INTRODUCTION:

The Unified Modeling Language (UML) is a modeling language that can be used for any purpose. The primary goal of UML is to establish a standard for visualizing the design of a system. It looks a lot like designs in other branches of engineering. UML is a visual language rather than a programming language. UML diagrams are used to depict a system's behavior and structure. UML is a modeling, design, and analysis tool for software engineers, businesspeople, and system architects. Unified Modelling Language was approved as a standard by the Object Management Group in 1997. Since then, OMG has been in charge of it. In 2005, the International Organization for Standardization (ISO) accepted UML as a standard. UML has been updated throughout time and is examined on a regular basis.

UNIFIED MODELING LANGUAGE (UML):

The Unified Modeling Language (UML) was developed to establish a common visual modeling language for the architecture, design, and implementation of large software systems' structure and behavior. UML has applications outside of software development, such as industrial processes. It consists of many types of diagrams and is similar to blueprints used in other domains. UML diagrams, in general, depict the boundaries, structure, and behavior of a system, as well as the objects contained within it. Although UML is not a programming language, there exist tools that produce code in multiple languages using UML diagrams.

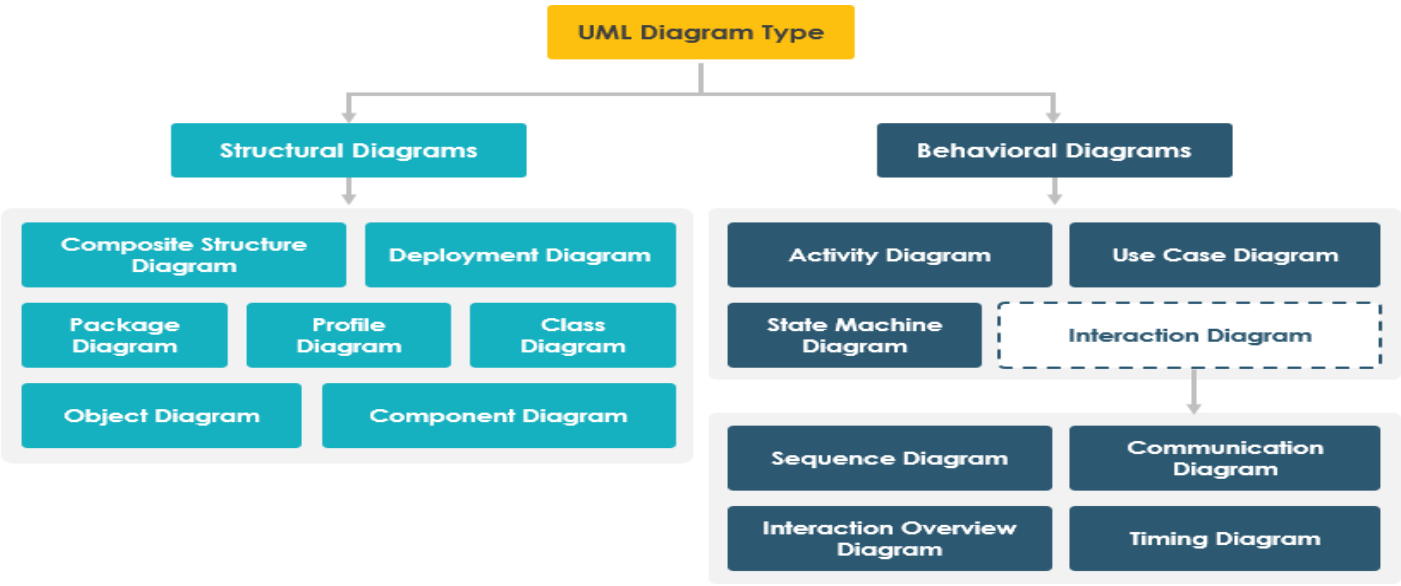
UML is linked with

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:

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

Object-oriented design and analysis are linked to UML. To create diagrams, UML takes items and creates associations between them. The following are some examples of UML diagrams.

- A **structural diagram** depicts a system's static characteristics or structure. Diagrams of structure are shown. Component diagrams, Object diagrams, Class diagrams, and Deployment diagrams are all examples of diagrams used in software development.
- A **behavior diagram** depicts a system's dynamic features or behavior. Diagrams of behavior are included. Use case diagrams, state diagrams, activity diagrams, and interaction diagrams to help you visualize your ideas.



- **Class**: A class defines the blueprint, i.e. the structure and functioning of an object, and is used in UML.
- **Objects**: Objects assist us in breaking down and modularizing complex systems. Modularity allows us to break down our system into easily understandable components, allowing us to create it piece by piece. The basic units (building blocks) of a system are objects, which are used to describe an entity.
- **Inheritance**: a mechanism that allows a child class to inherit the properties of its parent class.
- **Abstraction**: a method that shields the user from implementation specifics.
- **Encapsulation**: the process of putting data together and protecting it from the outside world.
- **Polymorphism**: a method that allows a function or entity to exist in several versions

.

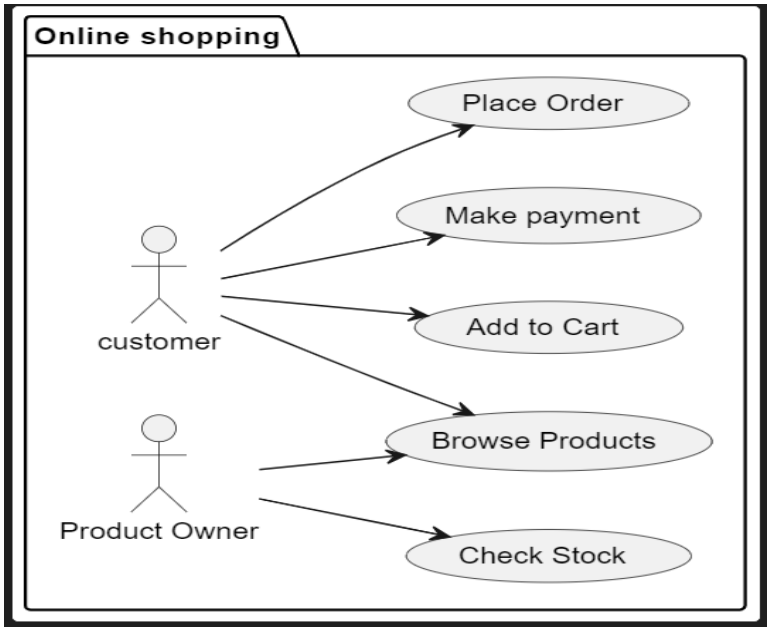
UML STRUCTURAL DIAGRAMS:

Class Diagram – The class diagram is the most extensively used UML diagram. It serves as the foundation for all object-oriented software systems. Class diagrams are used to depict a system's static structure by displaying its classes, methods, and properties. Class diagrams also assist us in determining the links among various classes or objects.

```
classuml.plantuml > {} classuml
@startuml
title Online shopping system
class "customer" as class1
class "product" as class2
class "Order" as class3
class "Cart" as class4
class "Payment" as class5
class "deliver" as class6
class "feedback" as class7
class "Return" as class8
class class1{

    String Customer_Name

    String Customer_ID

    String Address

    String product

    String Email

    Integer Phone_Number

    void order()

    void search()

    void rating()
```

Use Case Diagram are used to describe the functionality of a system or a component of a system.

They're frequently utilized to depict a system's functional requirements and interactions with external agents (actors). A use case is a diagram that depicts the various contexts in which a system can be applied. Use case diagrams provide us a high-level overview of what a system or element of a system performs without diving into the nitty-gritty of implementation.



usecase.plantuml > {} usecase

```
@startuml
left to right direction
package "Online shopping" {
actor "customer" as pm
  actor "Product Owner" as po
  usecase "Make payment" as UC1
  usecase "Place Order" as UC2
  usecase "Add to Cart" as UC3
  usecase "Browse Products" as UC4
  usecase "Check Stock" as UC5
}
pm --> UC1
pm --> UC2
pm --> UC3
pm --> UC4
po --> UC5
po --> UC4
@enduml
```

Sequence Diagram simply depicts the interactions between items in a series, i.e. the order in which these interactions take place. A sequence diagram can also be referred to as an event diagram or an event scenario. Sequence diagrams show how and in what order the components of a system work together.

Businesspeople and software engineers frequently use these diagrams to document and understand the needs of new and current systems.

Welcome | sequential.plantuml 1 | usecase.plantuml 1

sequential.plantuml > { } sequential

```
@startuml
actor Customer
participant "Order System" as OrderSystem
participant "Stock" as Stock

Customer -> OrderSystem: 1.Create Order
OrderSystem --> Customer: 2. Return if not clear
Customer -> OrderSystem: 3.[For each order] * add ite
OrderSystem -> Stock: 4.Check availability[product, q
Stock --> OrderSystem: 5.Return done
OrderSystem --> Customer: Confirm Order
Customer -> OrderSystem: Payment initiated
OrderSystem -> OrderSystem: Authentication Process
OrderSystem --> Customer: Order payment completed and

@enduml
```