

## Lecture - 1

### Object Oriented System Design

Systems development is the process of defining, designing, testing, and implementing a new software application or program. Programming is used as a tool for System Development.

There are mainly two approaches of system / software development, *procedural and object-oriented approach*.

#### 1. Procedure Oriented Approach

In procedure-based programming language, a main program can be divided into many functions where each function is connected to another function to accomplish the overall goal of the program. To perform any particular task, set of function are compulsory.

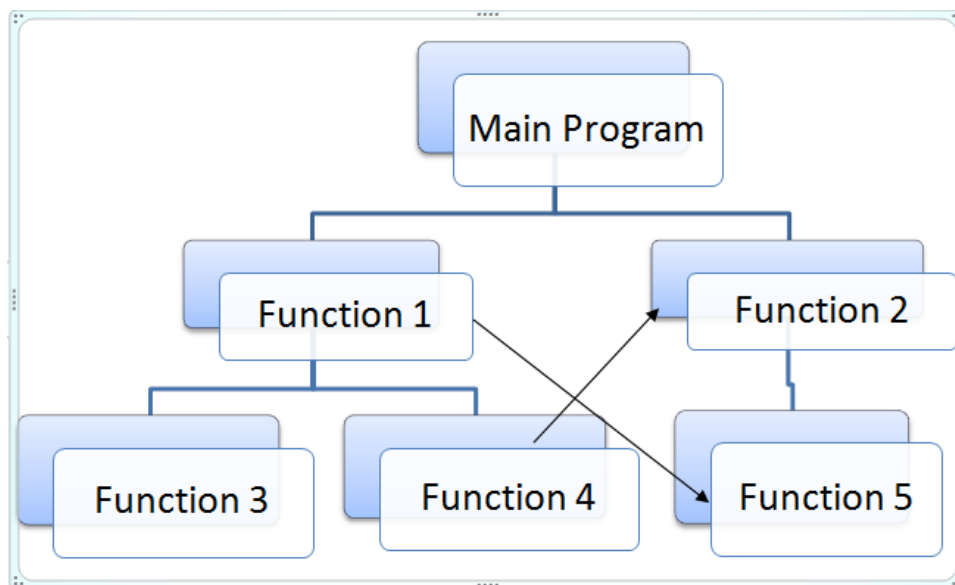


Fig. 1 Procedure-Oriented Programming approach

**For example,** a program may involve collecting data from user, performing some kind of calculation on that data and printing the data on screen when is requested. Calculating, reading or printing can be written in a program with the help of different functions on different tasks.

##### 1.1 Data Security Problem in Procedural Approach:

In the procedural approach some data is local i.e., within the scope of the function, while there is a need to keep some data as global which is shared among all the modules. So, there are chances of accidental modification of global data that may violate the security of data.

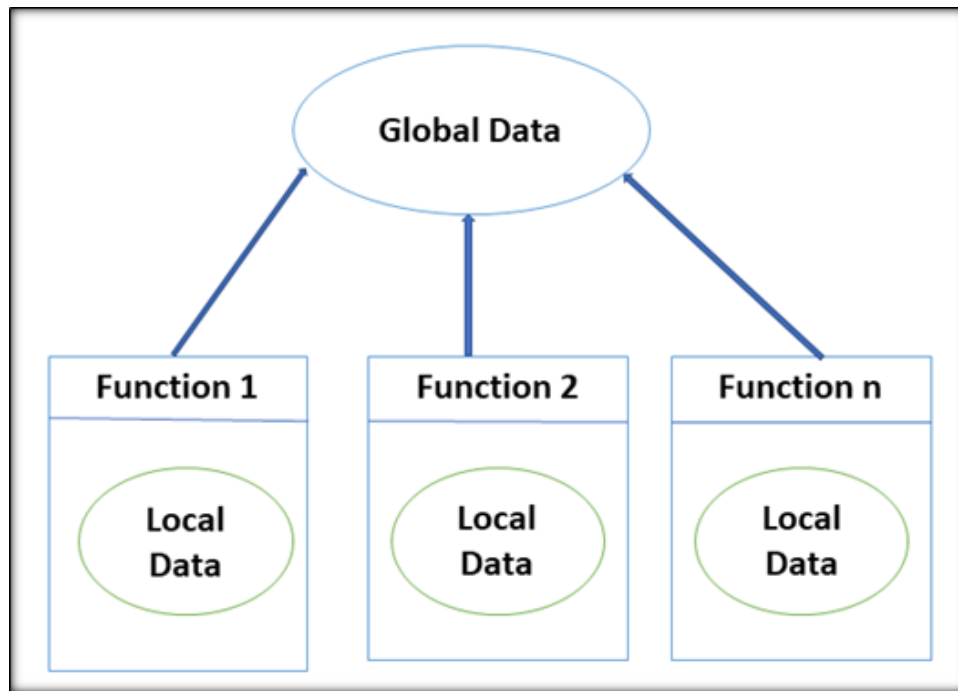


Fig. 2 Local vs. Global data

Each function has its own local data and common global data, as shown in Fig. 2.

### 1.2 Limitations of Procedural Approach:

- The program code is harder to write when Procedural Programming is employed.
- The Procedural code is often not reusable, which may pose the need to recreate the code if it is needed to use in another application
- Difficult to relate with real-world objects
- The importance is given to the operation rather than the data, which might pose issues in some data-sensitive cases
- The data is exposed to the whole program, making it not so much security friendly

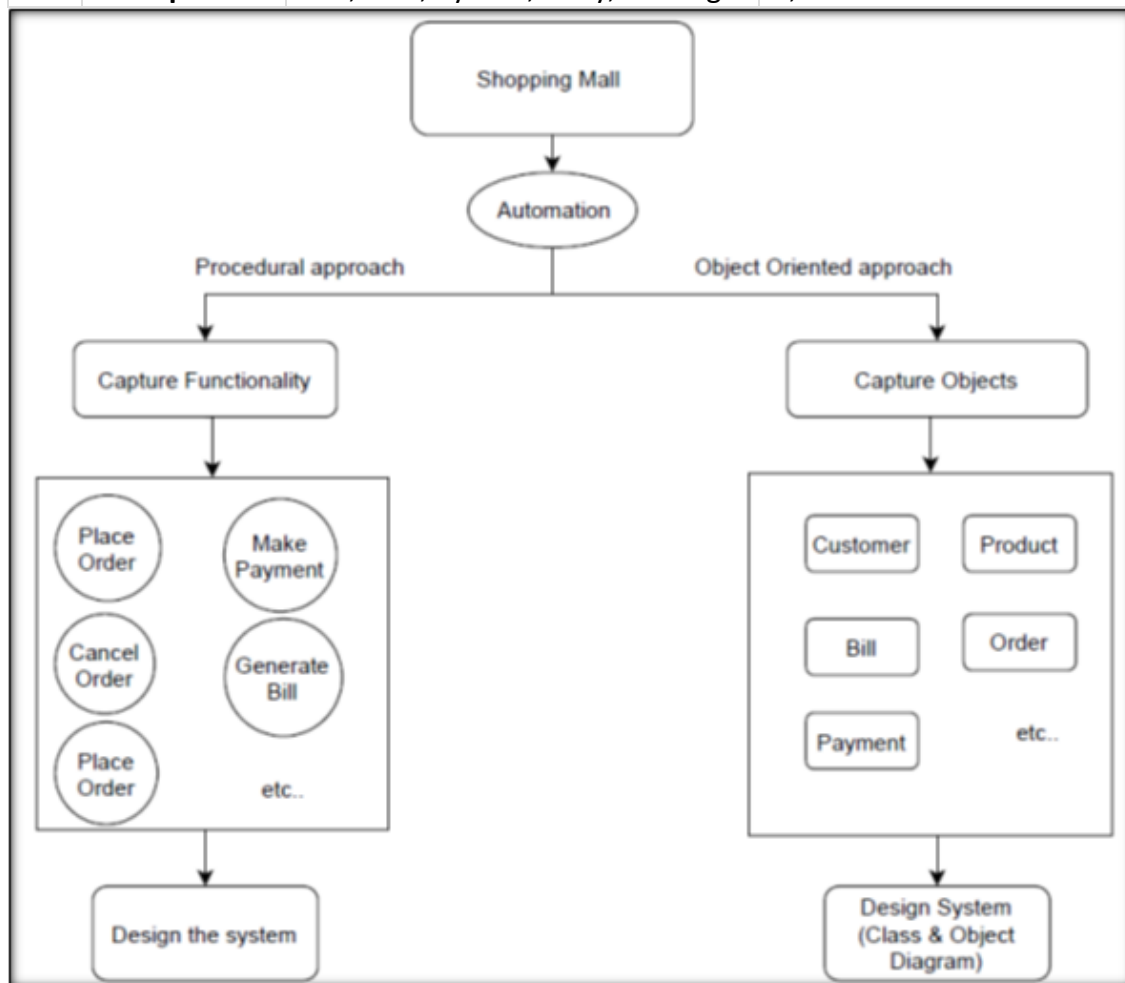
## Lecture - 2

### 2 Object Oriented Approach (OOP)

An OOP method differs from POP in its basic approach itself. All the best features of structured of OOP is developed by retaining the programming method, in which they have added number of concepts which makes efficient programming. Object oriented programming methods have number of features and it makes possible an entirely new way of approaching a program. This has to be noted that OOP retains all best features of POP method like functions/sub routines, structure etc.

## 2.1 Difference between Procedure Oriented and Object-Oriented Approach

S. N.	Features	Object Oriented Approach (OOP)	Procedure Oriented Approach (POP)
1	<b>Definition</b>	OOP stands for Object Oriented Programming.	POP stands for Procedural Oriented Programming.
2	<b>Approach</b>	OOP follows bottom-up approach – focused on building blocks and components.	POP follows top-down approach - focused on inputs and outputs
3	<b>Division</b>	A program is divided to objects and their interactions.	A program is divided into functions and they interact.
4	<b>Inheritance supported</b>	YES	NO support
5	<b>Access control</b>	Access control is supported via access modifiers.	No access modifiers are supported.
6	<b>Data Hiding</b>	Encapsulation is used to hide data.	No data hiding present. Data is globally accessible.
7	<b>Example</b>	C++, Java, Python, Ruby, GoLang	C, Pascal



### Top-Down vs Bottom-Up Approach

The object-oriented programming approach treats the real-world problem in terms of objects. These objects work together to reach the solution of a particular problem. This type

of object-oriented programming follows the bottom-up approach. In bottom-up approach component parts are implemented first then overall system is implemented by integrating the component parts.

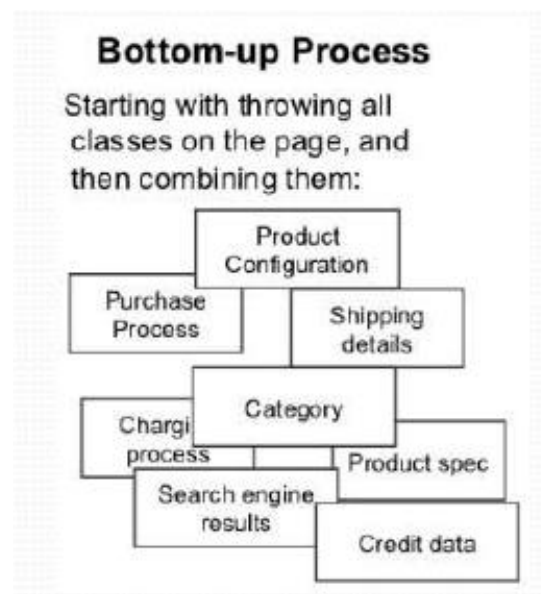
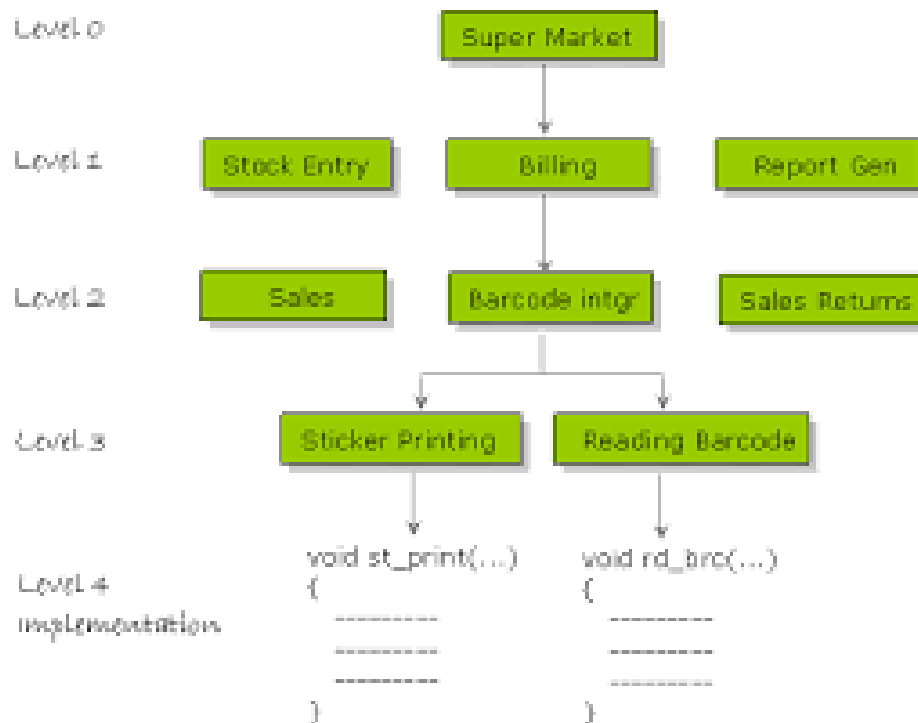
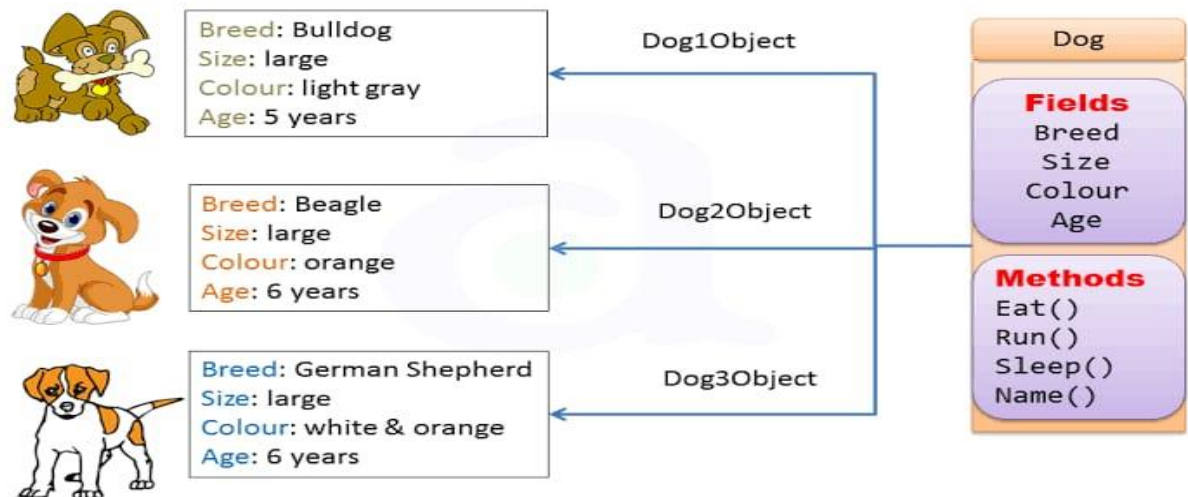


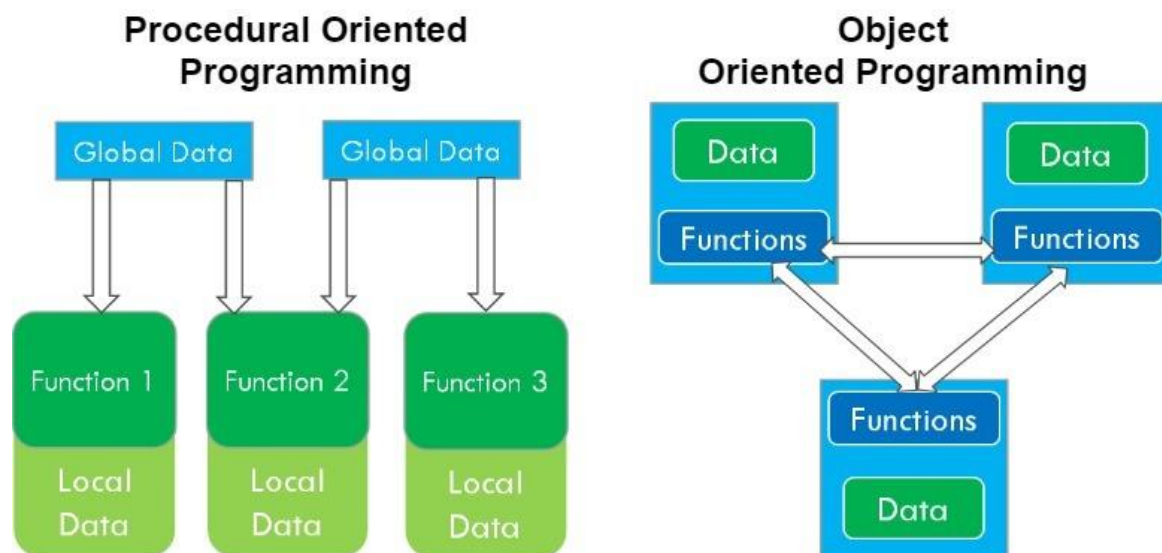
Fig. 1 Top-Down approach vs Bottom-up Approach

### 3. The meaning of Object Orientation:

The object-oriented approach, however, focuses on objects that represent abstract or concrete things in the real world. These objects are first defined by their character and their properties, which are represented by their internal structure and their attributes (data). The behaviour of these objects is described by methods (functions).



Objects form a capsule, which combines the characteristics with behaviour. Objects are intended to enable programmers to map a real problem and its proposed software solution on a one-to-one basis.



Object examples can be, 'Customer', 'Order', or 'Invoice'.

## Lecture - 3

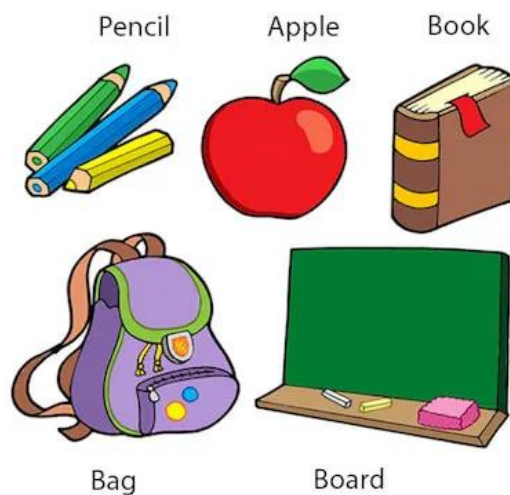
### 4. Concepts of OOP

The OOPs concepts helps the programmers to analyse the real-world problem and implement it in terms of programming language.

## 4.1 Object

An object is anything in the real-world. It exists in the form of physical or abstract. For example, it may represent a person, animal, bird or anything. For example: Elephant, Lion is an object.


### Objects: Real World Examples



#### 4.1.1 Object Identity

Object identity is a fundamental object orientation concept. Object Identity describes the property of Objects that distinguishes it from other Objects.

Identity is a property of an object that distinguishes the object from all other objects in the application.

Object	Identity	Identity
	Car Name: <b>Audi</b> Engine Number: <b>a123</b>	Car Name: <b>Audi</b> Engine Number: <b>a125</b>

## 4.2 Class

Class is a group of similar types of objects, having similar features and behaviour. In the following Figure different objects exists in the real world and based on the common features these are categorized into different classes Car, Flower and Person.

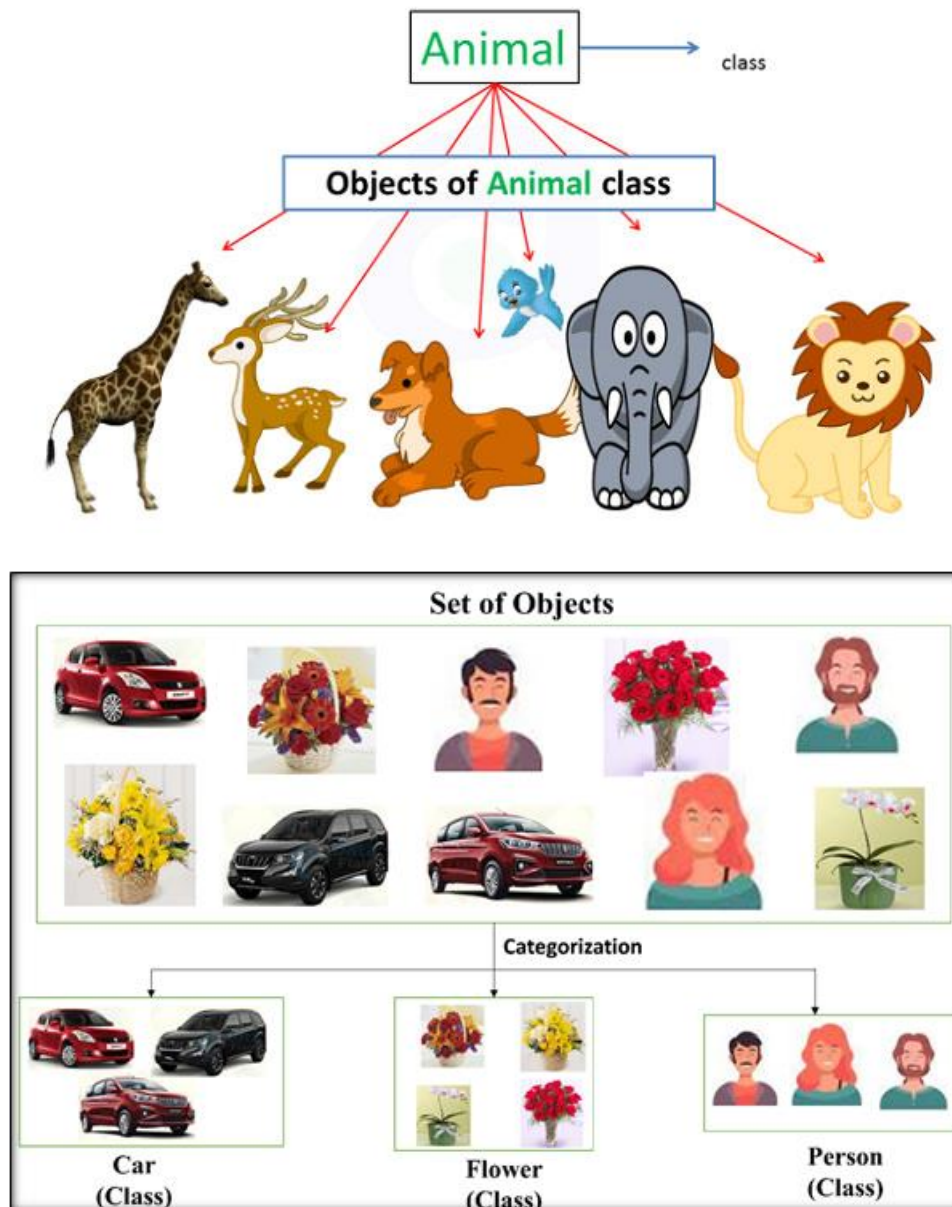


Fig. 2 Class and Object

Class is used to describe the structure of objects that how the objects will look likes. Class is also a pattern or template which produces similar kind of objects.

Class has data members (attributes) and behavior shown by object also called functionality.



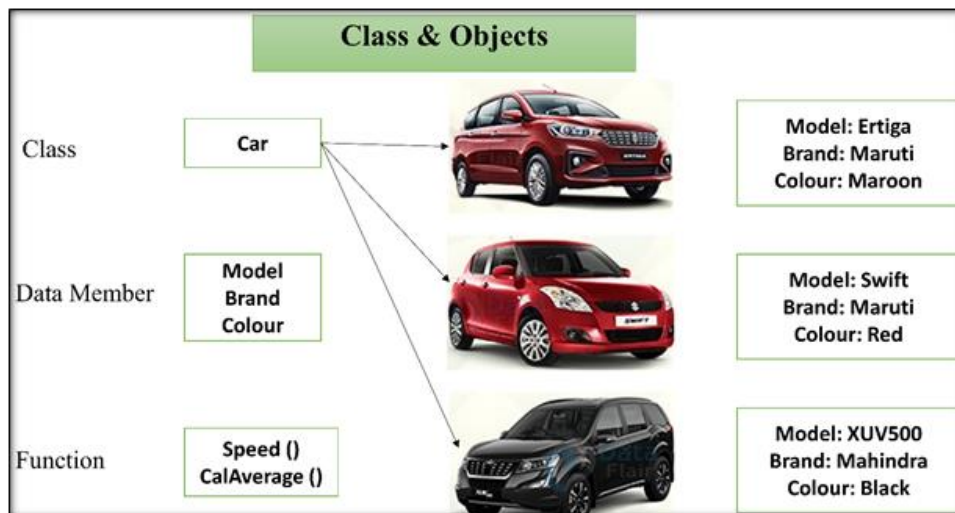


Fig.4 Data-Member and Functions of a Class

### 4.3 Encapsulation

Encapsulation is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates. Another way to think about encapsulation is, it is a protective shield that prevents the data from being accessed by the code outside this shield.

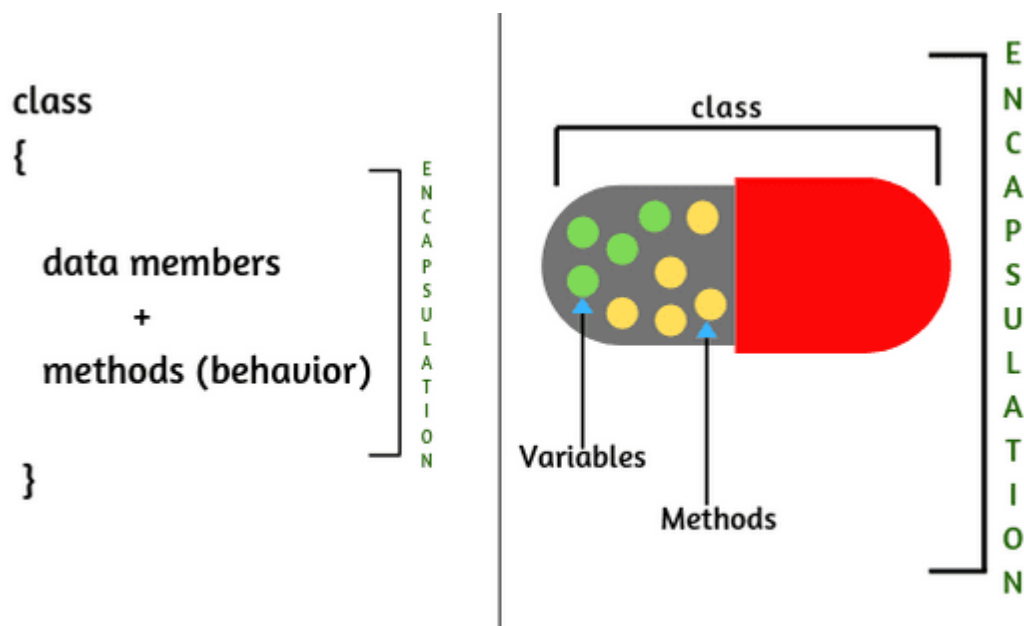


Fig: Encapsulation

#### Examples of Encapsulation:

- School bag is one of the most real examples of Encapsulation. School bag can keep our books, pens, etc.
- In network communication, the data is encapsulated in a single unit in the form of packets. These packets are delivered over the network.





Encapsulation in Object-Oriented Approach:

class Book
String book_title;
String author_name;
+ openBook()
+ closeBook()
+ readBook()

In this data and methods are encapsulated in a class Book.

#### Advantages of Encapsulation:

- The encapsulated code is more flexible and easier to change with new requirements.
- It prevents the other classes to access the private fields.
- Encapsulation allows modifying implemented code without breaking other code that has implemented the code.
- It keeps the data and codes safe from external inheritance. Thus, Encapsulation helps to achieve security.
- It improves the maintainability of the application.

## 4.4 Information Hiding

Information hiding is a powerful OOP feature. Information hiding is closely associated with encapsulation.

Information or data hiding is a programming concept which protects the data from direct modification by other parts of the program.

Information hiding includes a process of combining the data and functions into a single unit to conceal data within a class by restricting direct access to the data from outside the class.

“Hiding object details (state + behaviour) of a class from another class”

### Real Life Example of Information Hiding:

1. My Name and personal information is stored in My Brain, nobody can access this information directly. For getting this information you need to ask me about it and it will be up to myself that how much details I would like to share with you.
2. Facebook may have millions of user accounts. Facebook is just like a Class, it has Private, Public and Protected members. In private, there may be inbox, some personal photo or video. In public, it may be some post, or user information like d.o.b, where you live etc and in protected there may be some post that only your friend can see, and it's hidden from public.

In object-oriented approach we have objects with their attributes and behaviours that are hidden from other classes, so we can say that object-oriented programming follows the principle of information hiding.

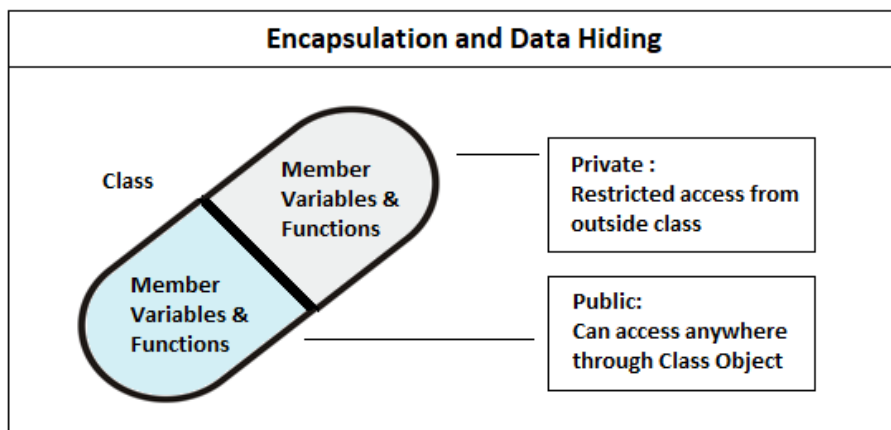


Fig: 6 Encapsulation & Data Hiding

### Demo of information hiding in C++:

```
class Student
{
private:
    char name[30];
    int marks;
public:
    void display();
};

int main()
{
    Student s;
    s.marks = 50; // Compilation Error – Not Accessible as the member is private
```

```
return 0;  
}
```

The accessibility often plays an important role in information hiding. Here the data element *marks* are private element and thus it cannot be accessed by main function or any other function except the member function *display()* of class student. To make it accessible in main function, it should be made a *public* member.

### Advantages of Information Hiding:

1. It ensures exclusive data access and prevents intended or unintended changes in the data.
2. It helps in reducing system complexity and increase the robustness of the program.
3. It heightens the security against hackers that are unable to access confidential data.
4. It prevents programmers from accidental linkage to incorrect data.

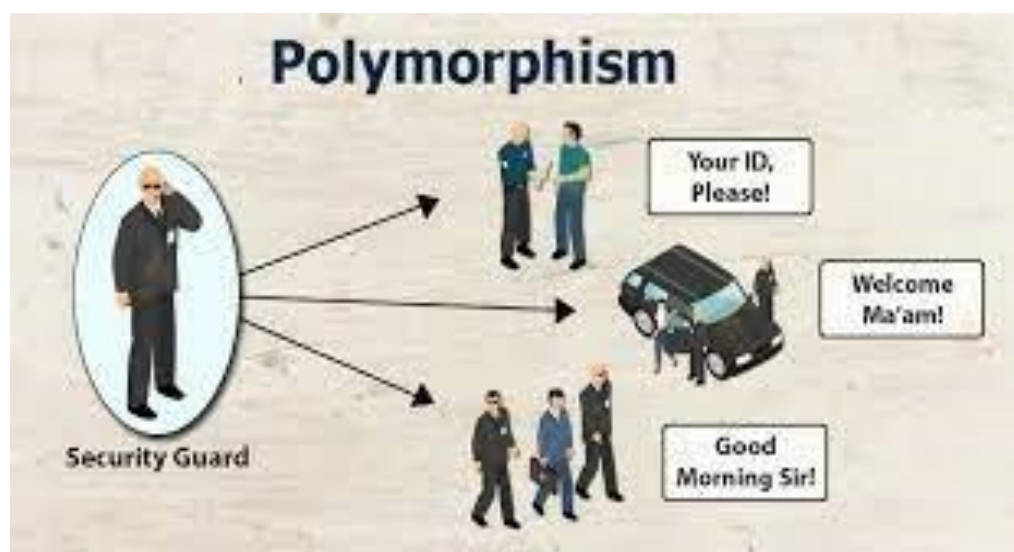
### Disadvantage of Information Hiding:

It may sometimes force the programmer to use extra coding for creating effects for hiding the data.

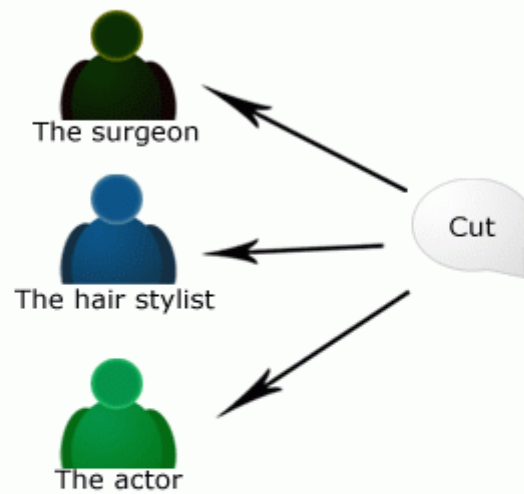
## Lecture-4

### 5. Polymorphism

The word “poly” means many and “morphs” means forms, so the word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.



IF ANY BODY SAYS "CUT" TO THESE PEOPLE



The surgeon would begin to make an incision.

The hair stylist would begin to cut someone's hair.

The actor would abruptly stop acting out the current scene, awaiting directorial guidance.

Fig 7 - Polymorphism

### Real life examples of polymorphism:

A same person (Security Guard) can have different behavior (role) in different situations:

1. Like a security guard during competitive Exam ask students to show Identity.
2. In Shopping Mall security guard says welcome Ma'am or Sir and do not say for identity.
3. In Office guard says Good Morning Sir or Ma'am.

So, the same person shows different behavior in different situations. This is called polymorphism.

## 6. Generosity:

**Generosity** is a property in object-oriented technology by which the container class can be created in which the implementation of any data types is contained. These classes are popularly known as templates.

**Example:** Consider the implementation of data structure stack. If we create an integer stack “stack of integers” then the push and pop operations will handle only integers elements. If we create a character stack “stack of characters” then the push and pop operations will handle only characters. But creating so many copies of implementation make the code complex to maintain. **Hence the principal of generosity is used and a template class can be created. This container class can handle any data type element at run time.**

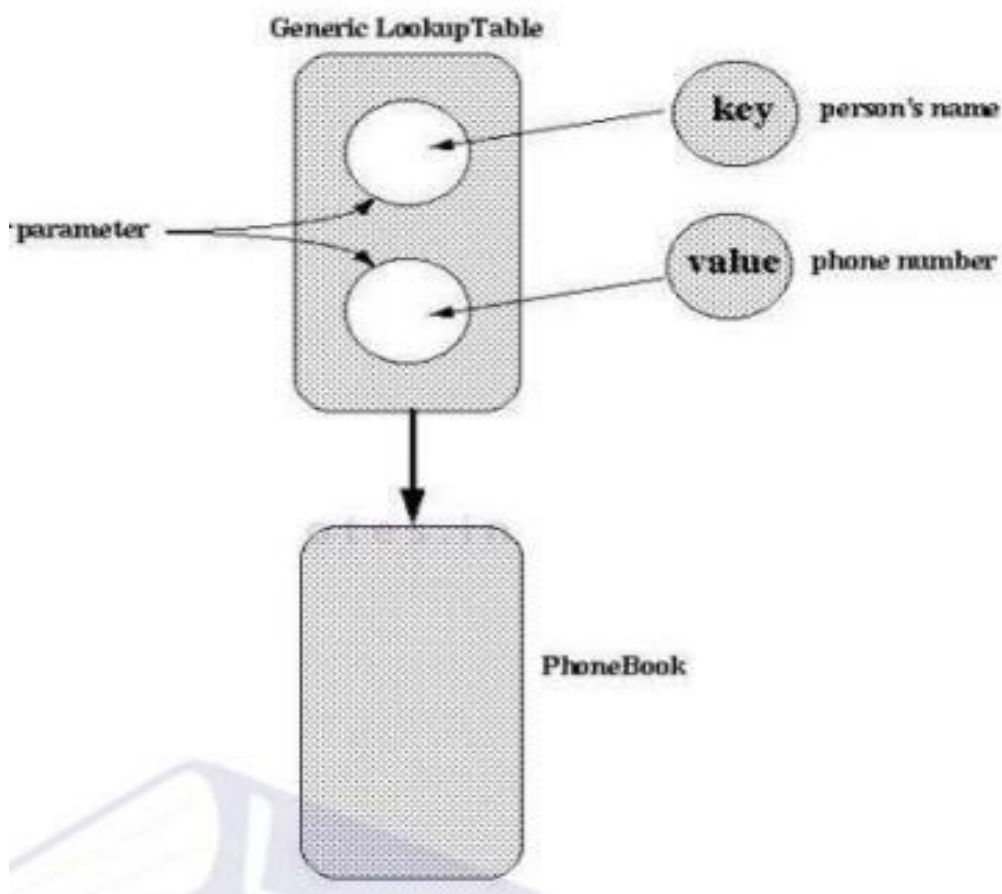
**For example:** In the given figure, the two parameters of a container class (Generic Lookup Table) include key and value. In this case key is person name and value is phone number will yield phone book data and in other cases it can be used as data for other tasks.

**Mapping\_Fun(key)-> Value**

Mapping\_Fun(person\_name)-> phone\_number

Mapping\_Fun(ip\_address)-> domain\_name

Mapping\_Fun(aadhaar\_number)-> pan\_number



## Lecture-5

### Modelling

Modelling is a central part of all activities that lead up to the deployment of good software. It is required to build quality software. We build models to communicate the desired structure and behaviour of our system. We build models to visualize and control the system's architecture. We build models to better understand the system we are building, often exposing opportunities for simplification and reuse. And we build models to manage risk."

For example, If you want to build a house for your family with same things, is it possible? It requires detailed planning, some sketches etc. Of Course, architecting a house is best achieved by a team. It requires detailed modelling, well-defined processes and powerful tools.



What about a high-rise building? Modeling is no doubt a critical part of any construction project!

### 7. Importance of modelling:

Through modelling, we can achieve following aims:

- Modelling gives graphical representation of system to be built.
- Modelling contributes to a successful software organization.
- Modelling is a proven and well accepted engineering technique.
- Modelling is not just a part of the building industry. It would be inconceivable to deploy a new aircraft or an automobile without first building models-from computer models to physical wind tunnels models to full scale prototypes.
- A model is a simplification of reality. A model provides the blueprint of a system.
- A model may be structural, emphasizing the organization of the system, or it may be behavioural, emphasizing the dynamics of the system.
- Models are build for better understanding of the system that we are developing:
  - a. Models help us to visualize a system as it is or as we want it to be.
  - b. Models permit us to specify the structure or behaviour of a system.
  - c. Models give us a template that guides us in constructing a system.
  - d. Models support the decisions we have made.

The larger and more complex the system becomes, the more important modelling becomes, for one very simple reason:

Every project can benefit from modelling. Modelling can help the development team better visualize the plan of their system and allow them to develop more rapidly by helping them build the right thing. The more complex your project, the more likely it is that you will fail or that you will build the wrong thing if you do on modelling at all.

## 8. Principles of modelling:

Modelling principles are as follows:

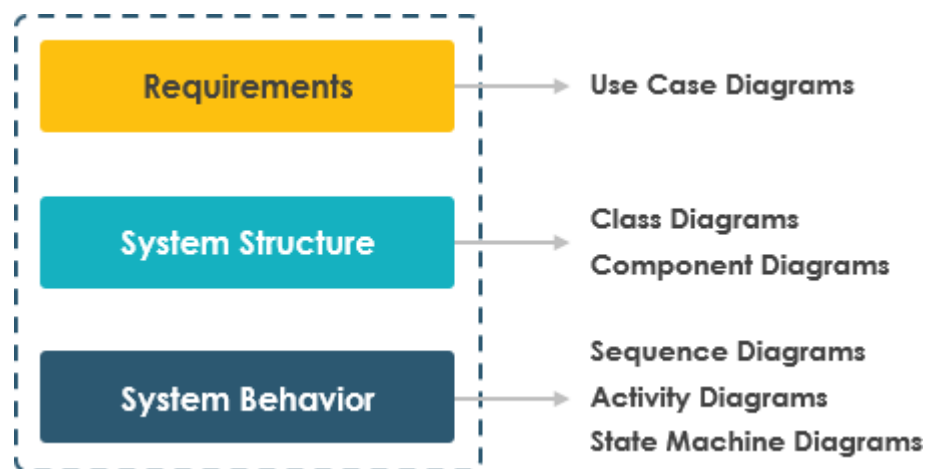
### a. The choice of model is important

“The choice of what models to create has a profound influence on how a problem is attacked and how a solution is shaped”.

We need to choose your models well.

- The right models will highlight the most critical development problems.
- Wrong models will mislead you, causing you to focus on irrelevant issues.

For Example: We can use different types of diagrams for different phases in software development.



### b. Every model may be expressed at different levels of precision

This means all the user and developers both may visualize a system at different levels of details at different time.

For Example,

- If you are building a high rise, sometimes you need a 30,000-foot view for instance, to help your investors visualize its look and feel.
- Other times, you need to get down to the level of the studs for instance, when there's a tricky pipe run or an unusual structural element.



### c. The best models are connected to reality

This means that the model must have things that are practically possible. They must satisfy the real word scenarios. All models simplify reality and a good model reflects important key characteristics.

### d. No single model is sufficient

**“Every nontrivial system is best approached through a small set of nearly independent models:”** This means you need to have use case view, design view, process view, implementation view and development view. Each of these views may have structural as well as behavioural aspects. Together these views represent a system.

Create models that can be built and studied separately, but are still interrelated. In the case of a building:

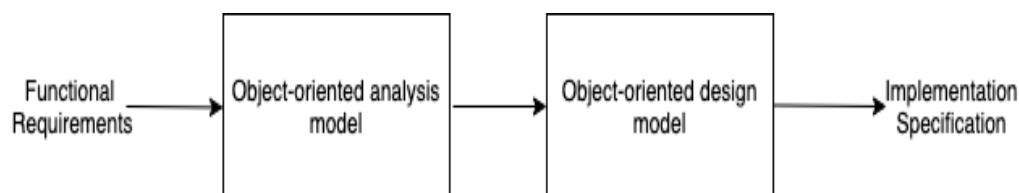
- You can study electrical plans in isolation
- But you can also see their mapping to the floor plan and perhaps even their interaction with the routing of pipes in the plumbing plan.

## 9. Object Oriented Modelling:

Object-oriented modelling is the process of preparing and designing what the model’s code will actually look like. During the construction or programming phase, the modelling techniques are implemented by using a language that supports the object-oriented programming model.

OOM consists of progressively developing object representation through three phases: analysis, design, and implementation. During the initial stages of development, the model developed is abstract because the external details of the system are the central focus. The model becomes more and more detailed as it evolves, while the central focus shifts toward understanding how the system will be constructed and how it should function.

Object-oriented modelling allows for object identification and communication while supporting data abstraction, inheritance and encapsulation.



### Benefits of Object Model

The benefits of using the object model are –

It helps in faster development of software.

It is easy to maintain. Suppose a module develops an error, then a programmer can fix that particular module, while the other parts of the software are still up and running.

It supports relatively hassle-free upgrades.

It enables reuse of objects, designs, and functions.

It reduces development risks, particularly in integration of complex systems.

### **Purpose of Models:**

- Testing a physical entity before building it
- Communication with customers
- Visualization
- Reduction of complexity

### **Types of Models:**

There are 3 types of models in the object oriented modelling and design are:

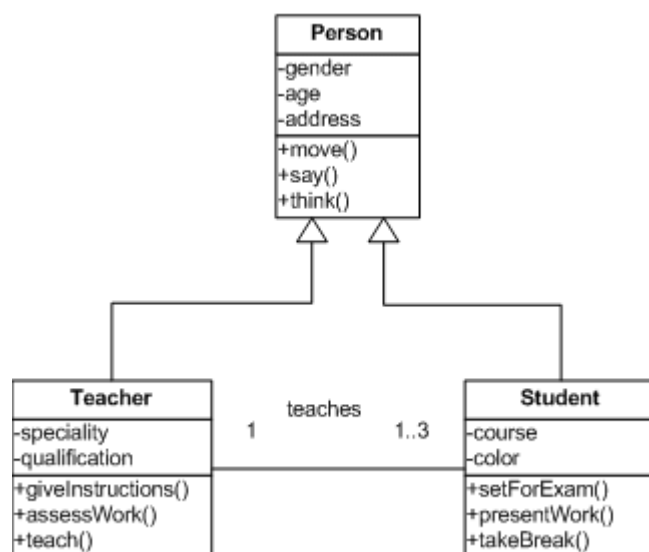
Class Model, State Model, and Interaction Model.

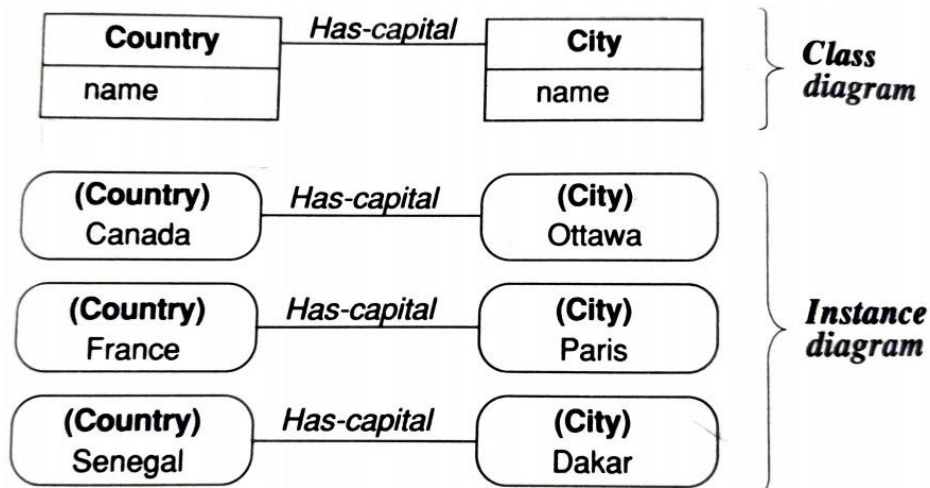
These are explained as following below.

### **Class Model:**

The class model shows all the classes present in the system. The class model shows the attributes and the behaviour associated with the objects.

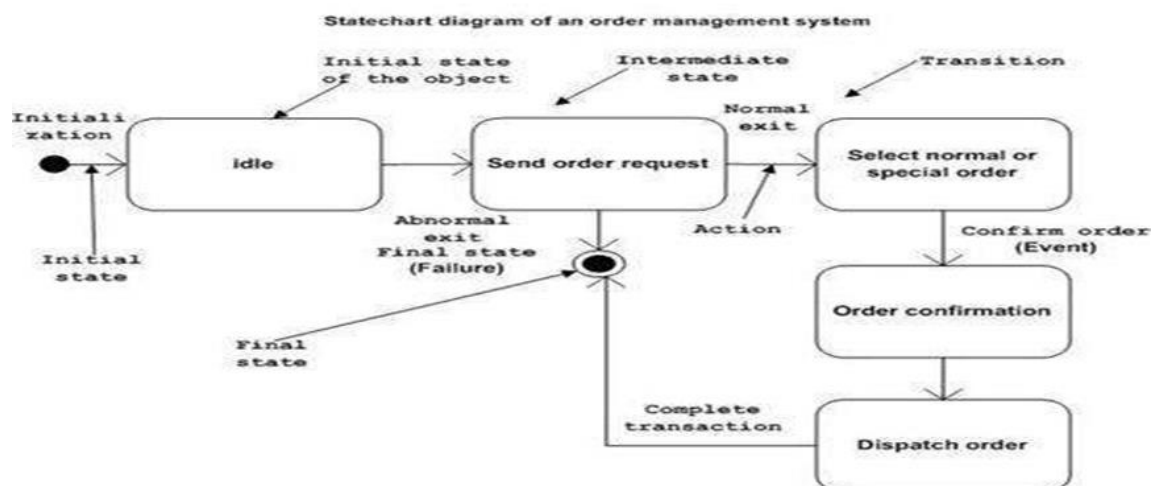
The class diagram is used to show the class model. The class diagram shows the class name followed by the attributes followed by the functions or the methods that are associated with the object of the class. Goal in constructing class model is to capture those concepts from the real world that are important to an application.





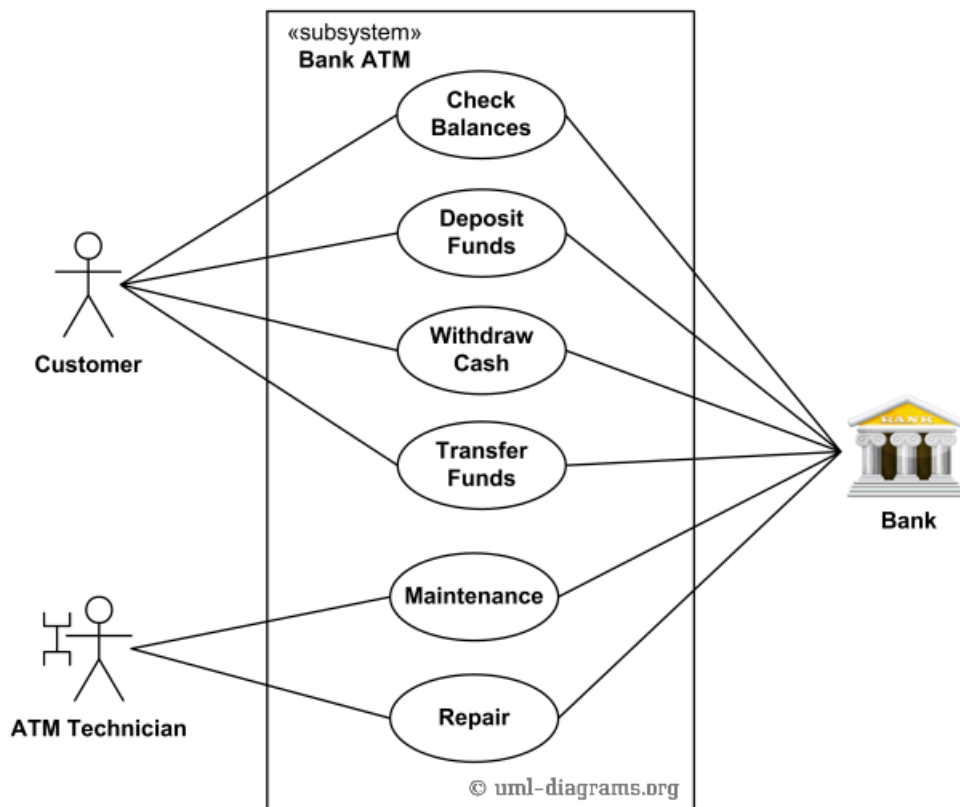
### State Model:

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. Actions and events in a state diagram become operations on objects in the class model. State diagram describes the state model.



### Interaction Model:

Interaction model is used to show the various interactions between objects, how the objects collaborate to achieve the behaviour of the system as a whole.



The following diagrams are used to show the interaction model:

- Use Case Diagram
- Sequence Diagram
- Activity Diagram

## Lecture-6

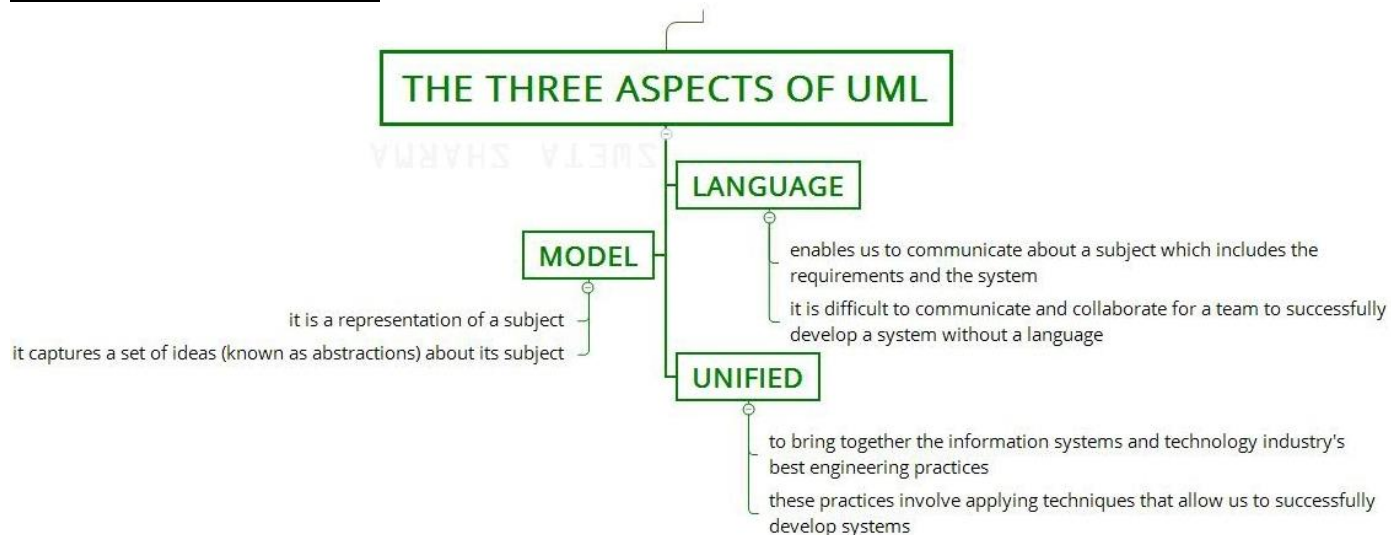
### Unified Modelling Language (UML)

- 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 Modelling 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 modelling language to visualize, specify, construct, and document software system. Although UML is generally used to model software systems, it is not limited within this boundary. It is also used to model

non-software systems as well. For example, the process flow in a manufacturing unit, etc.

- UML is not a programming language but tools can be used to generate code in various languages using UML diagrams. UML has a direct relation with object-oriented analysis and design. After some standardization, UML has become an OMG standard.

### **Three Aspects of UML**



**Figure – Three Aspects of UML**

#### **1. Language:**

- It enables us to communicate about a subject which includes the requirements and the system.
- It is difficult to communicate and collaborate for a team to successfully develop a system without a language.

#### **2. Model:**

- It is a representation of a subject.
- It captures a set of ideas (known as abstractions) about its subject.

#### **3. Unified:**

- It is to bring together the information systems and technology industry's best engineering practices.
- These practices involve applying techniques that allow us to successfully develop systems.

### **Goals of UML**

There are a number of goals for developing UML but the most important is to define some general-purpose modelling language, which all modellers can use and it also needs to be made simple to understand and use.

- UML diagrams are not only made for developers but also for business users, common people, and anybody interested to understand the system.
- The system can be a software or non-software system. Thus it must be clear that UML is not a development method rather it accompanies with processes to make it a successful system.

## **Where Can the UML Be Used?**

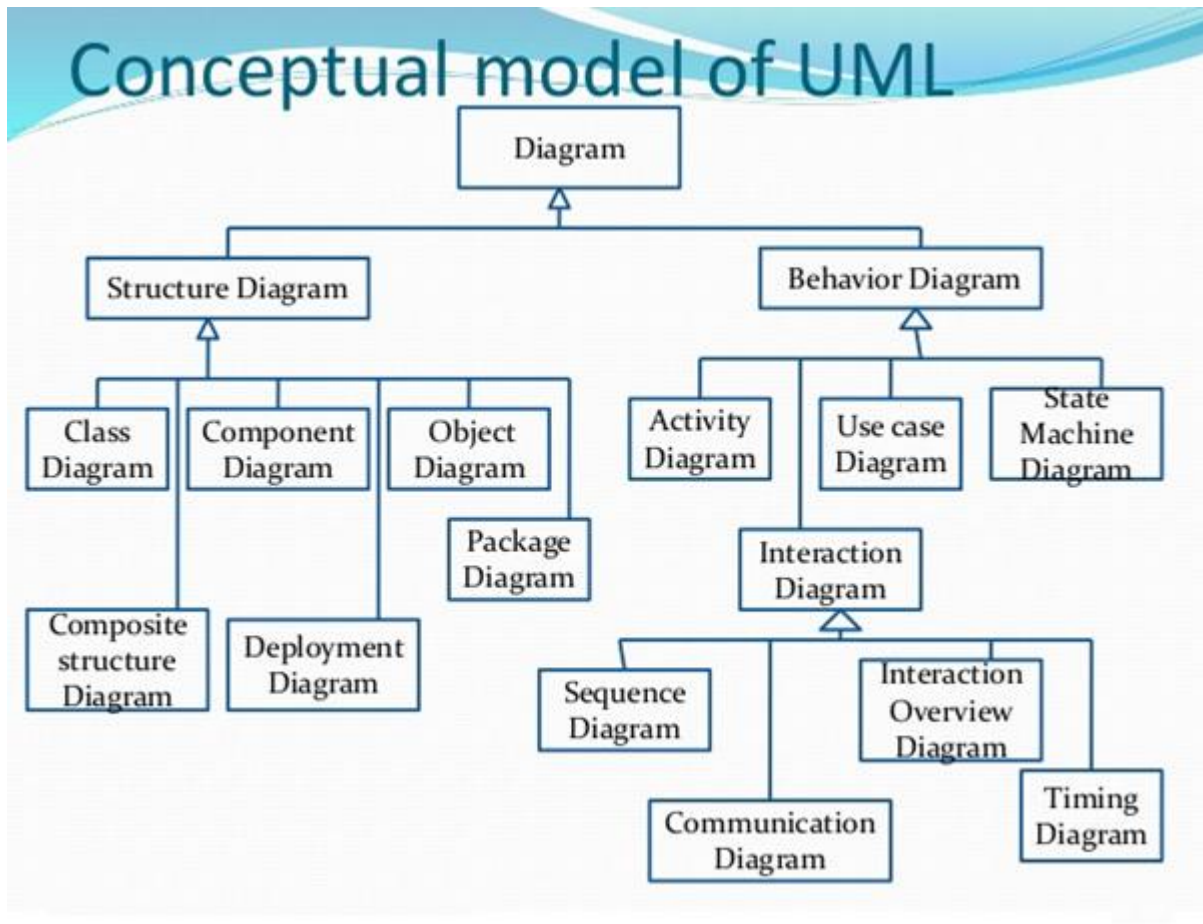
The UML is intended primarily for software-intensive systems. It has been used effectively for such domains as

- Enterprise information systems
- Banking and financial services
- Telecommunications
- Transportation
- Defence/aerospace
- Retail
- Medical electronics
- Scientific
- Distributed Web-based service

## **A Conceptual Model of the UML**

- A conceptual model needs to be formed by an individual to understand UML.
- UML contains three types of building blocks: things, relationships, and diagrams.
  1. Things
    - Structural things
      - Classes, interfaces, collaborations, use cases, components, and nodes.
    - Behavioural things
      - Messages and states.
    - Grouping things
      - Packages
    - Annotation things
      - Notes
  2. Relationships: Dependency, Association, Generalization and Realization.
  3. Diagrams: class, object, use case, sequence, collaboration, state chart, activity, component and deployment.

## Lecture-7

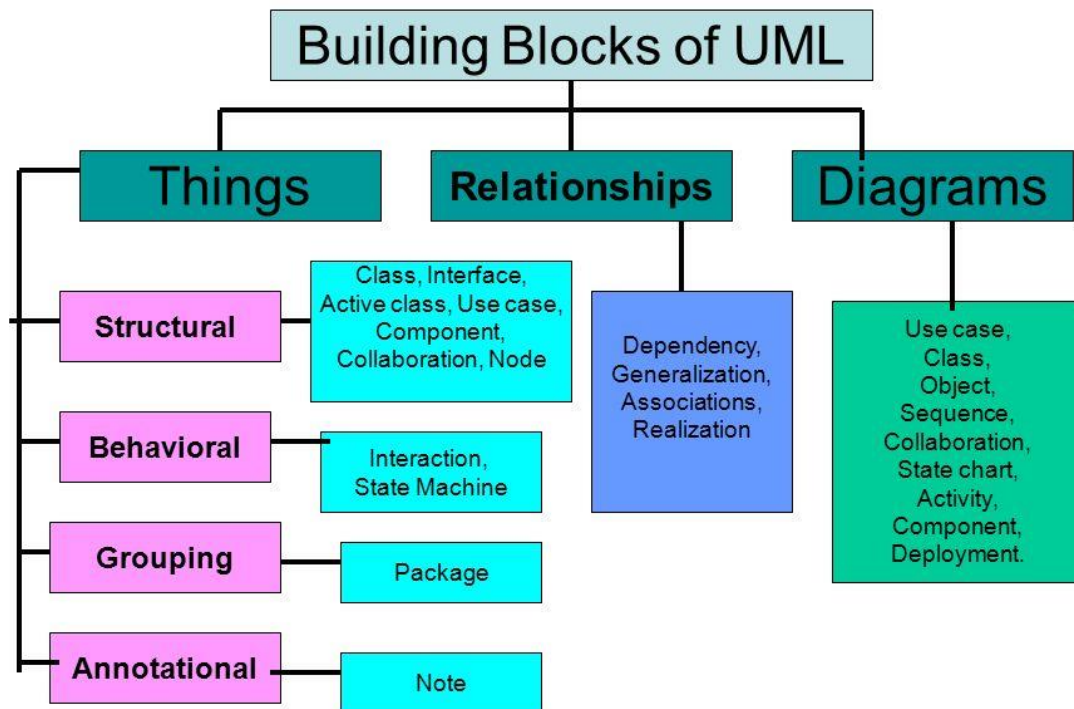


### Building Blocks of the UML

The vocabulary of the UML encompasses three kinds of building blocks:

1. Things
2. Relationships
3. Diagrams





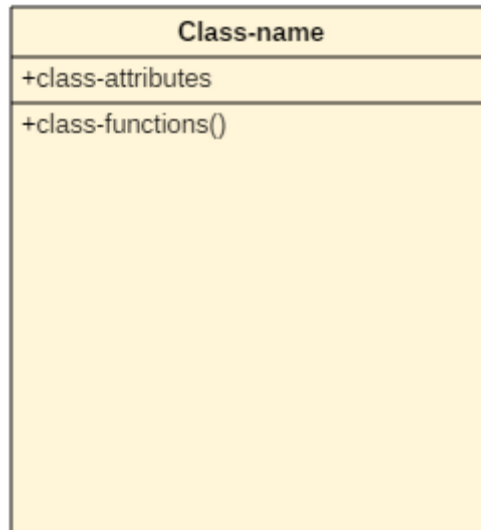
### Things in the UML

There are four kinds of things in the UML:

1. Structural things
2. Behavioural things
3. Grouping things
4. Annotation things

**Structural Things:** Structural things are the nouns of UML models. These are the mostly static parts of a model, representing elements that are either conceptual or physical. In all, there are seven kinds of structural things.

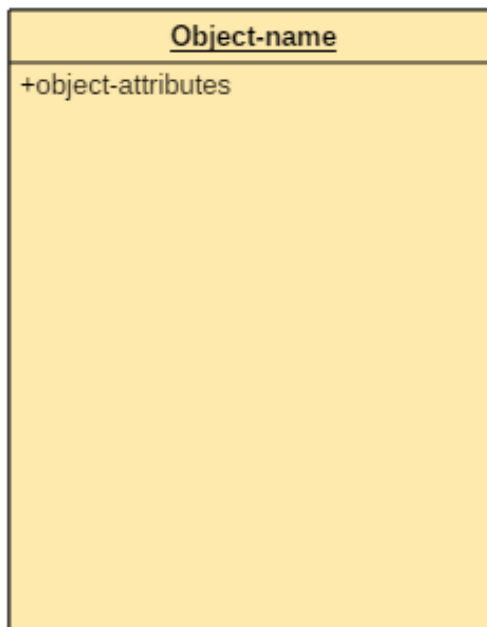
1. **Class:** A class is used to represent set of objects sharing similar properties and behaviour. It is used to define the properties and operations of an object. A class whose functionalities are not defined is called an abstract class. Any UML class diagram notations are generally expressed as below UML class diagrams example,



UML Class Symbol

- 2. Object:** An object is an entity which is used to describe the behaviour and functions of a system. The class and object have the same notations. The only difference is that an object name is always underlined in UML.

The UML notation of any object is given below.



UML Object Symbol

- 3. Interface:** An interface is similar to a template without implementation details. A circle notation represents it. When a class implements an interface, its functionality is also implemented.



### UML Interface Symbol

4. **Collaboration:** It is represented by a dotted ellipse with a name written inside it.



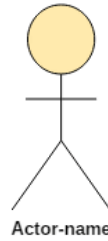
### UML Collaboration Notation

5. **Use-case:** Use-cases are one of the core concepts of object-oriented modelling. They are used to represent high-level functionalities and how the user will handle the system.



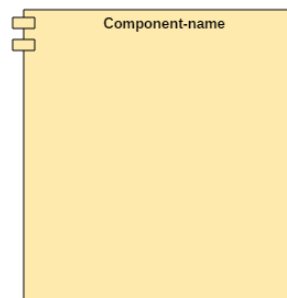
### UML Use Case

6. **Actor:** It is used inside use case diagrams. The Actor notation is used to denote an entity that interacts with the system. A user is the best example of an actor. The actor notation in UML is given below.



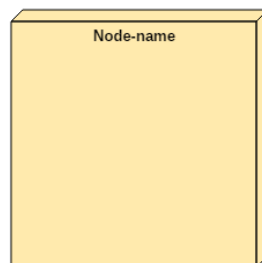
**UML Actor**

7. **Component:** A component notation is used to represent a part of the system. It is denoted in UML like given below,



**UML Component**

8. **Node:** A node is used to describe the physical part of a system. A node can be used to represent a network, server, routers, etc. Its notation is given below.



**UML Node**

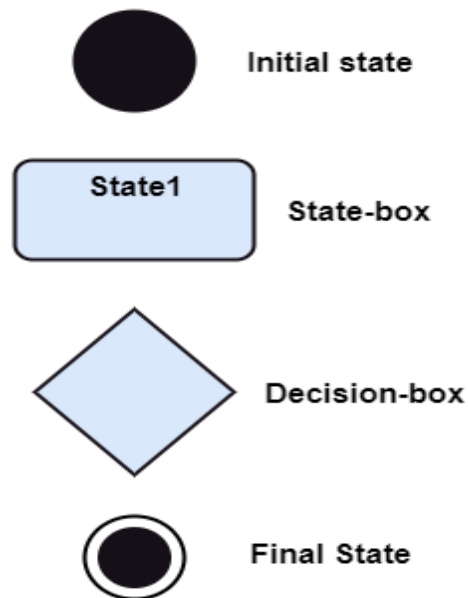
### **Behavioural Things:**

Behavioural things are the dynamic parts of UML models. These are the verbs of a model, representing behaviour over time and space. In all, there are two primary kinds of behavioural things.

1. **Messages:** An interaction is a behaviour that comprises a set of messages exchanged among a set of objects within a particular context to accomplish a specific purpose. Graphically, a message is rendered as a directed line, almost always including the name of its operation display.



2. **States:** A state machine is a behaviour that specifies the sequences of states an object or an interaction goes through during its lifetime in response to events, together with its responses to those events.



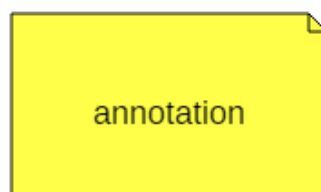
**Grouping Things:** Grouping things are the organizational parts of UML models. These are the boxes into which a model can be decomposed. There is one primary kind of grouping thing, namely, packages.

**Packages:** A package is a general-purpose mechanism for organizing elements into groups. Graphically, a package is rendered as a tabbed folder, usually including only its name and, sometimes, its contents.



**Annotational Things:** Annotational things are the explanatory parts of UML models. These are the comments you may apply to describe, illuminate, and remark about any element in a model.

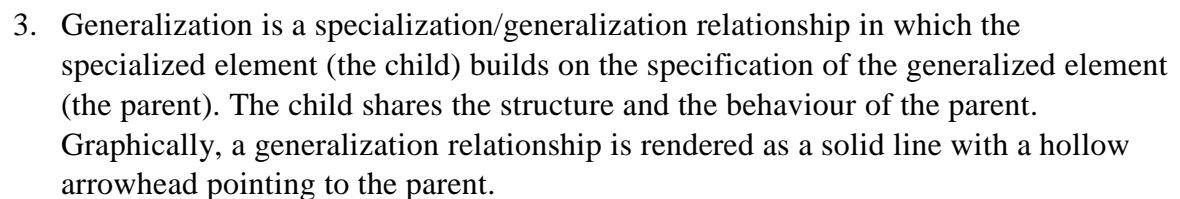
There is one primary kind of annotation thing, called a note. A note is simply a symbol for rendering constraints and comments attached to an element or a collection of elements.



It illustrates the meaningful connections between things. It shows the association between the entities and defines the functionality of an application. There are four kinds of relationships in the UML:

1. Dependency is a semantic relationship between two model elements in which a change to one element (the independent one) may affect the semantics of the other element (the dependent one). Graphically, a dependency is rendered as a dashed line, possibly directed, and occasionally including a label.

2. Association is a structural relationship among classes that describes a set of links, a link being a connection among objects that are instances of the classes. Graphically, an association is rendered as a solid line, possibly directed, occasionally including a label, and often containing other adornments, such as multiplicity and end names.



4. Realization is a semantic relationship between classifiers, wherein one classifier specifies a contract that another classifier guarantees to carry out. Generalization and a dependency relationship.

- - - - - Realization - - - 

## Lecture-8

### UML Diagrams

Any real-world system is used by different users. The users can be developers, testers, business people, analysts, and many more. Hence, before designing a system, the architecture is made with different perspectives in mind. The most important part is to visualize the system from the perspective of different viewers. The better we understand the better we can build the system.

A diagram is the graphical presentation of a set of elements, most often rendered as a connected graph of vertices (things) and paths (relationships).

- A diagram represents an elided view of the elements that make up a system.
- In theory, a diagram may contain any combination of things and relationships.

UML plays an important role in defining different perspectives of a system. These perspectives are –

- Design
- Implementation
- Process
- Deployment

❖ The center is the Use Case view which connects all these four. A Use Case represents the functionality of the system. Hence, other perspectives are connected with use case.

❖ Design of a system consists of classes, interfaces, and collaboration. UML provides class diagram, object diagram to support this.

❖ Implementation defines the components assembled together to make a complete physical system. UML component diagram is used to support the implementation perspective.

❖ Process defines the flow of the system. Hence, the same elements as used in Design are also used to support this perspective.

❖ Deployment represents the physical nodes of the system that forms the hardware.

UML deployment diagram is used to support this perspective. The UML includes nine kinds of diagrams:

1. Class diagram
2. Object diagram
3. Use case diagram
4. Sequence diagram
5. Collaboration diagram
6. State-chart diagram
7. Activity diagram
8. Component diagram

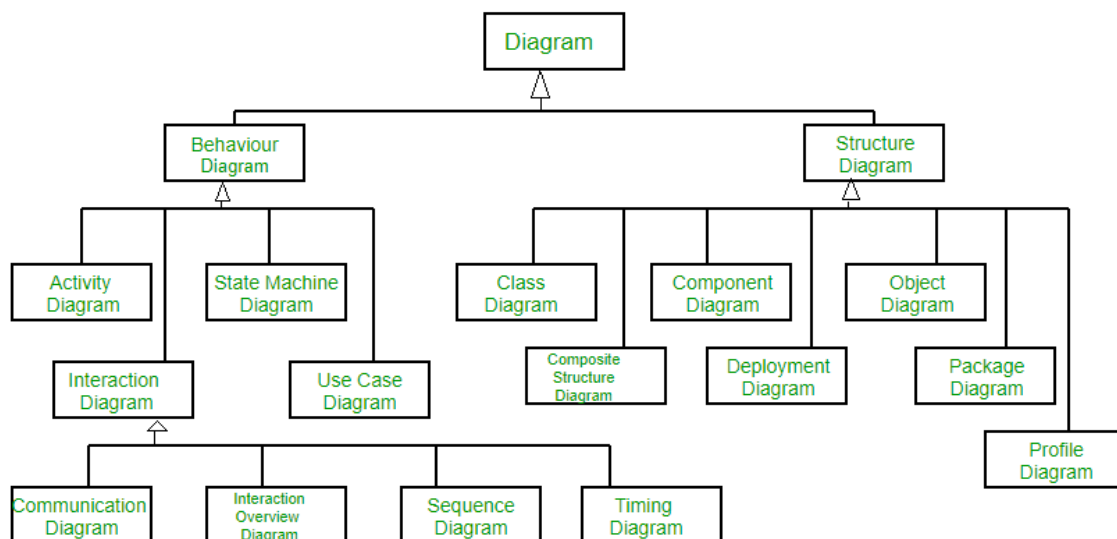
### Role of UML in Object Oriented (OO) Design

- UML is a modelling language used to model software and non-software systems.
- The emphasis is on modelling OO software applications.



- The relation between OO design and UML is very important to understand. The OO design is transformed into UML diagrams according to the requirement. Diagrams in UML can be broadly classified as:
  - **Structural Diagrams** – Capture static aspects or structure of a system. Structural Diagrams include: Component Diagrams, Object Diagrams, Class Diagrams and Deployment Diagrams.
  - **Behaviour Diagrams** – Capture dynamic aspects or behaviour of the system. Behaviour diagrams include: Use Case Diagrams, State Diagrams, Activity Diagrams and Interaction Diagrams.

The image below shows the hierarchy of diagrams according to UML



### Object Oriented Concepts Used in UML –

- **Class** – A class defines the blue print i.e. structure and functions of an object.
- **Objects** – Objects help us to decompose large systems and help us to modularize our system. Modularity helps to divide our system into understandable components so that we can build our system piece by piece. An object is the fundamental unit (building block) of a system which is used to depict an entity.
- **Inheritance** – Inheritance is a mechanism by which child classes inherit the properties of their parent classes.
- **Abstraction** – Mechanism by which implementation details are hidden from user.
- **Encapsulation** – Binding data together and protecting it from the outer world is referred to as encapsulation.
- **Polymorphism** – Mechanism by which functions or entities are able to exist in different forms.

## **Structural UML Diagrams –**

1. **Class Diagram** – The most widely use UML diagram is the class diagram. It is the building block of all object-oriented software systems. We use class diagrams to depict the static structure of a system by showing system's classes, their methods and attributes. Class diagrams also help us identify relationship between different classes or objects.
2. **Composite Structure Diagram** – We use composite structure diagrams to represent the internal structure of a class and its interaction points with other parts of the system. A composite structure diagram represents relationship between parts and their configuration which determine how the classifier (class, a component, or a deployment node) behaves. They represent internal structure of a structured classifier making the use of parts, ports, and connectors. We can also model collaborations using composite structure diagrams. They are similar to class diagrams except they represent individual parts in detail as compared to the entire class.
3. **Object Diagram** – An Object Diagram can be referred to as a screenshot of the instances in a system and the relationship that exists between them. Since object diagrams depict behaviour when objects have been instantiated, we are able to study the behaviour of the system at a particular instant. An object diagram is similar to a class diagram except it shows the instances of classes in the system. We depict actual classifiers and their relationships making the use of class diagrams. On the other hand, an Object Diagram represents specific instances of classes and relationships between them at a point of time.
4. **Component Diagram** – Component diagrams are used to represent the how the physical components in a system have been organized. We use them for modelling implementation details. Component Diagrams depict the structural relationship between software system elements and help us in understanding if functional requirements have been covered by planned development. Component Diagrams become essential to use when we design and build complex systems. Interfaces are used by components of the system to communicate with each other.
5. **Deployment Diagram** – Deployment Diagrams are used to represent system hardware and its software. It tells us what hardware components exist and what software components run on them. We illustrate system architecture as distribution of software artifacts over distributed targets. An artifact is the information that is generated by system software. They are primarily used when a software is being used, distributed or deployed over multiple machines with different configurations.
6. **Package Diagram** – We use Package Diagrams to depict how packages and their elements have been organized. A package diagram simply shows us the dependencies between different packages and internal composition of packages. Packages help us to organise UML diagrams into meaningful groups and make the diagram easy to understand. They are primarily used to organise class and use case diagrams.

## **Behaviour Diagrams –**

1. **State Machine Diagrams** – A state diagram is used to represent the condition of the system or part of the system at finite instances of time. It's a behavioural

diagram and it represents the behaviour using finite state transitions. State diagrams are also referred to as State machines and State-chart Diagrams. These terms are often used interchangeably. So simply, a state diagram is used to model the dynamic behaviour of a class in response to time and changing external stimuli.

2. **Activity Diagrams** – We use Activity Diagrams to illustrate the flow of control in a system. We can also use an activity diagram to refer to the steps involved in the execution of a use case. We model sequential and concurrent activities using activity diagrams. So, we basically depict workflows visually using an activity diagram. An activity diagram focuses on condition of flow and the sequence in which it happens. We describe or depict what causes a particular event using an activity diagram.
3. **Use Case Diagrams** – Use Case Diagrams are used to depict the functionality of a system or a part of a system. They are widely used to illustrate the functional requirements of the system and its interaction with external agents (actors). A use case is basically a diagram representing different scenarios where the system can be used. A use case diagram gives us a high-level view of what the system or a part of the system does without going into implementation details.
4. **Sequence Diagram** – A sequence diagram simply depicts interaction between objects in a sequential order i.e., the order in which these interactions take place. We can also use the terms event diagrams or event scenarios to refer to a sequence diagram. Sequence diagrams describe how and in what order the objects in a system function. These diagrams are widely used by businessmen and software developers to document and understand requirements for new and existing systems.
5. **Communication Diagram** – A Communication Diagram (known as Collaboration Diagram in UML 1.x) is used to show sequenced messages exchanged between objects. A communication diagram focuses primarily on objects and their relationships. We can represent similar information using Sequence diagrams. However, communication diagrams represent objects and links in a free form.
6. **Timing Diagram** – Timing Diagram are a special form of Sequence diagrams which are used to depict the behaviour of objects over a time frame. We use them to show time and duration constraints which govern changes in states and behaviour of objects.
7. **Interaction Overview Diagram** – An Interaction Overview Diagram models a sequence of actions and helps us simplify complex interactions into simpler occurrences. It is a mixture of activity and sequence diagrams.

## **Lecture-9**

### **UML- Architecture**

Software architecture is all about how a software system is built at its highest level. It is needed to think big from multiple perspectives with quality and design in mind. The software team is tied to many practical concerns, such as:

- The structure of the development team.
- The needs of the business.
- Development cycle.
- The intent of the structure itself.

Software architecture provides a basic design of a complete software system. It defines the elements included in the system, the functions each element has, and how each element relates to one another. In short, it is a big picture or overall structure of the whole system, how everything works together.

To form an architecture, the software architect will take several factors into consideration:

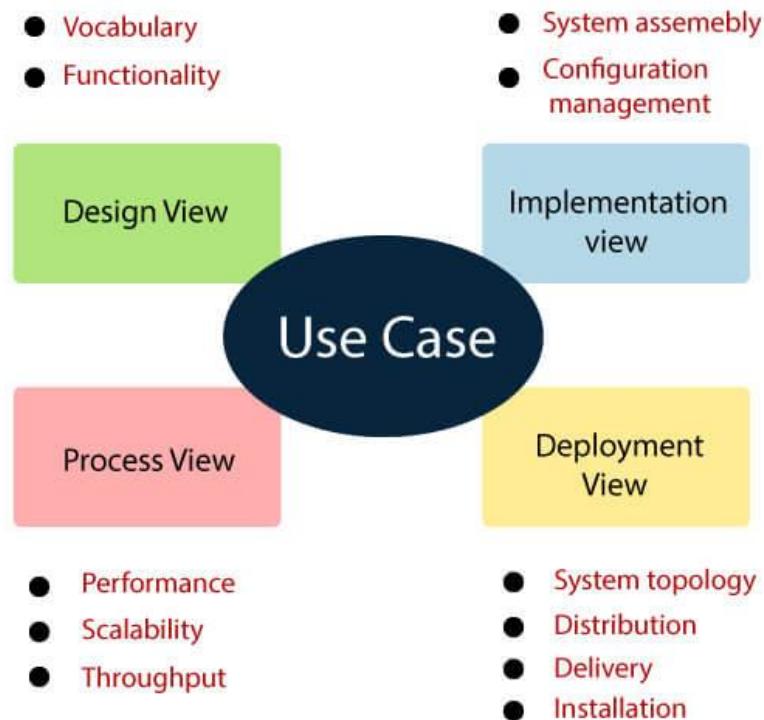
- What will the system be used for?
- Who will be using the system?
- What quality matters to them?
- Where will the system run?

The architect plans the structure of the system to meet the needs like these. It is essential to have proper software architecture, mainly for a large software system. Having a clear design of a complete system as a starting point provides a solid basis for developers to follow.

Each developer will know what needs to be implemented and how things relate to meet the desired needs efficiently. One of the main advantages of software architecture is that it provides high productivity to the software team. The software development becomes more effective as it comes up with an explained structure in place to coordinate work, implement individual features, or ground discussions on potential issues. With a lucid architecture, it is easier to know where the key responsibilities are residing in the system and where to make changes to add new requirements or simply fixing the failures.

In addition, a clear architecture will help to achieve quality in the software with a well-designed structure using principles like separation of concerns; the system becomes easier to maintain, reuse, and adapt. The software architecture is useful to people such as software developers, the project manager, the client, and the end-user. Each one will have different perspectives to view the system and will bring different agendas to a project. Also, it provides a collection of several views. It can be best understood as a collection of five views:

1. Use case view
2. Design view
3. Implementation view
4. Process view
5. Development view



#### Use case view

1. It is a view that shows the functionality of the system as perceived by external actors.
2. It reveals the requirements of the system.

#### Design View

1. It is a view that shows how the functionality is designed inside the system in terms of static structure and dynamic behavior.
2. It captures the vocabulary of the problem space and solution space.
3. With UML, it represents the static aspects of this view in class and object diagrams, whereas its dynamic aspects are captured in interaction diagrams, state chart diagrams, and activity diagrams.

#### Implementation View

1. It is the view that represents the organization of the core components and files.
2. It primarily addresses the configuration management of the system's releases.
3. With UML, its static aspects are expressed in component diagrams, and the dynamic aspects are captured in interaction diagrams, state chart diagrams, and activity diagrams.

#### Process View

1. It is the view that demonstrates the concurrency of the system.

2. It incorporates the threads and processes that make concurrent system and synchronized mechanisms.
3. It primarily addresses the system's scalability, throughput, and performance.
4. Its static and dynamic aspects are expressed the same way as the design view but focus more on the active classes that represent these threads and processes.

#### Deployment View

1. It is the view that shows the deployment of the system in terms of physical architecture.
2. It includes the nodes, which form the system hardware topology where the system will be executed.
3. It primarily addresses the distribution, delivery, and installation of the parts that build the physical system.