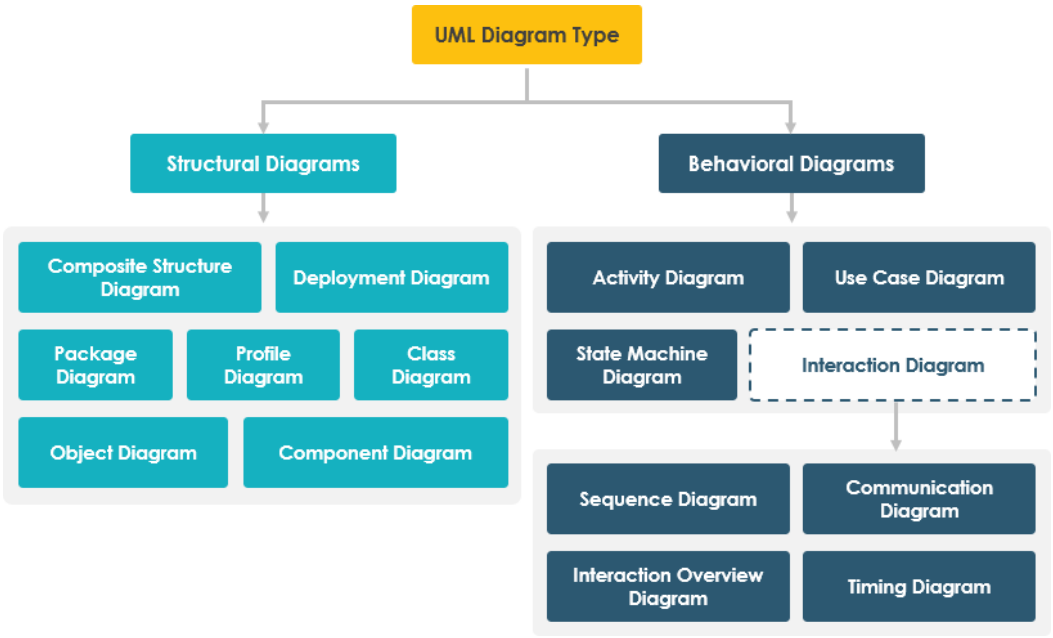
OOAD – Unit – I

Different types of UML diagrams.

- UML is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems.
- UML was created by the Object Management Group (OMG) and UML 1.0 specification draft was proposed to the OMG in January 1997.
- OMG is continuously making efforts to create a truly industry standard.
- UML stands for Unified Modeling Language.
- UML is different from the other common programming languages such as C++, Java, COBOL, etc.
- UML is a pictorial language used to make software blueprints.
- UML can be described as a general purpose visual modeling language to visualize, specify, construct, and document software system.



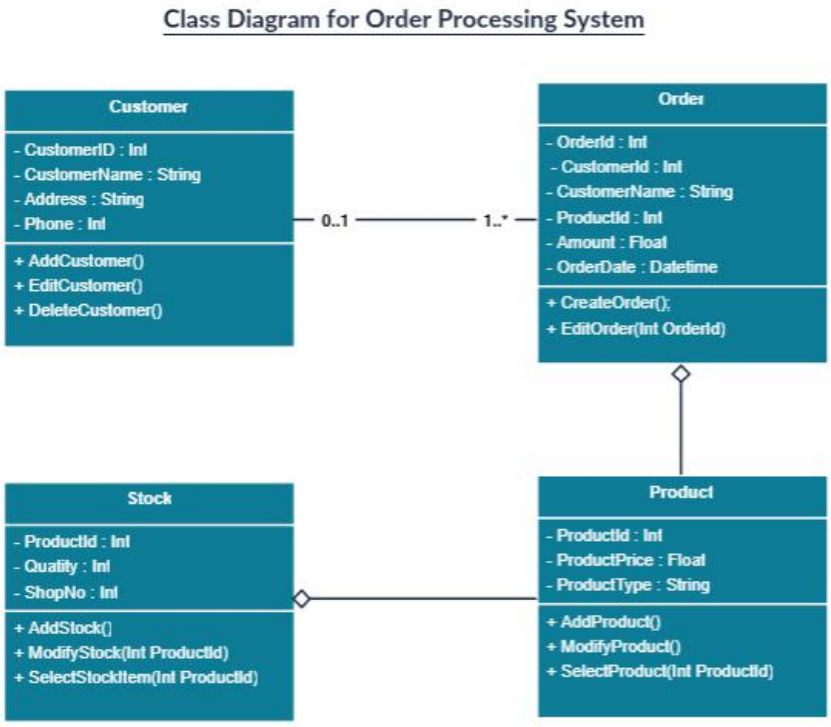
Structural Diagrams

The structural diagrams represent the static aspect of the system. These static aspects represent those parts of a diagram, which forms the main structure and are therefore stable.

- Class diagram
- Object diagram
- Component diagram
- Deployment diagram

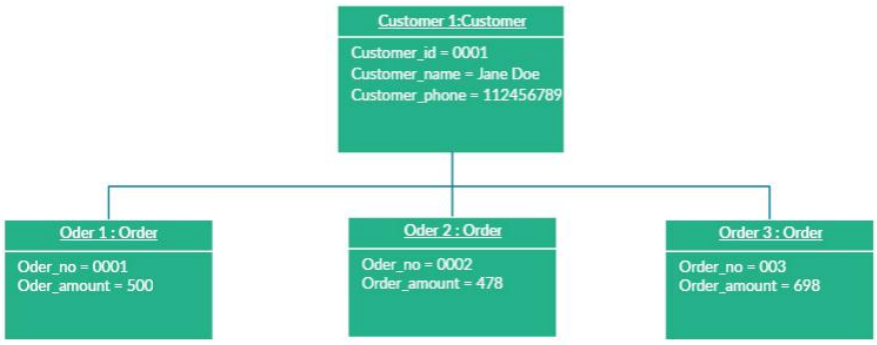
Class Diagram

Class diagrams are the most common diagrams used in UML. Class diagram consists of classes, interfaces, associations, and collaboration. Class diagrams basically represent the object-oriented view of a system, which is static in nature.



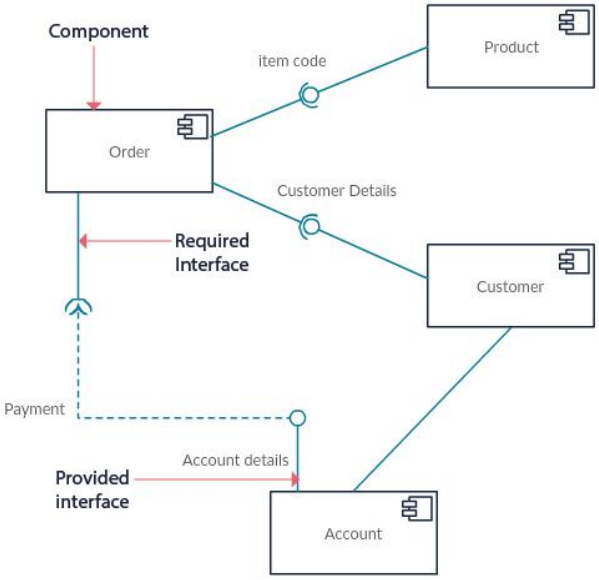
Object Diagram

Object diagrams can be described as an instance of class diagram. Thus, these diagrams are more close to real-life scenarios where we implement a system.



Component Diagram

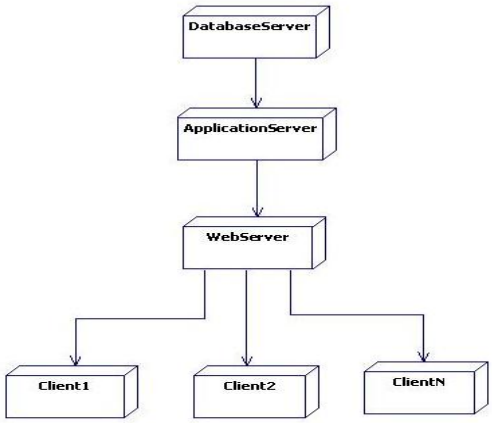
Component diagrams represent a set of components and their relationships. These components consist of classes, interfaces, or collaborations. Component diagrams represent the implementation view of a system.



Deployment Diagram

Deployment diagrams are a set of nodes and their relationships. These nodes are physical entities where the components are deployed.

Deployment diagrams are used for visualizing the deployment view of a system. This is generally used by the deployment team.



Behavioral Diagrams

Behavioral diagrams basically capture the dynamic aspect of a system. Dynamic aspect can be further described as the changing/moving parts of a system.

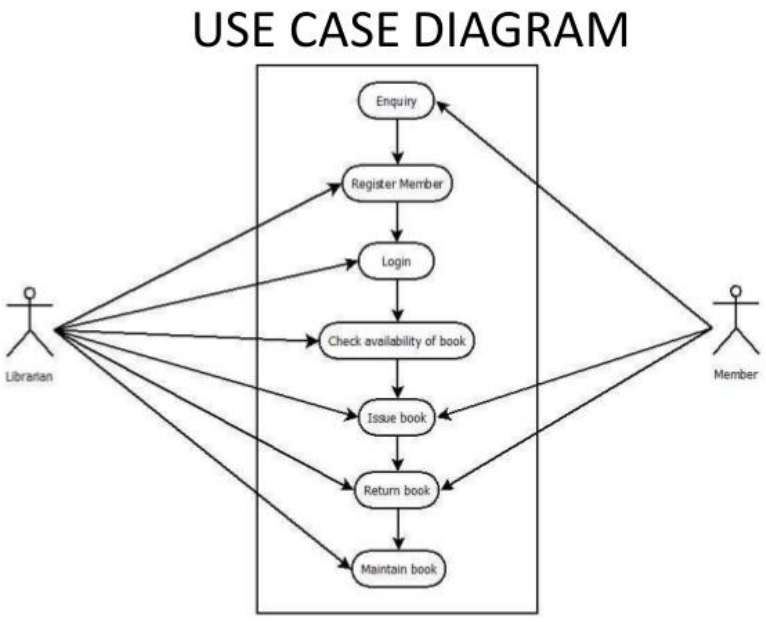
UML has the following five types of behavioral diagrams −

- Use case diagram
- Sequence diagram
- Collaboration diagram
- Statechart diagram
- Activity diagram

Use Case Diagram

Use case diagrams are a set of use cases, actors, and their relationships. They represent the use case view of a system.

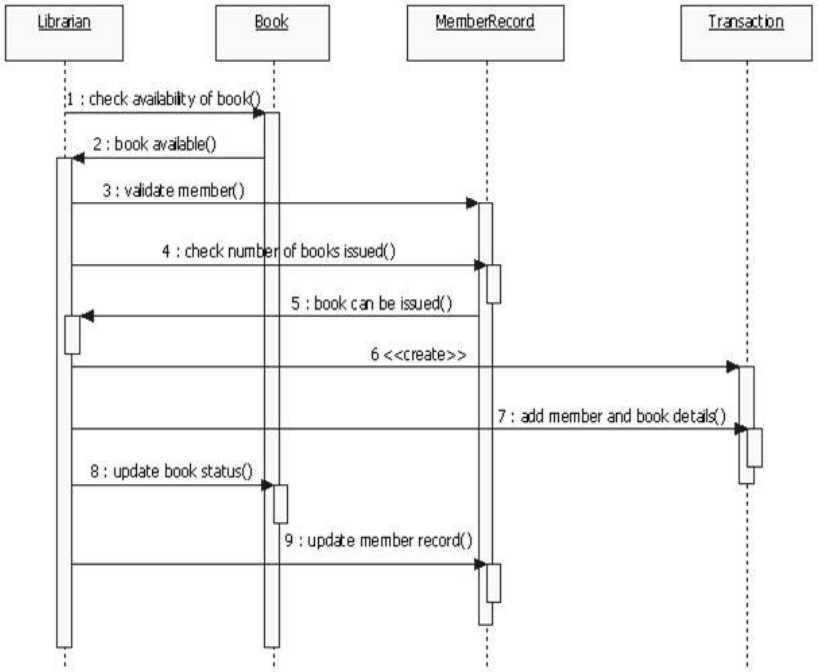
A use case represents a particular functionality of a system. Hence, use case diagram is used to describe the relationships among the functionalities and their internal/external controllers. These controllers are known as actors.



Sequence Diagram

A sequence diagram is an interaction diagram. From the name, it is clear that the diagram deals with some sequences, which are the sequence of messages flowing from one object to another.

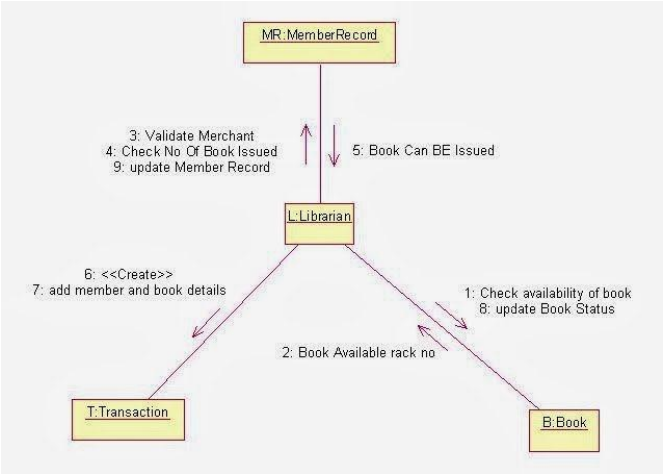
Interaction among the components of a system is very important from implementation and execution perspective. Sequence diagram is used to visualize the sequence of calls in a system to perform a specific functionality.



Collaboration Diagram

Collaboration diagram is another form of interaction diagram. It represents the structural organization of a system and the messages sent/received. Structural organization consists of objects and links.

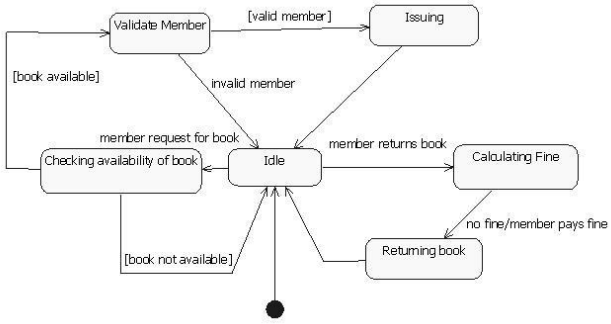
The purpose of collaboration diagram is similar to sequence diagram. However, the specific purpose of collaboration diagram is to visualize the organization of objects and their interaction.



Statechart Diagram

Any real-time system is expected to be reacted by some kind of internal/external events. These events are responsible for state change of the system.

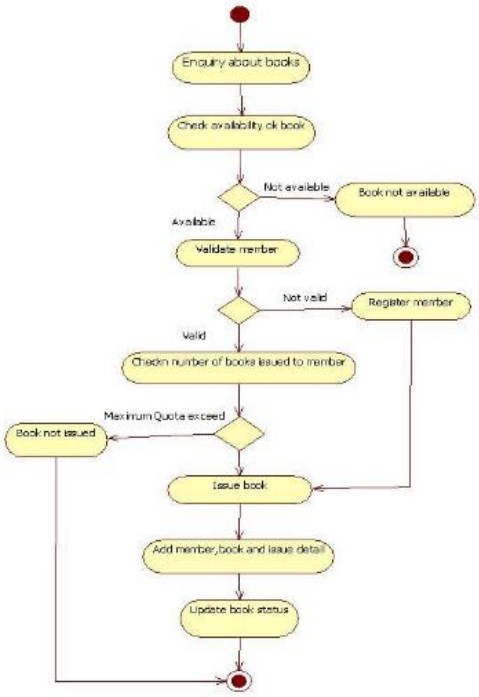
Statechart diagram is used to represent the event driven state change of a system. It basically describes the state change of a class, interface, etc.



Activity Diagram

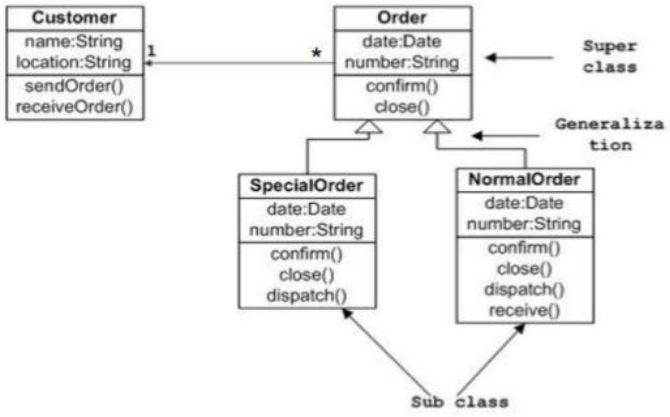
Activity diagram describes the flow of control in a system. It consists of activities and links. The flow can be sequential, concurrent, or branched.

Activities are nothing but the functions of a system. Numbers of activity diagrams are prepared to capture the entire flow in a system.

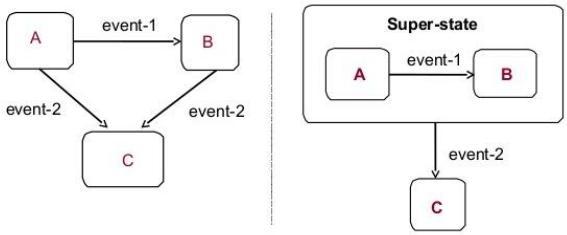


Three models

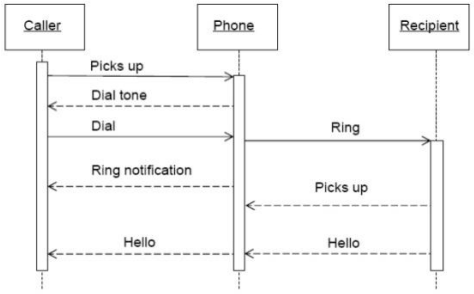
1. **Class Model:**
   - It represents the static, structural, "data" aspects of a system.
   - It describes the structure of objects in a system- their identity, their relationships to other objects, their attributes, and their operations.
   - Goal in constructing class model is to capture those concepts from the real world that are important to an application.
   - Class diagrams express the class model.



2. **State Model:**
   - It represents the temporal, behavioral, "control" aspects of a system.
   - State model describes those aspects of objects concerned with time and the sequencing of operations – events that mark changes, states that define the context for events, and the organization of events and states.
   - State diagram express the state model.
   - Each state diagram shows the state and event sequences permitted in a system for one class of objects.
   - State diagram refer to the other models.
   - Actions and events in a state diagram become operations on objects in the class model. References between state diagrams become interactions in the interaction model.



3. **Interaction model**
    - It represents the collaboration of individual objects, the "interaction" aspects of a system.
    - Interaction model describes interactions between objects – how individual objects collaborate to achieve the behavior of the system as a whole.
    - The state and interaction models describe different aspects of behavior, and you need both to describe behavior fully.
    - Use cases, sequence diagrams and activity diagrams document the interaction model.

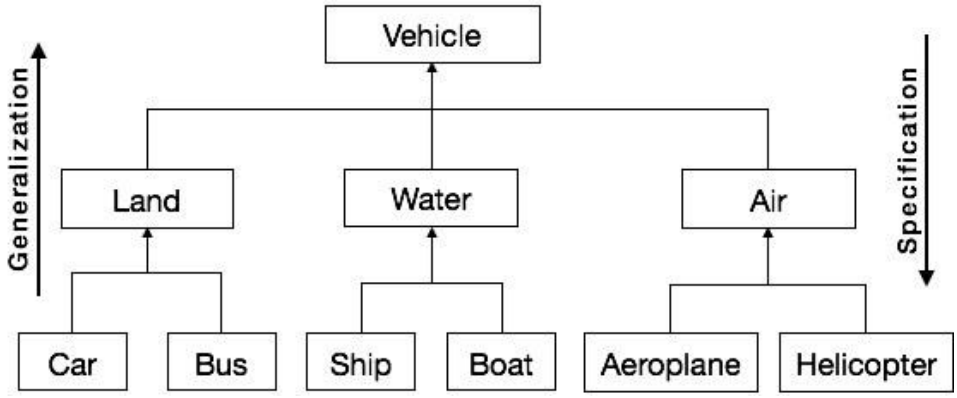


Modeling Concept

MODELING

- A model is simplification of reality
- It provides the blueprints of a system
- Models can encompass detailed plans
- A good model includes those elements that have broad abstraction
- It helps to visualize a system as it is or according to the need.
- Models permit to specify the structure or behavior of a system
- Models give a template that guides in constructing a model

Example



Models serve several purposes –

Testing a physical entity before building it: Medieval built scale models of Gothic Cathedrals to test the forces on the structures. Engineers test scale models of airplanes, cars and boats to improve their dynamics.

Communication with customers: Architects and product designers build models to show their customers (note: mock-ups are demonstration products that imitate some of the external behavior of a system).

Visualization: Storyboards of movies, TV shows and advertisements let writers see how their ideas flow.

Reduction of complexity: Models reduce complexity to understand directly by separating out a small number of important things to do with at a time.

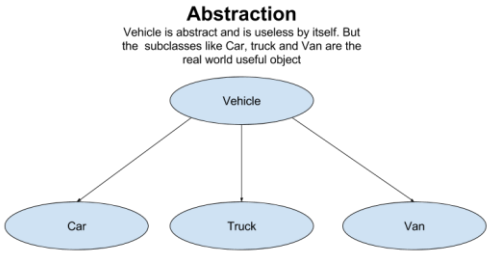
ABSTRACTION

Abstraction means to focus on the essential features of an element or object in OOP, ignoring its extraneous or accidental properties. The essential features are relative to the context in which the object is being used.

Grady Booch has defined abstraction as follows −

"An abstraction denotes the essential characteristics of an object that distinguish it from all other kinds of objects and thus provide crisply defined conceptual boundaries, relative to the perspective of the viewer."

Example − When a class Student is designed, the attributes enrolment_number, name, course, and address are included while characteristics like pulse_rate and size_of_shoe are eliminated, since they are irrelevant in the perspective of the educational institution.



Evidence for usefulness of object oriented development

WHAT IS OBJECT ORIENTATION?

OO means that we organize software as a collection of discrete objects (that incorporate both data structure and behavior).

There are four aspects (characteristics) required by an OO approaches.

- Identity
- Classification.
- Inheritance.
- Polymorphism.

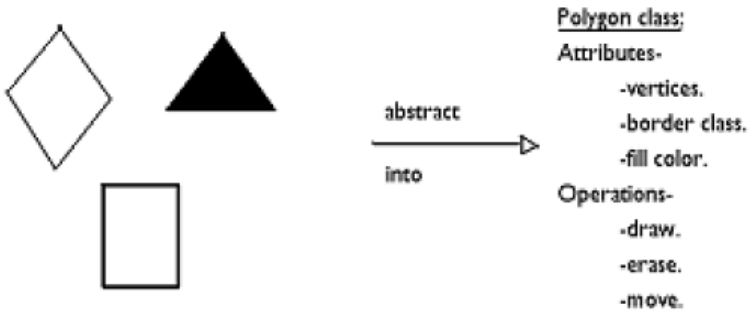
Identity:

- Identity means that data is quantized into discrete, distinguishable entities called objects.
- E.g. for objects: personal computer, bicycle, queen in chess etc
- Objects can be concrete (such as a file in a file system) or conceptual (such as scheduling policy in a multiprocessing OS). Each object has its own inherent identity. (i.e two objects are distinct even if all their attribute values are identical).

Classification:

Classification means that objects with the same data structure (attribute) and behavior (operations) are grouped into a class.

- E.g. paragraph, monitor, chess piece.
- Each object is said to be an instance of its class.
- Fig below shows objects and classes: Each class describes a possibly infinite set of individual objects.



Inheritance:

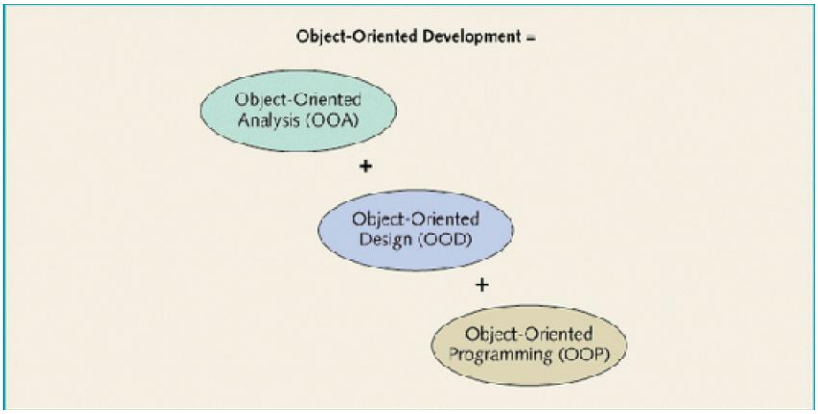
It is the sharing of attributes and operations (features) among classes based on a hierarchical relationship. A super class has general information that sub classes refine and elaborate.

E.g. Scrolling window and fixed window are sub classes of window.

Polymorphism:

Polymorphism means that the same operation may behave differently for different classes.
For E.g. move operation behaves differently for a pawn than for the queen in a chess game.

WHAT IS OO DEVELOPMENT?



Development refers to the software life cycle: Analysis, Design and Implementation. The essence of OO Development is the identification and organization of application concepts, rather than their final representation in a programming language. It's a conceptual process independent of programming languages. OO development is fundamentally a way of thinking and not a programming technique.

OO methodology

Here we present a process for OO development and a graphical notation for representing OO concepts. The process consists of building a model of an application and then adding details to it during design.

The methodology has the following stages

System conception: Software development begins with business analysis or users conceiving an application and formulating tentative requirements.

Analysis: The analyst scrutinizes and rigorously restates the requirements from the system conception by constructing models. The analysis model is a concise, precise abstraction of what the desired system must do, not how it will be done.

The analysis model has two parts

Domain Model- a description of real world objects reflected within the system.

Application Model- a description of parts of the application system itself that are visible to the user.

E.g. In case of stock broker application-

Domain objects may include- stock, bond, trade & commission.

Application objects might control the execution of trades and present the results.

System Design: The development teams devise a high-level strategy- The System Architecture- for solving the application problem. The system designer should decide what performance characteristics to optimize, chose a strategy of attacking the problem, and make tentative resource allocations.

Class Design: The class designer adds details to the analysis model in accordance with the system design strategy. His focus is the data structures and algorithms needed to implement each class.

Implementation: Implementers translate the classes and relationships developed during class design into a particular programming language, database or hardware.

Three models

The three kinds of models to describe a system from different view points.

1. Class Model—for the objects in the system & their relationships. It describes the static structure of the objects in the system and their relationships. Class model contains class diagrams- a graph whose nodes are classes and arcs are relationships among the classes.

2. State model—for the life history of objects. It describes the aspects of an object that change over time. It specifies and implements control with state diagrams-a graph whose nodes are states and whose arcs are transition between states caused by events.

3. Interaction Model—for the interaction among objects. It describes how the objects in the system co-operate to achieve broader results. This model starts with use cases that are then elaborated with sequence and activity diagrams.

Use case – focuses on functionality of a system – i.e what a system does for users.

Sequence diagrams – shows the object that interact and the time sequence of their interactions.

Activity diagrams – elaborates important processing steps.

OO THEMES

1. Abstraction

Abstraction lets you focus on essential aspects of an application while ignoring details i.e focusing on what an object is and does, before deciding how to implement it. It's the most important skill required for OO development.

2. Encapsulation (information hiding)

It separates the external aspects of an object (that are accessible to other objects) from the internal implementation details (that are hidden from other objects). Encapsulation prevents portions of a program from becoming so interdependent that a small change has massive ripple effects.

3. Combining data and behavior

Caller of an operation need not consider how many implementations exist. In OO system the data structure hierarchy matches the operation inheritance hierarchy

4. Sharing

- OO techniques provide sharing at different levels.
- Inheritance of both data structure and behavior lets sub classes share common code.
- OO development not only lets you share information within an application, but also offers the prospect of reusing designs and code on future projects.