relationships with other classes. A relationship can be one of the following types: (Refer to the figure on the right for the graphical representation of relationships).
- The following table shows the class relationship types:

Class Relationships	Graphical Representation
<b>Inheritance</b> (or Generalization): <ul style="list-style-type: none"> <li>• Represents an "is-a" relationship.</li> <li>• SubClass1 and SubClass2 are specializations of Super Class.</li> <li>• A solid line with a hollow arrowhead that point from the child to the parent class</li> </ul>	 <pre> classDiagram     SuperClass &lt; -- Subclass1     SuperClass &lt; -- Subclass2           </pre>
<b>Simple Association:</b> <ul style="list-style-type: none"> <li>• A structural link between two peer classes.</li> <li>• There is an association between Class1 and Class2 • A solid line connecting two classes</li> </ul>	 <pre> classDiagram     Class1 --- Class2           </pre>
<b>Aggregation:</b> A special type of association. It represents a "part of" relationship. <ul style="list-style-type: none"> <li>• Class2 is part of Class1.</li> <li>• Many instances (denoted by the *) of Class2 can be associated with Class1.</li> <li>• Objects of Class1 and Class2 have separate lifetimes. • A solid line with an unfilled diamond at the association end connected to the class of composite</li> </ul>	 <pre> classDiagram     Class1 o-- "*" Class2           </pre>

<p><b>Composition:</b>  A special type of aggregation where parts are destroyed when the whole is destroyed.</p> <ul style="list-style-type: none"> <li>• Objects of Class2 live and die with Class1.</li> <li>• Class2 cannot stand by itself.</li> <li>• A solid line with a filled diamond at the association connected to the class of composite</li> </ul>	 <pre> classDiagram     "1" *-- "*" Class2     class Class1     class Class2 </pre>
<p><b>Dependency:</b></p> <ul style="list-style-type: none"> <li>• Exists between two classes if the changes to the definition of one may cause changes to the other (but not the other way around).</li> <li>• Class1 depends on Class2</li> <li>• A dashed line with an open arrow</li> </ul>	 <pre> classDiagram     Class1 ..&gt; Class2     class Class1     class Class2 </pre>

A data dictionary is used to define a standard definition of data elements so that stakeholders can use and apply the data elements consistently and these can be shared by a number of solutions. A data dictionary can be created that defines data elements, including details such as names, aliases descriptions and allowable values, and including whether multiple values are permissible.

## Problem Statement

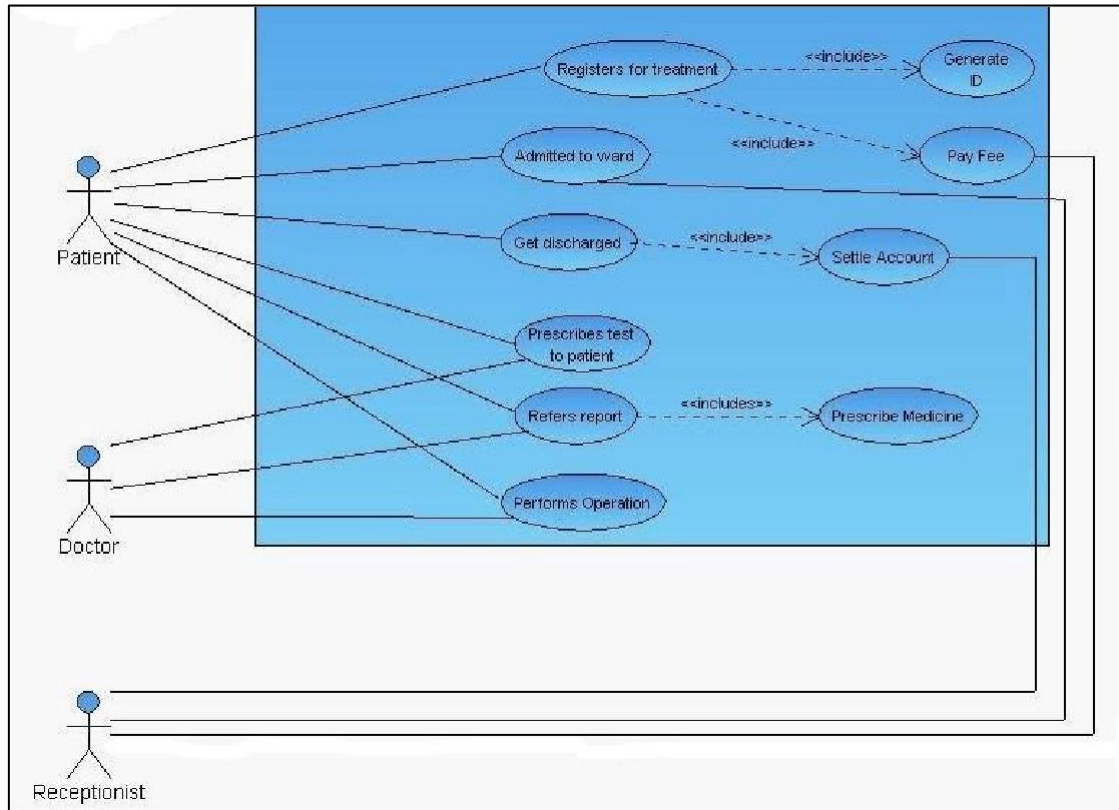
The Hospital Management System aims at providing a smooth experience of patients, staff and hospital authorities. The interactions between the hospital and the patient can be simplified for the convenience of both sides. The main function of the system is to register and store patient details and doctor details and various other details and retrieve these details as and when required, and also to manipulate these details meaningfully. Since security remains the main criteria so there will be a facility to give a unique id for every patient. All medical records have to be protected and only accessible for the allowed users. Users can search availability of a doctor and the details of a patient using the id. The database can be entered using a username and password. It is accessible either by an administrator or receptionist. Only they can add data and make changes in the database. The data can be retrieved easily. The interface is very user-friendly.

## Use Case Scenarios:

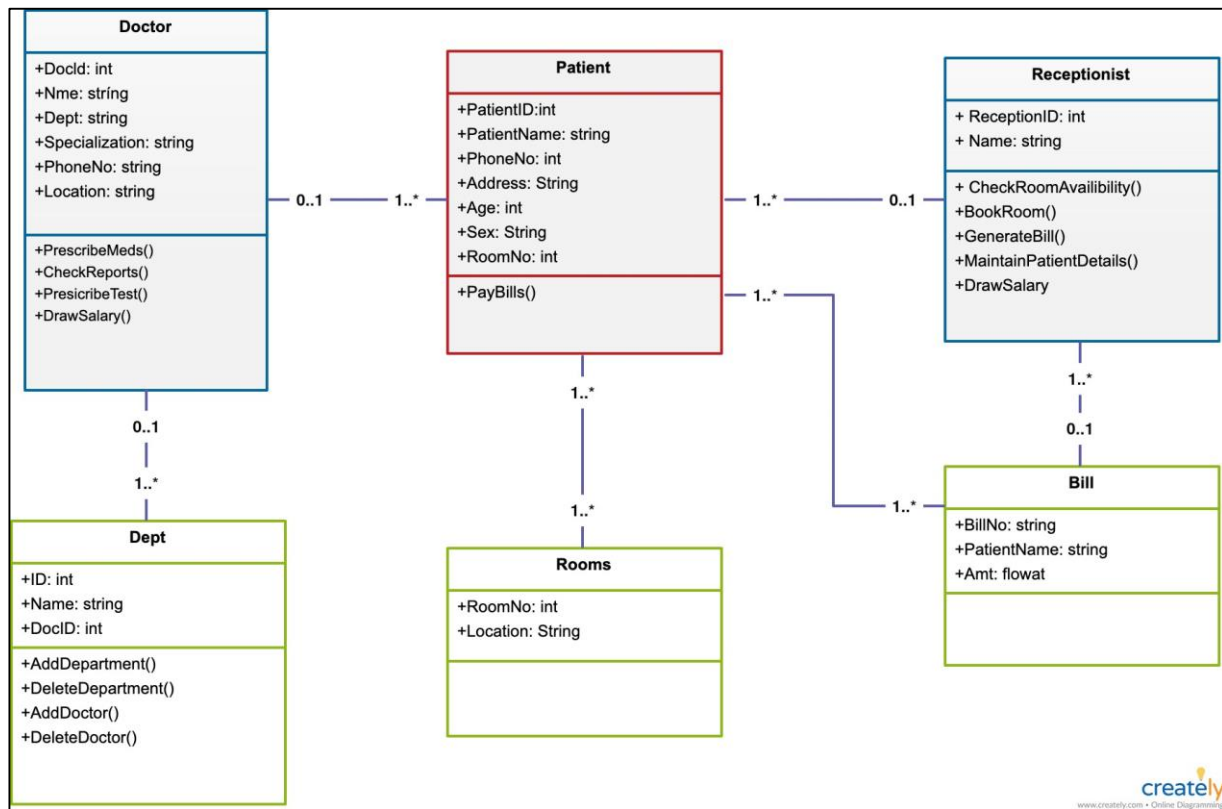
- Check by entering the correct URL in the browser, and the application should be loading properly
- Check is there any user verification functionality present on the application.
- Check by entering valid credential like username and password user should be able to login
- Check by entering invalid credentials the user should not be login into the application, and an error message should be displayed
- Check if the hospital management system application has an option to add a new patient
- Check whether all the mandatory fields are present registration portal

- Check after adding a new patient, and after completion of the payment process, the patient cards should be printed
- Check whether the patient card has the details like assign doctor name comma department, present application number comma date of join and also located bed details, etc
- Check after completion of patient check up process the details should be updated in the patient database
- Check if the patient exists in the database, and he performs some checkup then the user should be able to search the details of the present in the database
- Check if the doctors are also able to update the passenger details after check
- Check the number of roles in the hospital management system like the patient, doctor, admin, accountant, etc
- Check that the authorized users can see the doctor details in the portal like the doctors' timings and fees.
- Check if there is any functionality to add a new doctor in the hospital management system, for instance, we have added patient details in the database
- Check whether the admin users can delete doctor and patient information by the hospital management system portal
- Check whether an accountant user type can calculate the bills for patients by collecting data from different systems.
- Check after the formation of a bill that should be an option to print the bill or to generate a hard copy of the bill.
- Check the authorized users have the privilege to check the details report of the patients like day wise
- Check the admin has all the access

## Use Case Diagram:



## Class diagram:



## Conclusion:

In this experiment we learnt to Identify scenarios and to develop UML Use cases and Class Diagrams. UML Use case diagrams represent the “requirements” of the system whereas Class diagrams show the building blocks (classes) that are needed to build the software/system. In a Hospital Management System, the actors are the Doctor, Patient and Receptionist at max. Each patient’s records are automatically stored on the system. We then created the Use case Diagram and Class diagram for Hospital Management System. Thus, we have completed the experiment successfully.