Introduction to UML

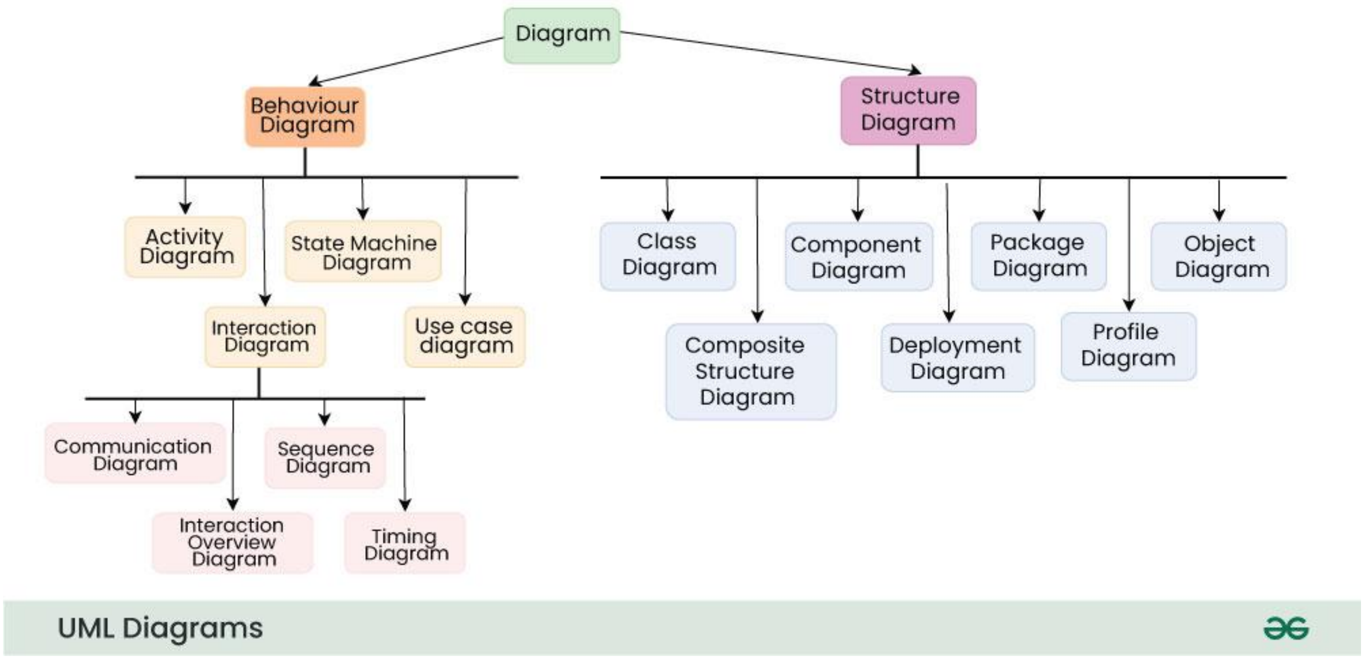
What is UML?

- Unified Modeling Language (UML) is a visual general purpose modeling language for defining systems. It is not a programming language nor a process but instead a set of tools for expressing thoughts related to software design. It is Created for anyone who is interested to understand the system no matter what the person's job or position is! Such as developers, users and business stakeholders.
- Developed by "The 3 Amigos": Grady Booch, Ivar Jacobson, and James Rumbaugh.
- It is endorsed by the Object Management Group (OMG).

UML Diagrams

- These are a set of standard and easy to understand graphical representations used to portray the behavior and structure of the system.
- Some UML diagrams can be forward engineered to actual code, saving time and reducing errors. Additionally, some tools allow reverse engineering of source code.
- There are two broad categories of diagrams
    - Behavioral (Dynamic): Show how the different parts of the system interact with each other
        1. **Use case diagram**: This is a diagram representing the different scenarios where the system can be used. It gives us a high level view of what the system or a part of the system does without going into implementation details.
        2. **Sequence Diagram**: describe how and in what order the objects in a system function. It is used to visualize the sequence of calls in a system to perform a specific functionality.
        3. **Communication Diagram / Collaboration Diagram**: It's somehow like the Sequence Diagram, but its goal is to represent the structural organization of a system and the messages sent/received.
        4. **State Machine 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.
        5. **Activity Diagram**: These are used to depict the workflows of concurrent and sequential Activities. Focuses on flow condition and the sequence in which it happens and also what causes a particular event to happen.
        6. **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.
        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.
        8. **Interaction Diagrams**: describe how groups of objects collaborate in some behavior. Two main types are Sequence Diagrams and Collaboration Diagrams.
    - Structural (Static): Show how the different parts of the system relate to each other and with their environment.

1. **Class diagram**: It's the most common diagram and represent the object oriented view of a system.
2. **Object Diagram**: It's an instance of class diagram. The usage of object diagrams is similar to class diagrams but they are used to build prototype of a system from practical perspective.
3. **Component Diagram**: Represents represent how the physical components in a system have been organized. We use them for modelling implementation details.
4. **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.
5. **Package Diagram**: depict how packages and their elements have been organized. It shows the dependencies between different packages and internal composition of packages.
6. **Composite Structure Diagram:** - used to represent the internal structure of a class and its interaction points with other parts of the system. They are similar to class diagrams except they represent individual parts in detail as compared to the entire class.



Object-Oriented Concepts Used in UML Diagrams

1. **Class**
   - A class defines the blue print i.e. structure and functions of an object.
2. **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.
3. **Inheritance**
   - Inheritance is a mechanism by which child classes inherit the properties of their parent classes.

4. **Abstraction**
    - Abstraction in UML refers to the process of emphasizing the essential aspects of a system or object while disregarding irrelevant details. By abstracting away unnecessary complexities, abstraction facilitates a clearer understanding and communication among stakeholders.
5. **capsulation**
    - Binding data together and protecting it from the outer world is referred to as encapsulation.
6. **Polymorphism**
    - Mechanism by which functions or entities are able to exist in different forms.

Actors:

- **Definition:** An actor in UML represents an external entity (either a person, another system, or a hardware device) that interacts with the system being modeled. Actors are typically individuals or entities that play a role in a specific use case.
- **Representation:** In UML diagrams, actors are usually depicted as stick figures or simple shapes. They are connected to use cases to indicate their involvement in the system's functionalities.
- **Example:** In a banking system, actors could include "Customer," "Bank Teller," and "ATM." Each of these entities interacts with the system in different ways.

Use Cases:

- **Definition:** A use case represents a specific functionality or behavior of a system from the perspective of an external actor. It describes a sequence of actions or interactions between the system and an actor to achieve a specific goal.
- **Representation:** Use cases are often represented as ellipses in UML diagrams. They are connected to actors through lines to show the association between actors and the functionalities they trigger.
- **Example:** In the same banking system, a use case could be "Withdraw Cash." This use case would describe the steps involved when a customer interacts with an ATM to withdraw money.

Unified Modeling Language (UML) Phases with the activities:

1. **Requirements Analysis Phase:**
    - Define the use case model.
    - Create use case diagrams.
    - Develop use case scenarios.
2. **Design Phase:**
    - Create UML diagrams.
    - Develop class diagrams.
    - Draw statechart diagrams.
3. **Implementation/Documentation Phase:**
    - Modify the UML diagrams.

- Develop and document the system.

Relationships between Classes

There are four kinds of relationships between classes: Association, Aggregation, Composition, and Generalization (inheritance). Relationships may include constructs such as stereotypes, role names, multiplicity, constraints, and navigability.

1. **Association:**
    - **Definition:** Describes a bi-directional connection between two classes.
    - **Representation:** Represented by a solid line connecting the classes involved.
    - **Attributes:** Can include attributes like stereotypes, role names, multiplicity, and navigability to provide additional details about the association.
2. **Aggregation:**
    - **Definition:** Represents a "whole-part" relationship where one class is a part of another class.
    - **Representation:** Shown by a hollow diamond at the containing class end.
    - **Attributes:** Similar to association, it can include stereotypes, role names, multiplicity, and navigability.
3. **Composition:**
    - **Definition:** Similar to aggregation but with a stronger relationship, implying a strong ownership where the part cannot exist independently of the whole.
    - **Representation:** Indicated by a filled diamond at the containing class end.
    - **Attributes:** Like association and aggregation, it can include additional attributes.
4. **Generalization (Inheritance):**
    - **Definition:** Represents an "is-a" relationship where one class is a specialized version of another (subclass and superclass).
    - **Representation:** Shown by an arrowhead pointing from the subclass to the superclass.
    - **Attributes:** Can include stereotypes, constraints, and other details.

Relationship Constructs

1. **Stereotypes:**
    - **Definition:** Stereotypes are custom tags or labels applied to elements in UML to extend their semantics. They help convey additional information about the nature or role of a specific element.
    - **Example:** If you have a generalization relationship between a base class "Vehicle" and a derived class "Car," you might use a stereotype like «is-a» to emphasize that it represents an inheritance relationship.
2. **Role Names:**
    - **Definition:** Role names are names assigned to the participating ends of an association to indicate the role that a class plays in the relationship.
    - **Example:** In a binary association between classes "Person" and "Address," you might have roles like "Owner" and "Residence" to indicate the roles each class plays.

3. **Multiplicity:**
   - **Definition:** Multiplicity defines the number of instances that can participate in a relationship. It specifies how many objects from one class are related to how many objects in another class.
   - **Example:** If a Person can have multiple Addresses, the multiplicity might be expressed as "1..*" (one or more).
4. **Constraints:**
   - **Definition:** Constraints, as previously explained, provide additional conditions or rules that must be satisfied for the relationship or its elements.
   - **Example:** A constraint might specify that the association between two classes is only valid if certain conditions are met.
5. **Navigability:**
   - **Definition:** Navigability indicates whether an object at one end of an association can access the objects at the other end directly. It shows the direction in which objects can be traversed.
   - **Example:** If a Person can navigate to their Address, it implies a navigability from Person to Address.

Comparison of some Relationships between Classes

1. **Association vs. Aggregation:**
   - **Association:** Represents a general connection or link between classes. It's a more generic relationship and doesn't imply any specific part-whole or ownership semantics.
   - **Aggregation:** Signifies a "whole-part" relationship, where one class (the whole) contains another class (the part). Aggregation is a weaker form of association and implies that the part can exist independently of the whole.
2. **Association vs. Composition:**
   - **Association:** Represents a simple link between classes without specifying any particular ownership or lifecycle dependencies. It's a more loosely coupled relationship.
   - **Composition:** Implies a stronger relationship where one class (the whole) is composed of another class (the part). The part cannot exist independently of the whole, indicating a stronger ownership and lifecycle dependency.
3. **Association vs. Generalization (Inheritance):**
   - **Association:** Represents a connection between classes without indicating any hierarchical structure. It's a way to relate classes in a more general sense.
   - **Generalization (Inheritance):** Represents a hierarchical relationship where a subclass inherits attributes and behaviors from a superclass. It signifies an "is-a" relationship and defines an "inheritance" hierarchy.
4. **Aggregation vs. Composition:**
   - **Aggregation:** Indicates a "whole-part" relationship, but the part can exist independently. It's a more loosely coupled relationship.
   - **Composition:** Represents a stronger form of "whole-part" relationship, where the part cannot exist independently of the whole. It implies ownership and a stronger lifecycle dependency.