Unit 8 : Introduction to Object-Oriented Development (OOD)

Introduction

Object-oriented development (OOD) is a software development approach that revolves around the concept of "objects." Objects are instances of classes, which are templates that define both the properties (attributes) and behaviors (methods or functions) that the object will have. This approach contrasts with procedural programming, where the focus is on functions or procedures that operate on data. By focusing on objects, OOD enhances modularity, reusability, and maintainability of code. Here are some key concepts in object-oriented development:

Classes and Objects:

A class is a blueprint or template that defines the attributes and methods that its objects will have. For example, a Car class could define attributes like color, model, and speed, along with methods like accelerate() and brake().
An object is an instance of a class. So, a Car object could be a specific car, like a redToyotaCorolla with certain values for color, model, and speed.

Encapsulation:

Encapsulation refers to bundling data (attributes) and methods that operate on the data within a single unit, known as a class. It hides the internal state of objects from the outside world, only exposing a controlled interface through methods. This helps prevent unauthorized access and modification of an object's data, enhancing security and reducing complexity.

Inheritance:
Inheritance allows a class to inherit properties and behaviors from another class. This promotes code reusability and the creation of hierarchical relationships between classes. For instance, a Truck class can inherit from a Vehicle class, gaining general attributes and behaviors like drive() while adding its own specific attributes such as cargoCapacity.

Polymorphism:
Polymorphism means that one method can have different behaviors based on the object it is acting upon. This can be achieved through method overloading (same method name but different parameters) or method overriding (redefining a method in a subclass). It allows for flexibility and the ability to call the same method on different objects, resulting in different outcomes.

Abstraction:

Abstraction involves hiding complex implementation details and showing only the essential features of an object. By defining abstract classes or interfaces, developers can create a clear separation between what the object does and how it achieves its functionality.
By using these principles, OOD helps break down large systems into smaller, manageable components (objects), making development and maintenance easier.

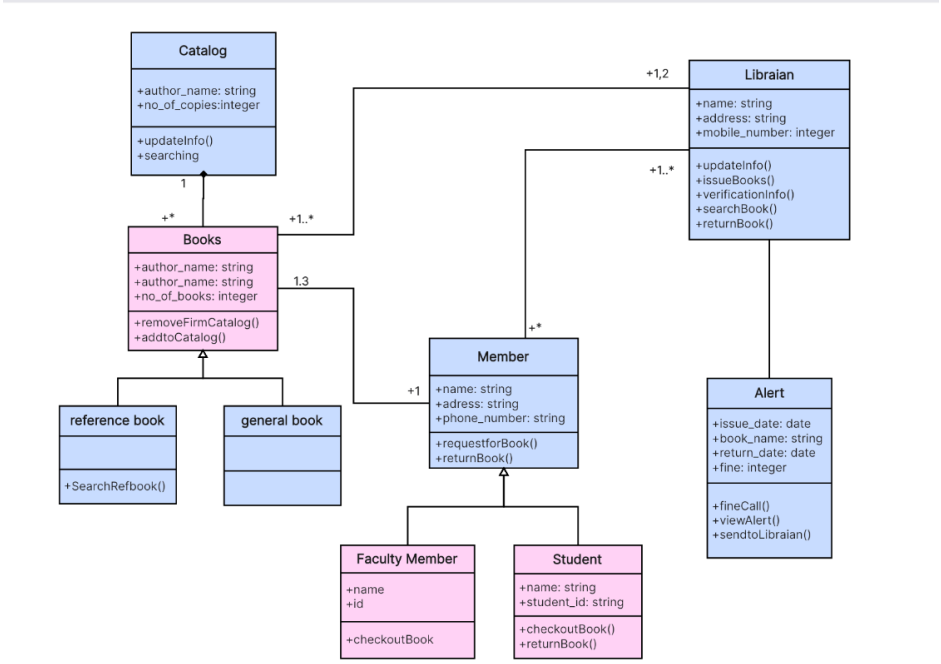
Unified Modeling Language (UML)

Unified Modeling Language (UML) is a standardized modeling language used to visualize, specify, construct, and document the artifacts of software systems. UML helps software engineers and designers represent system architecture, making it easier to understand complex systems. It is widely used in object-oriented development to illustrate the design and interactions of various components in the system.

UML includes a set of diagrams to represent different perspectives of the system, including:

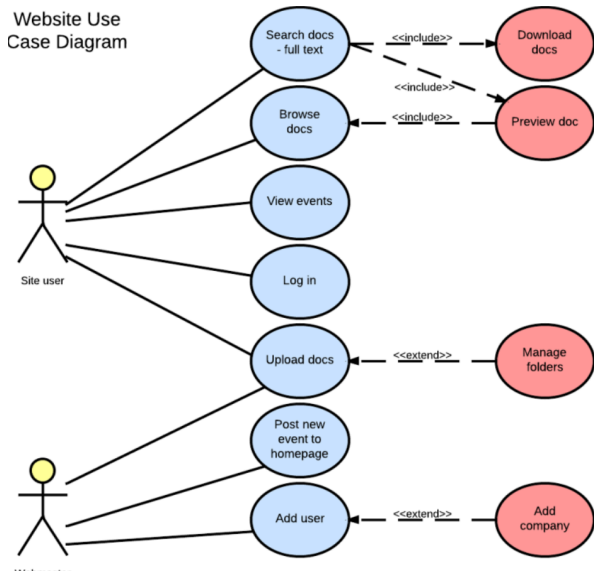
Class Diagrams:

These diagrams show the classes in the system, their attributes, methods, and relationships (such as inheritance and associations). They are used to visualize the static structure of the system.



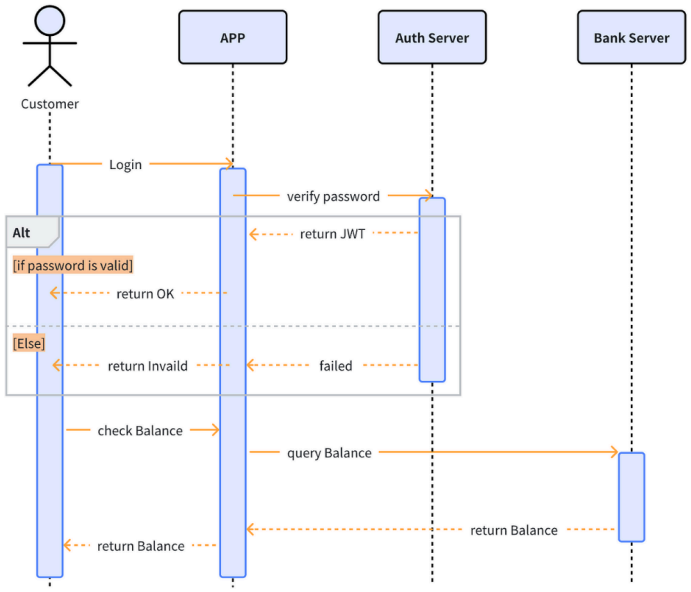
Use Case Diagrams:

Use case diagrams capture the functionality of a system from the end user's perspective. They depict the actors (users or other systems) and the use cases (services or functions) that the system provides.



Sequence Diagrams:

Sequence diagrams illustrate how objects interact over time, showing the order of messages exchanged between objects to complete a process. These diagrams are useful for modeling dynamic behavior in the system.

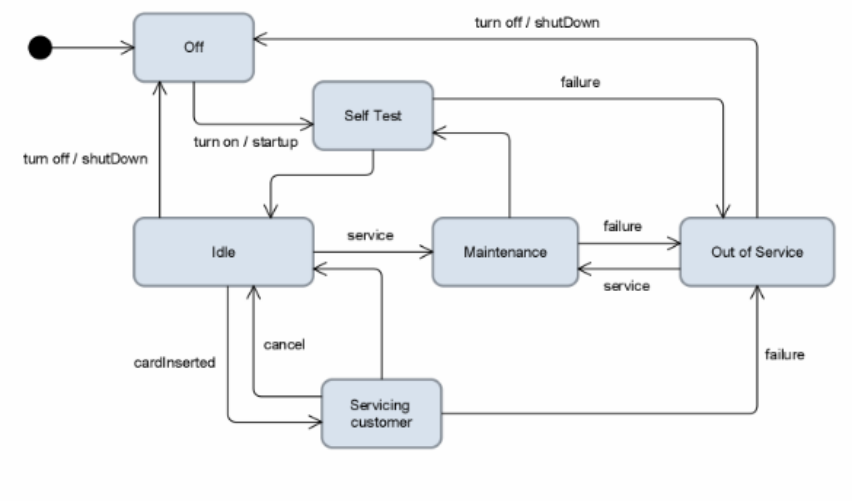


Activity Diagrams:

Activity diagrams are used to model workflows, describing the flow of control between different activities or actions in the system. They are particularly useful for understanding business processes or the flow of operations.

State Diagrams:

State diagrams depict the different states an object can be in during its lifecycle and the events that cause it to transition between states. These are especially useful in systems where objects have a complex set of states, such as in embedded systems or simulations.



Component Diagrams:

These diagrams describe the physical components of the system, such as software components and their dependencies. They help in modeling the system's modular architecture.

Deployment Diagrams:

Deployment diagrams show the physical deployment of software components on hardware nodes. They help visualize the system's runtime architecture.
By using these diagrams, UML allows developers, designers, and stakeholders to communicate more effectively, making it easier to plan and implement object-oriented designs.

Functional, Structural, and Behavioral Models

In object-oriented development, different models are used to represent various aspects of the system. These models help to break down the system from different viewpoints, providing a comprehensive understanding of its functionality, structure, and behavior.

Functional Models:

Functional models describe what the system does and define the operations that the system performs. These models are focused on capturing the system's behavior from a high level, often using diagrams like use case diagrams and activity diagrams. They help to identify the key functions the system will provide and how they interact with users or other systems. For example, a functional model for an e-commerce system may describe how users browse products, add items to the cart, and complete a purchase.

A functional model refers to a conceptual or technical representation of the behavior of a system or component. It focuses on how the system performs tasks, processes, or operations based on inputs and outputs. Functional models are used to describe the logic, workflow, and operations within the software. They can be seen as blueprints that guide developers in understanding how different components of the system interact and function together.

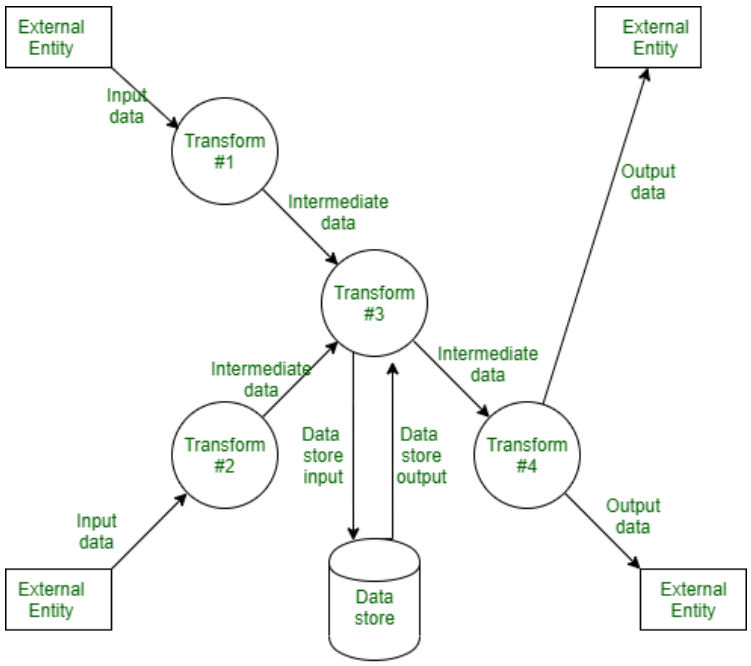
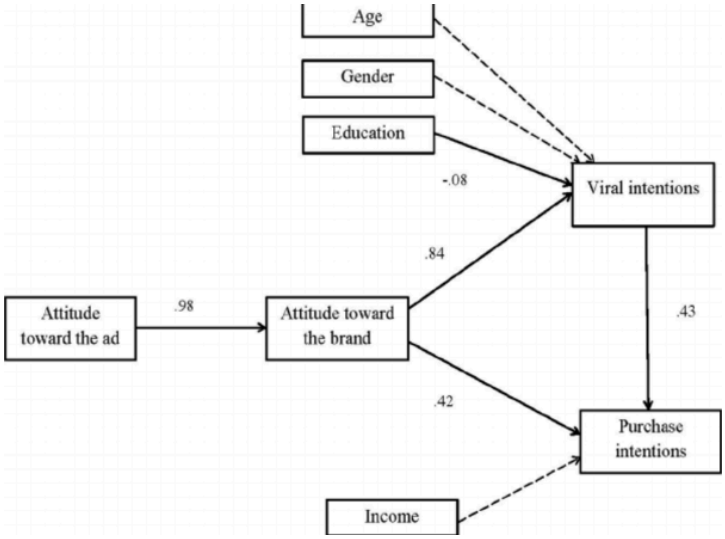


Figure - Information-flow Model

Structural Models:

Structural models depict the organization of the system, showing how its components are structured and how they relate to each other. These models focus on the system's static architecture and include class diagrams and component diagrams. Structural models are used to define the system's data and objects and their relationships. For instance, a structural model for a banking system may show how accounts, transactions, and customers are represented as classes and how these entities interact.



Behavioral Models:

Behavioral models focus on how the system behaves in response to different inputs or events. These models depict the dynamic aspects of the system, such as how objects interact, change state, and collaborate to perform tasks. Sequence diagrams and state diagrams are commonly used to represent the system's behavior. For example, a behavioral model for a ticket reservation system might describe how a reservation request is processed, how the system handles different user actions (like seat selection and payment), and how the state of the reservation changes over time.
Together, these three models provide a holistic view of the system:

Aspect	Functional Models	Structural Models	Behavioral Models
Purpose	Describe what the system does and its functional requirements.	Define the organization and relationships of system components.	Show interactions and changes in system state over time.
Focus	User needs and system operations.	System architecture and data organization.	System behavior, workflows, and responses to events.
Key Components	Use cases, user stories, and functional requirements.	Class diagrams, data structures, and object relationships.	Sequence diagrams, state diagrams, and activity diagrams.
Examples	Use case diagrams, functional flows.	Class diagrams, component diagrams.	Sequence diagrams, state machine diagrams.
Representation	Models functions as actions and processes.	Models entities as objects and their relationships.	Models how entities interact over time and respond to events.
Role in System Design	Defines what the system must achieve from the user's perspective.	Structures system elements for efficient data management and organization.	Demonstrates system dynamics and how it reacts to inputs.

Object-oriented development offers a robust framework for designing complex systems. By using UML, developers can document and communicate their designs effectively, while functional, structural, and behavioral models provide comprehensive views of system operations. This combination of methodologies makes object-oriented development a powerful approach for creating modular, maintainable, and scalable software systems.

USE CASE DIAGRAM OF MELODY MART

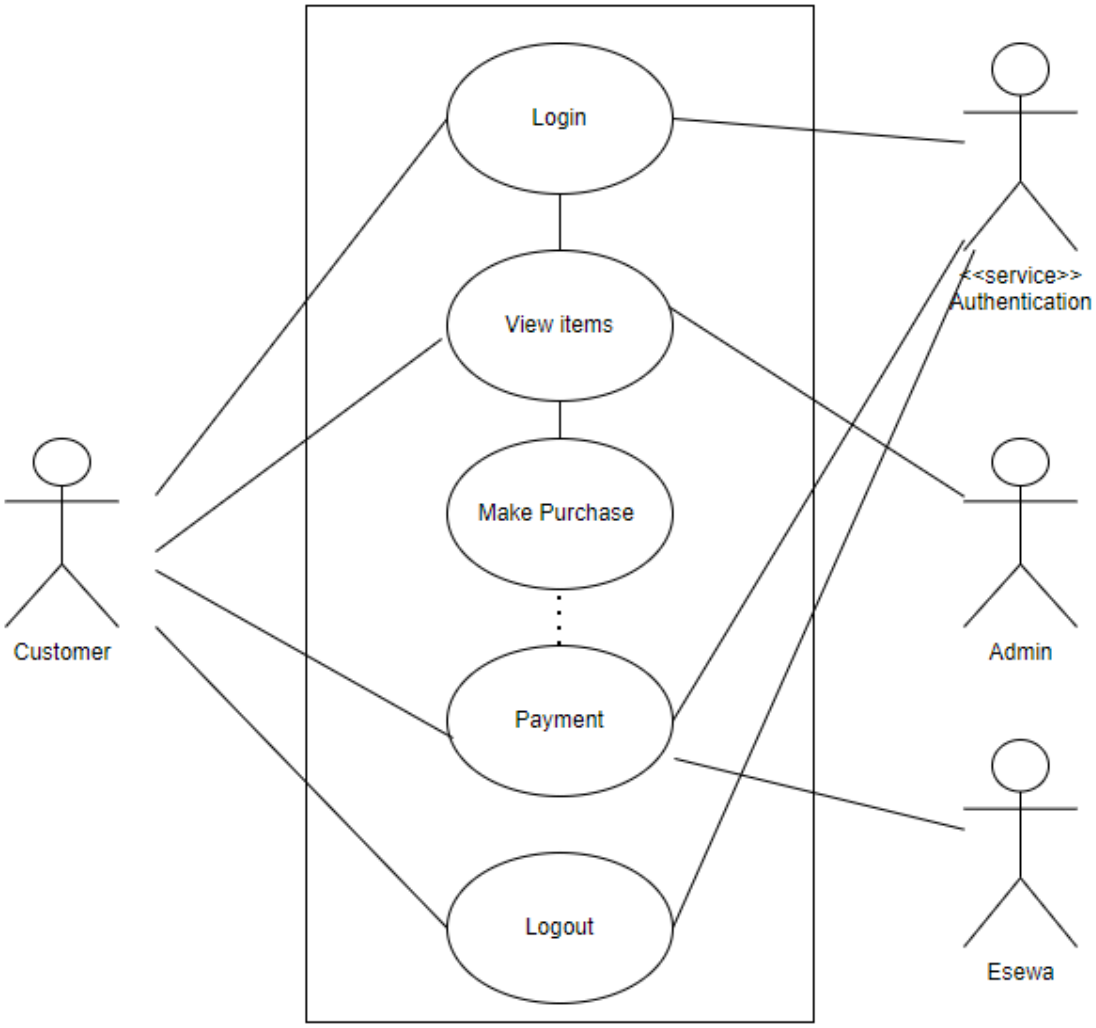


Fig:User case diagram of instrumental insight management system