**Practical No 1**

**Aim: Introduction To UML**

Unified Modelling Language (UML) is a general-purpose modelling 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 behaviour and structure of a system.

UML helps software engineers, businessmen and system architects with modelling, design and analysis.

The Object Management Group (OMG) adopted Unified Modelling Language as a standard in 1997.

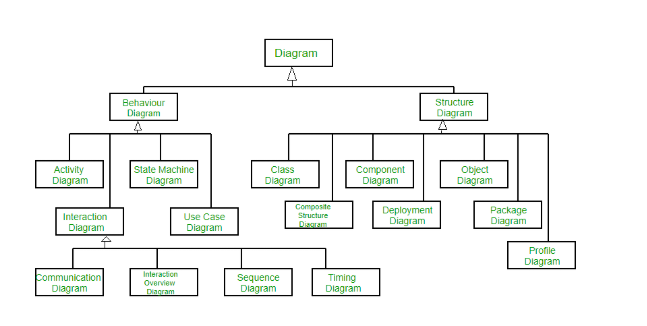
It’s been managed by OMG ever since. International Organization for Standardization (ISO) published UML as an approved standard in 2005.

UML has been revised over the years and is reviewed periodically.

UML is linked with object-oriented design and analysis.

UML makes the use of elements and forms associations between them to form diagrams. Diagrams in UML can be broadly classified as:

* **Structural Diagrams**: Capture static aspects or structure of a system. Structural Diagrams include: Component Diagrams, Object Diagrams, Class Diagrams and Deployment Diagrams.
* **Behavior Diagrams**: Capture dynamic aspects or behavior of the system. Behavior diagrams include: Use Case Diagrams, State Diagrams, Activity Diagrams and Interaction Diagrams.



The image below shows the hierarchy of diagrams according to UML 2.2

**Object Oriented Concepts Used in UML are**

* **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 Consists Of**

* **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.
* **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. 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.
* **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. 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.
* **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.
* **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.
* **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.

**Behavior Diagrams Consists Of**

* **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 behavioral diagram and it represents the behavior 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 behavior of a class in response to time and changing external stimuli.
* **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.
* **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.
* **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.
* **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.
* **Timing Diagram –** Timing Diagram are a special form of Sequence diagrams which are used to depict the behavior of objects over a time frame. We use them to show time and duration constraints which govern changes in states and behavior of objects.
* **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.

**Conclusion: We have studied about the introduction to UML in details.**