

Practical - 1

Aim - Identifying the requirements from the problem statement.

1. Requirements for Student Profile Management Website:

For the 'Student Profile Management Website' requirements are as follows:

- Website shows Student's Personal information which includes (Name, both Parent name, student's Mobile number, Address of residence, parent's mobile number, field of study, result of mid & end semester exams, fees information, attendance (subject - wise) and weekly timetable).
- Dashboard for subject teachers where they can update student's attendance, add marks according to the entry already provided, and view the weekly timetable.
- Website's full control is with the administrator who is the super user and can create entries for new students enrolled in the university, can edit/view student's all information, create/edit weekly timetables, check/update fees information.

2. Methods of gathering requirement for Student Profile Management Website:

- For the 'Student Profile Management Website' requirement gathered from resources on internet and brainstorming,
- evaluating other documents to get the insight and functionalities.

3. Functional & non-functional requirements for Student Profile Management Website:

Functional Requirements:

Role of Admin:

- Ability to create/edit student profiles, including personal information.
- Exclusive access to creating/editing lecture timetables.
- Full control to edit any information on the website.
- Check/Update fees information of students.

- Managing the Database.

Teacher:

- View lecture timetable.
- Add and edit marks for students.
- Mark and edit attendance records for students.

Student:

- Access to personal information but cannot edit it.
- Access to view personal lecture timetables.
- View their marks for various subjects.
- Check their attendance records.
- Access information about fees per semester.

Non-Functional Requirements:

1. Security:
 - a. User authentication and authorization for different roles (Admin, Teacher, Student) by assigning email addresses and password.
 - b. Data encryption for sensitive information like passwords and personal data by installing SSL Certificate.
2. Performance:
 - a. The system should respond within a reasonable time for various actions.
 - b. Handling multiple users (Admin, Teacher, Student) without significant performance degradation.
 - c. low latency for reflecting information on the front end.
3. Usability:
 - a. User-friendly interface with easy navigation for all types of users.
 - b. Clear and intuitive design for entering marks, attendance, and other data.
 - c. safe environment for making fees payments.
4. Reliability:
 - a. The system should be available and reliable for use during regular operational hours.
 - b. Data integrity and consistency are crucial; data should not be lost or corrupted.
 - c. Backup the data will be available and refresh whenever new information is added

4. processes covered under functional & non-functional requirements for Student Profile Management Website:

Functional Processes:

For Admin:

- Student profile (creation and editing).
- Lecture timetable creation and editing.
- Fees check and edit.

For Teacher:

- Viewing lecture timetable.
- Adding and editing student marks.
- Marking and editing attendance.

Student:

- Viewing a personal timetable.
- Checking marks, attendance, and fee information.

Non-Functional Processes:

Security:

- User authentication and access control.
- Data encryption for sensitive data.

Performance:

- Task response time monitoring and optimization.
- Load balancing to handle several users at same time.

Usability:

- Interface design and user experience optimization.
- Clear and concise user interface.

Reliability:

- Regular system backups to ensure data recovery.
- Monitoring system uptime and addressing downtime promptly.

Practical - 2

Aim - Introduction to UML Diagrams

1. UML.

Unified Modeling Language (UML) is a general purpose modeling language. The main aim of UML is to define a standard way to visualize the way a system has been designed. It is quite similar to blueprints used in other fields of engineering. 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 helps software engineers, businessmen and system architects with modeling, design and analysis.

2. Introduction of UML diagrams with history.

The main aim of UML is to define a standard way to visualize the way a system has been designed. It is quite similar to blueprints used in other fields of engineering. 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 helps software engineers, businessmen and system architects with modeling, design and analysis.

The 1990s was the period of development of object-oriented languages such as C++. These object-oriented languages (OOL) were used to build complex but compelling systems.

It was developed by software engineers Grady Booch, Ivar Jacobson, and James Rumbaugh of Rational software during 1994 and 1995. It was under development till 1996.

All UML creators, viz, Grady Booch, Ivar Jacobson, and James Rumbaugh had a fabulous idea for creating a language that will reduce the complexity.

- Booch's method was flexible to work with throughout the design and creation of objects.
- Jacobson's method contributed a great way to work on use-cases. It further has a great approach for high-level design.
- Rumbaugh's method turned out to be useful while handling sensitive systems.

- Behavioral models and state-charts were added in the UML by David Harel.

UML was acknowledged as a norm by the Object Management Group (OMG) in 1997. Object Management Group is responsible for maintaining UML regularly since it was adopted as a standard.

In 2005, the International Organization for Standardization established UML as an ISO standard. It is used in many industries for designing object-oriented models.

UML's latest version is 2.5.1 which was released in December 2017.

3. Categories of UML diagrams (Static & Dynamic) & nature (Structural & Behavioural).

There are two main categories: structure diagrams and behavioral diagrams.

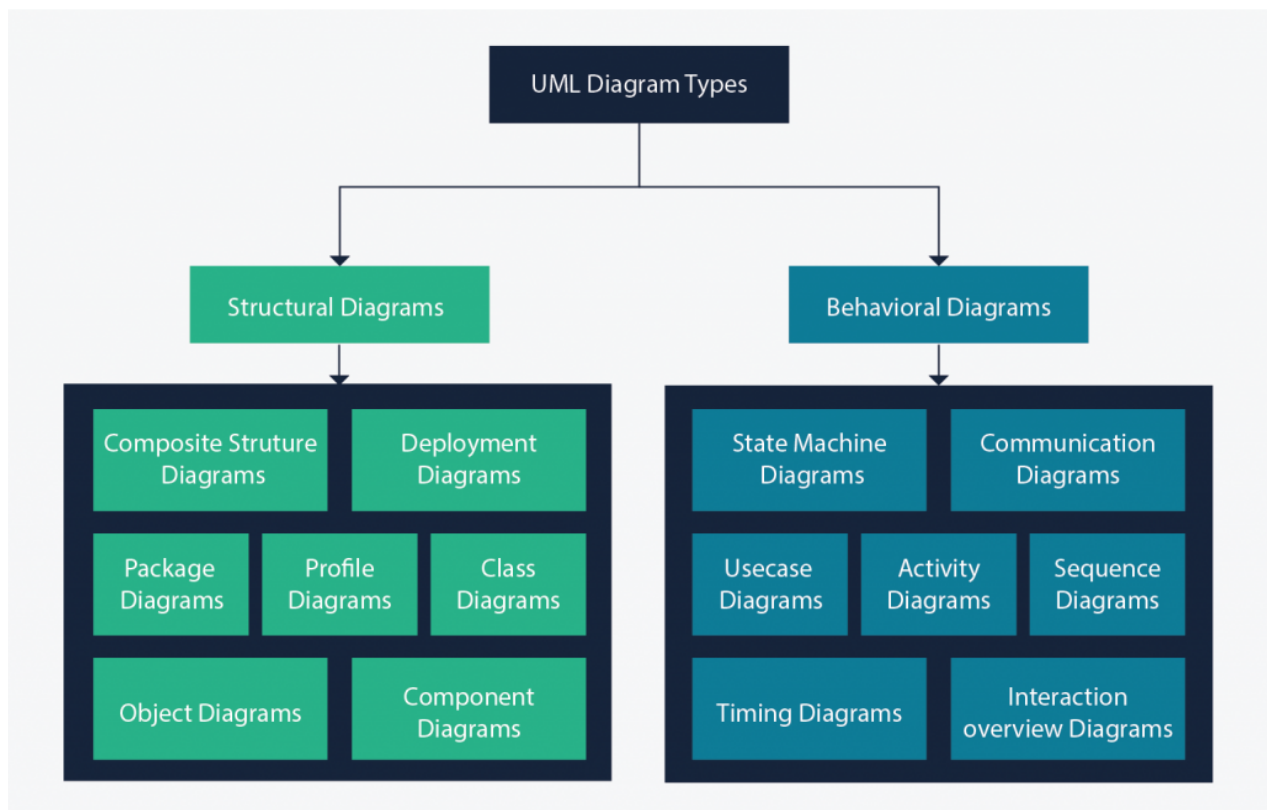


Figure: 01

Static View:

- Class Diagram: Illustrates the structure of classes, attributes, methods, and their relationships in the system.
- Object Diagram: Shows instances of classes and their relationships at a specific point in time.
- Component Diagram: Depicts physical components and their dependencies, aiding in system architecture design.
- Package Diagram: Represents how elements are organized into packages and namespaces.

Dynamic View:

- Use Case Diagram: Displays user interactions and system functionality from an external perspective.
- Sequence Diagram: Depicts interactions and message flow between objects over time.
- Communication Diagram: Highlights interactions between objects with focus on messages exchanged.
- State Machine Diagram: Models behavior transitions of objects or components through different states.
- Activity Diagram: Represents workflows, processes, and activities within the system, showing control flow.

Structural Diagrams:

- Class Diagram: Depicts classes, their attributes, methods, and relationships in the system's static structure.
- Object Diagram: Illustrates specific instances of classes and their relationships at a given point in time.
- Component Diagram: Displays the physical components and dependencies in the system architecture.
- Package Diagram: Represents how system elements are grouped into packages and namespaces.
- Deployment Diagram: Shows the distribution of software components onto hardware nodes.

Behavioral Diagrams:

- Use Case Diagram: Depicts user interactions and system functionality, helping to define use cases.
- Sequence Diagram: Illustrates the interactions and message sequences between objects over time.
- Communication Diagram: Emphasizes object interactions through messages exchanged.
- State Machine Diagram: Models the behavior of a single object or component as it transitions through different states.
- Activity Diagram: Represents workflows, processes, and activities within the system, demonstrating control flow.

Practical - 3

Aim - Modelling UML Use case diagrams and capturing Use case scenarios.

1. Introduction of Use case diagrams with symbols (Actor, Use case, relationship, etc):

In the Unified Modeling Language (UML), a use case diagram can summarize the details of your system's users (also known as actors) and their interactions with the system. To build one, you'll use a set of specialized symbols and connectors. An effective use case diagram can help your team discuss and represent:

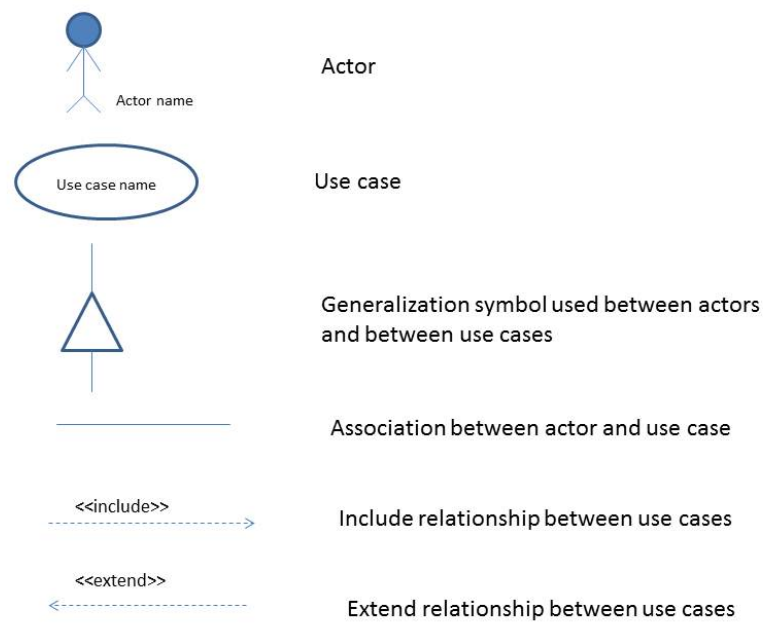
- Scenarios in which your system or application interacts with people, organizations, or external systems,
- Goals that your system or application helps those entities (known as actors) to achieve,
- The scope of your system.

Common components include:

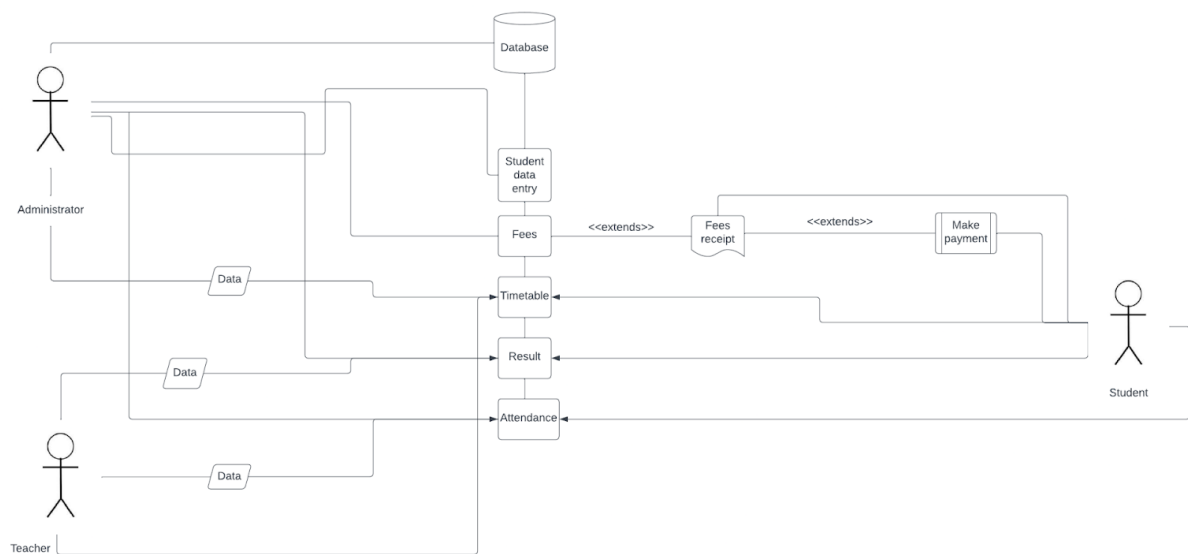
- Actors: The users that interact with a system. An actor can be a person, an organization, or an outside system that interacts with your application or system. They must be external objects that produce or consume data.
- System: A specific sequence of actions and interactions between actors and the system. A system may also be referred to as a scenario.
- Goals: The end result of most use cases. A successful diagram should describe the activities and variants used to reach the goal.

Use case diagram symbols and notation:

- Use cases: Horizontally shaped ovals that represent the different uses that a user might have.
- Actors: Stick figures that represent the people actually employing the use cases.
- Associations: A line between actors and use cases. In complex diagrams, it is important to know which actors are associated with which use cases.
- System boundary boxes: A box that sets a system scope to use cases. All use cases outside the box would be considered outside the scope of that system. For example, Psycho Killer is outside the scope of occupations in the chainsaw example found below.
- Packages: A UML shape that allows you to put different elements into groups. Just as with component diagrams, these groupings are represented as file folders.

**Figure: 01**

2. Use case diagram for Student Profile Management Website:

**Figure: 02**

Practical - 4

Aim - Identifying domain classes from problem statement & modeling UML Class Diagrams

1. Give a brief introduction about Class diagrams with symbols:

A class diagram is a fundamental concept in the Unified Modeling Language (UML), a widely-used notation for visualizing and documenting software systems. Class diagrams are used to represent the static structure of a system, focusing on the classes, their attributes, methods, and the relationships between them. They provide a visual representation of the object-oriented design of a system.

Here are some key symbols and elements commonly found in class diagrams:

Class:

- Symbol: A rectangle divided into three compartments.
- Purpose: Represents a class in the system. The top compartment contains the class name, the middle compartment lists the class attributes, and the bottom compartment lists the class methods.

Attributes:

- Symbol: Typically shown as name: type, e.g., "name: String".
- Purpose: Describes the properties or characteristics of a class. These are variables or fields that store data.

Methods:

- Symbol: Typically shown as name(parameters): return type, e.g., "calculateDiscount(price: double): double".
- Purpose: Represents the operations or behaviors that a class can perform. Methods define the functionality of the class.

Associations:

- Symbol: A line connecting two classes with optional multiplicity notations (e.g., 1, *, 0..1) at each end.
- Purpose: Represents relationships between classes. Multiplicity indicates the number of instances of one class related to another.

Inheritance/Generalization:

- Symbol: A solid line with a closed, hollow triangle at the superclass end.
- Purpose: Indicates that one class (the subclass) inherits the attributes and methods of another class (the superclass). It represents an "is-a" relationship.

Realization/Interface:

- Symbol: A dashed line with a hollow triangle at the interface end.
- Purpose: Indicates that a class implements the methods declared in an interface. It represents a contract or "provides-a" relationship.

Dependency:

- Symbol: A dashed arrow line pointing from the dependent class to the independent class.
- Purpose: Represents a weaker relationship between classes, often indicating that one class depends on another, such as through method parameters or local variables.

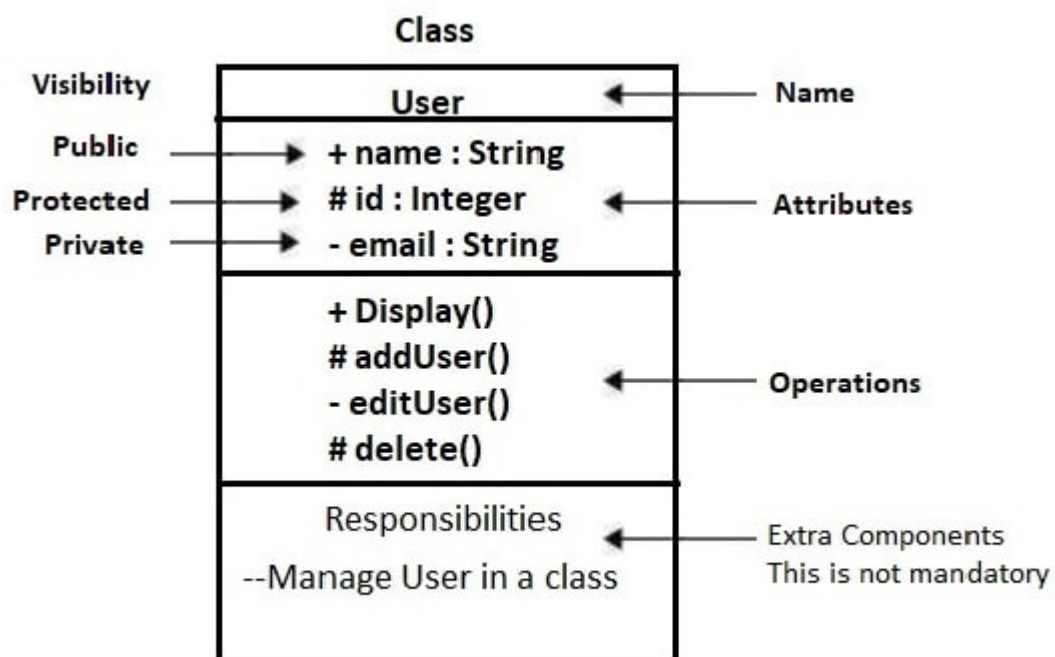
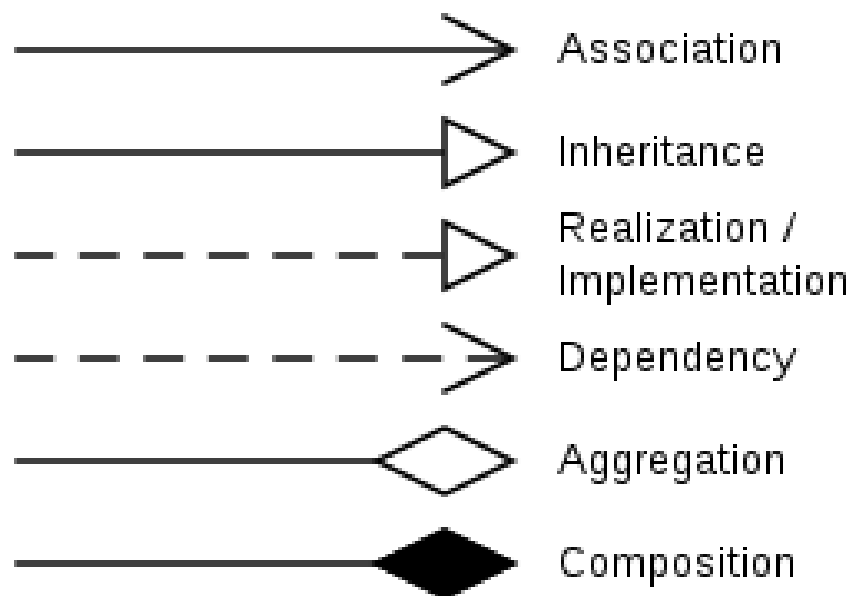
Aggregation:

- Symbol: A diamond shape line connecting the whole (diamond end) to the part (class end).
- Purpose: Indicates a "has-a" relationship between classes, where one class contains or is composed of instances of another class. Aggregation implies a weaker association than composition.

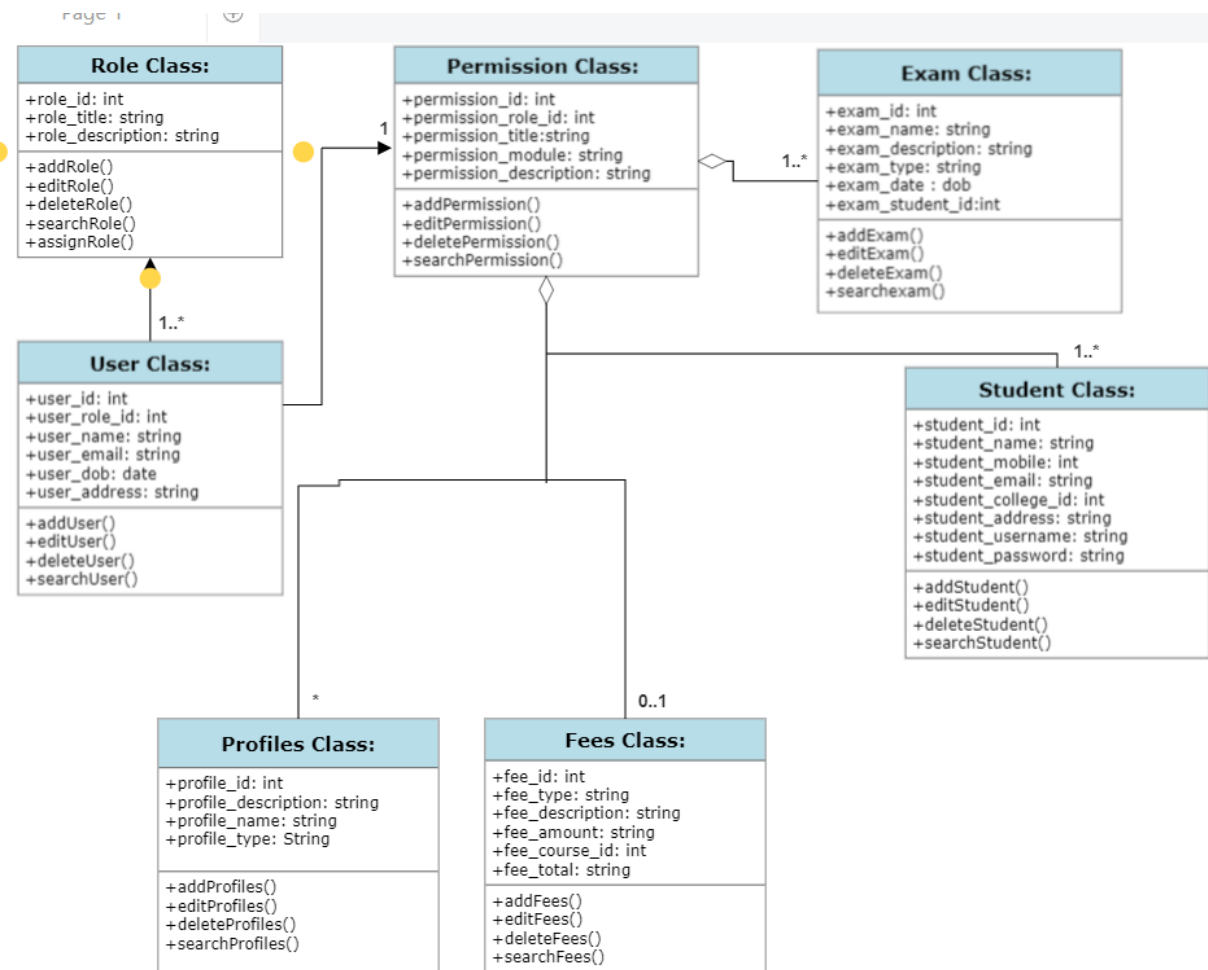
Composition:

- Symbol: A solid diamond shape line connecting the whole (diamond end) to the part (class end).
- Purpose: Represents a strong "whole-part" relationship, where one class is composed of, and directly owns, instances of another class. The lifetime of the part is dependent on the whole.

Class diagrams are valuable tools for software architects and developers to visualize and design the structure of their systems, aiding in communication and documentation throughout the software development process. They help clarify the relationships and responsibilities of classes, making it easier to plan, implement, and maintain software systems.



2. Class diagram for Student Profile Management:



Practical - 5

Aim – Activity Modeling

1. Introduction about activity diagram for Student Profile Management Website.

An activity diagram is a visual representation of a process or workflow, often used in software engineering and business modeling. It helps to illustrate the sequence of actions, decisions, and interactions that occur within a system or a business process. Activity diagrams use various symbols to represent different elements and aspects of the process. Here's a brief introduction to some of the key symbols used in activity diagrams:

1. Initial Node: This is represented by a solid black circle and marks the starting point of the activity diagram.
2. Activity State: Represented by a rounded rectangle, an activity state signifies a specific action or task within the process.
3. Decision Node: Represented by a diamond shape, a decision node indicates a point in the process where a decision must be made. The outgoing flows from this node represent different possible paths based on the decision outcome.
4. Fork Node: This is represented by a solid black bar that splits into multiple outgoing flows. It signifies parallel execution of activities.
5. Join Node: Similar to the fork node, a join node is represented by a solid black bar that merges multiple incoming flows into one. It indicates the synchronization of parallel activities.
6. Final Node: This is represented by a solid black circle with a border, marking the end point of the activity diagram.
7. Control Flow: Represented by arrows, control flows indicate the sequence of activities and the order in which they are executed.
8. Object Flow: This is represented by a dashed arrow and indicates the flow of data or objects between activities.
9. Swimlanes: Swimlanes are used to group related activities or actions. They are represented by columns or rows that categorize different actors or roles involved in the process.




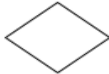
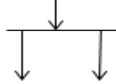


Sr. No	Name	Symbol
1.	Start Node	
2.	Action State	
3.	Control Flow	
4.	Decision Node	
5.	Fork	
6.	Join	
7.	End State	

Figure: 1

2. Activity diagram of Student Profile Management Website.

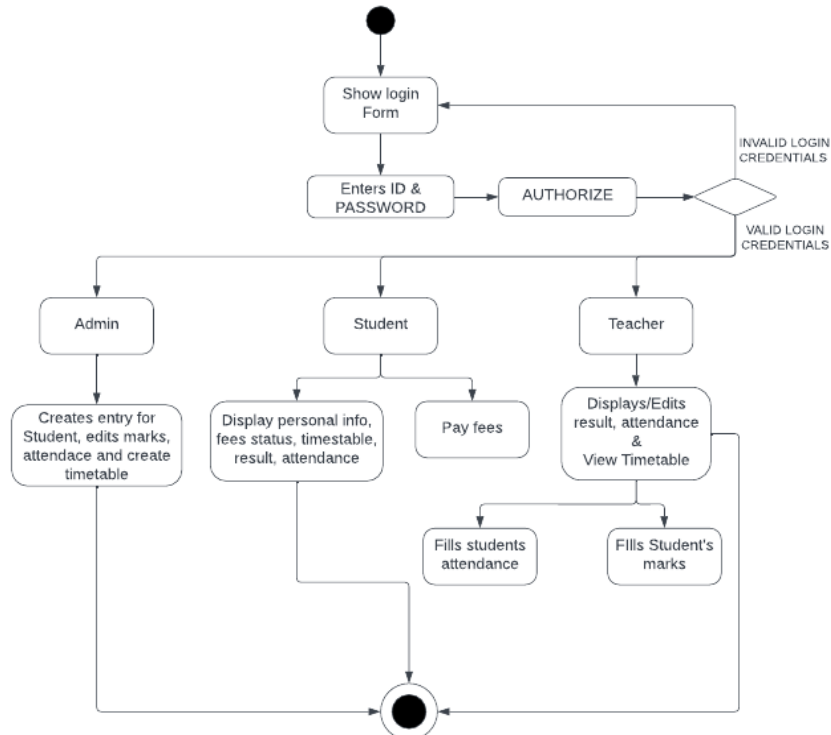


Figure: 2