Unified Modelling Language (UML)

Ruchita Shah

Introduction

- UML is a general-purpose, graphical modeling language in the field of Software Engineering
- Used to specify, visualize, construct, and document the artifacts (major elements) of the software system
- It was initially developed by Grady Booch, Ivar Jacobson, and James Rumbaugh in 1994-95 at Rational software, and its further development was carried out through 1996
- The Object Management Group (OMG) is an association of several companies that controls the open standard UML

Goals of UML

- Since it is a general-purpose modeling language, it can be utilized by all the modelers.
- UML came into existence after the introduction of object-oriented concepts to systemize and consolidate the object-oriented development, due to the absence of standard methods at that time.
- The UML diagrams are made for business users, developers, ordinary people, or anyone who is looking forward to understand the software or non-software system
- **Thus it can be concluded that the UML is a simple modeling approach that is used to model all the practical systems**

Characteristics of UML

- It is a generalized modeling language.
- It is distinct from other programming languages like C++, Python, etc.
- It is interrelated to object-oriented analysis and design.
- It is used to visualize the workflow of the system.
- It is a pictorial language, used to generate powerful modeling artifacts

Conceptual Modeling

- A conceptual model is composed of several interrelated concepts
- It makes it easy to understand the objects and how they interact with each other
- This is the first step before drawing UML diagrams.
- Following are some object-oriented concepts that are needed to begin with UML

OO Concepts

- Object: An object is a real world entity. There are many objects present within a single system. It is a fundamental building block of UML
- Class: A class is a software blueprint for objects, which means that it defines the variables and methods common to all the objects of a particular type
- Abstraction: Abstraction is the process of portraying the essential characteristics of an object to the users while hiding the irrelevant information. Basically, it is used to envision the functioning of an object

OO Concepts

- Inheritance: Inheritance is the process of deriving a new class from the existing ones
- Polymorphism: It is a mechanism of representing objects having multiple forms used for different purposes
- Encapsulation: It binds the data and the object together as a single unit, enabling tight coupling between them

OO Analysis and Design

- OO is an analysis of objects
- and design means combining those identified objects
- Main purpose of OO analysis is identifying the objects for designing a system
- The analysis can also be done for an existing system
- The analysis can be more efficient if we can identify the objects
- Once we have identified the objects, their relationships are then identified, and the design is also produced

OO Analysis and Design

- The purpose of OO is given below:
  - To identify the objects of a system
  - To identify their relationships
  - To make a design that is executable when the concepts of OO are employed

OO Analysis and Design

- **Step 1: OO Analysis**
- The main purpose of OO analysis is identifying the objects and describing them correctly
- After the objects are identified, the designing step is easily carried out
- It is a must to identify the objects with responsibilities
- Here the responsibility refers to the functions performed by the objects
- Each individual object has its own functions to perform
- The purpose of the system is fulfilled by collaborating these responsibilities

OO Analysis and Design

- **Step 2: OO Design**
- This phase mainly emphasizes on meeting the requirements
- In this phase, the objects are joined together as per the intended associations
- After the association is completed, the designing phase also gets complete.

OO Analysis and Design

- **Step 3: OO Implementation**
- This is the last phase that comes after the designing is done
- It implements the design using any OO languages like C++, Java, etc

Role of UML in OO design

- UML is a modeling language used to model software as well as non-software systems
- Here it focuses on modeling OO software applications
- It is essential to understand the relation between the OO design and UML
- The OO design can be converted into the UML as and when required
- The OO languages influence the programming world as they model real world objects

Role of UML in OO design

- The UML itself is an amalgamation of object-oriented notations like Object-Oriented Design (OOD), Object Modeling Technique (OMT), and Object-Oriented Software Engineering (OOSE)
- The strength of these three approaches is utilized by the UML to represent more consistency

UML-Building Blocks

- UML is composed of three main building blocks
  - Things
  - Relationships
  - Diagrams
- Building blocks generate one complete UML model diagram by rotating around several different blocks
- It plays an essential role in developing UML diagrams

Things

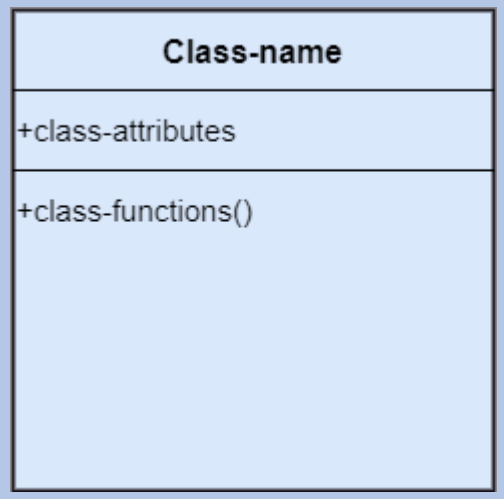
- Anything that is a real world entity or object is termed as things. It can be divided into several different categories:
  - Structural things
  - Behavioral things
  - Grouping things
  - Annotational things

Things-Structural things

- Nouns that depicts the static behavior of a model is termed as structural things
- They display the physical and conceptual components
- They include class, object, interface, node, collaboration, component, and a use case

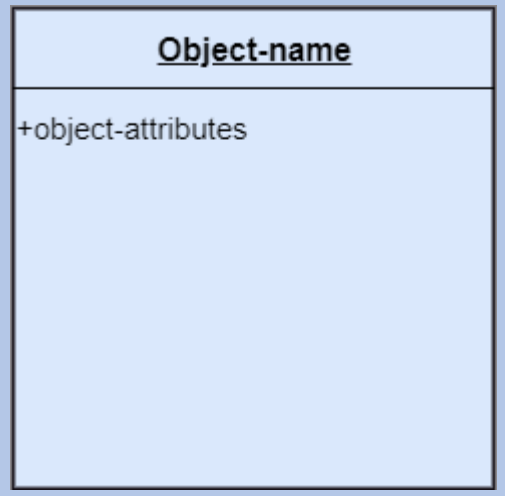
Things-Structural things

- Class : A Class is a set of identical things that outlines the functionality and properties of an object. It also represents the abstract class whose functionalities are not defined. Class notation is given below



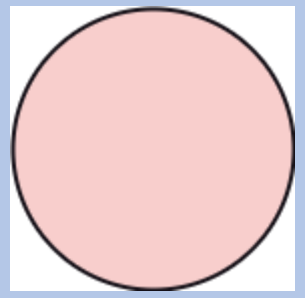
Things-Structural things

- Object: An individual that describes the behavior and the functions of a system. The notation of the object is similar to that of the class; the only difference is that the object name is always underlined and its notation is given below



Things-Structural things

- Interface: A set of operations that describes the functionality of a class, which is implemented whenever an interface is implemented

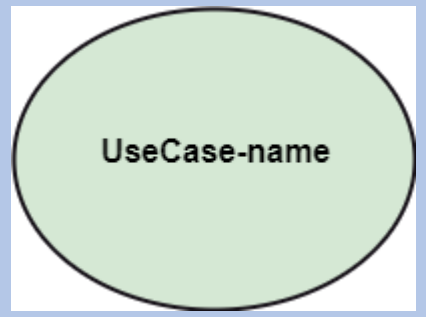


Things-Structural things

- Collaboration: It represents the interaction between things that is done to meet the goal. It is symbolized as a dotted ellipse with its name written inside it

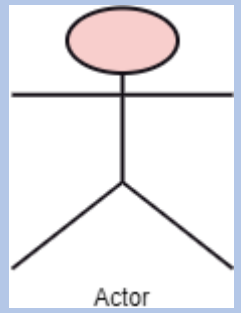
Things-Structural things

- Use case: Use case is the core concept of object-oriented modeling. It portrays a set of actions executed by a system to achieve the goal



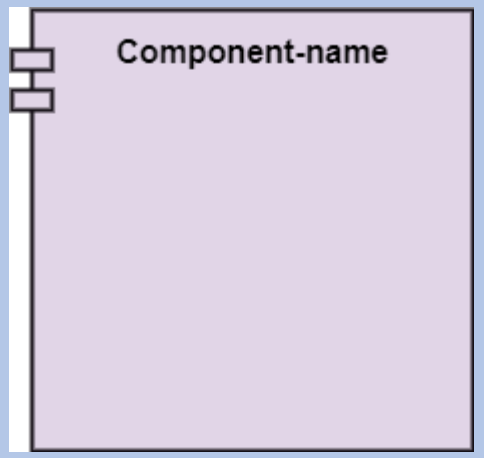
Things-Structural things

- Actor: It comes under the use case diagrams. It is an object that interacts with the system, for example, a user



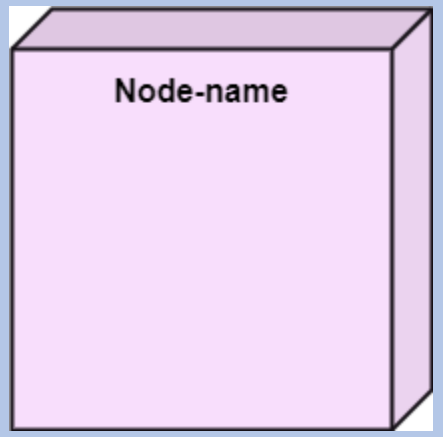
Things-Structural things

- Component: It represents the physical part of the system



Things-Structural things

- Node: A physical element that exists at run time



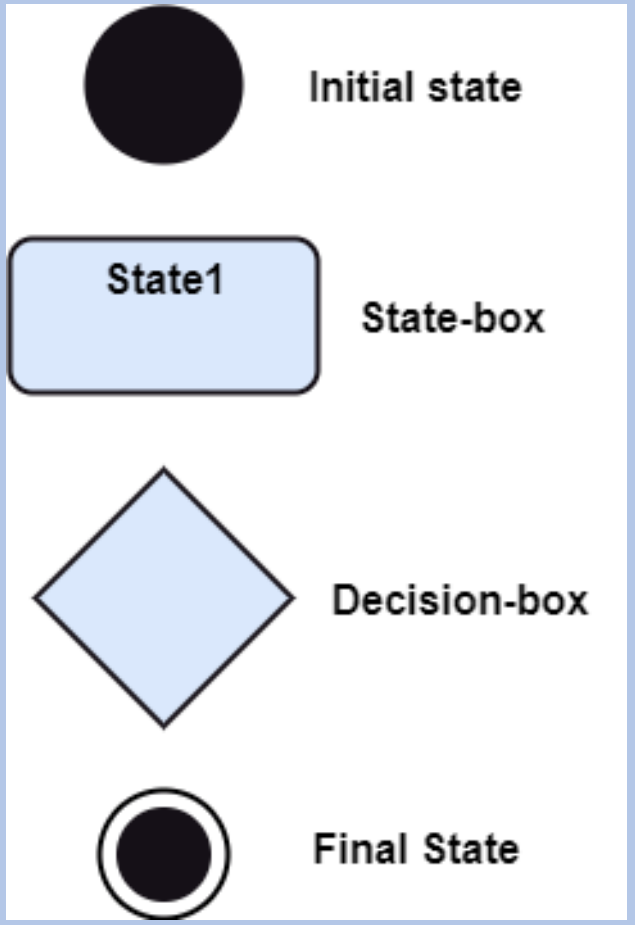
Things-Behavioral Things

- They are the verbs that encompass the dynamic parts of a model
- It depicts the behavior of a system
- They involve state machine, activity diagram, interaction diagram, grouping things, annotation things

Things-Behavioral Things

State Machine

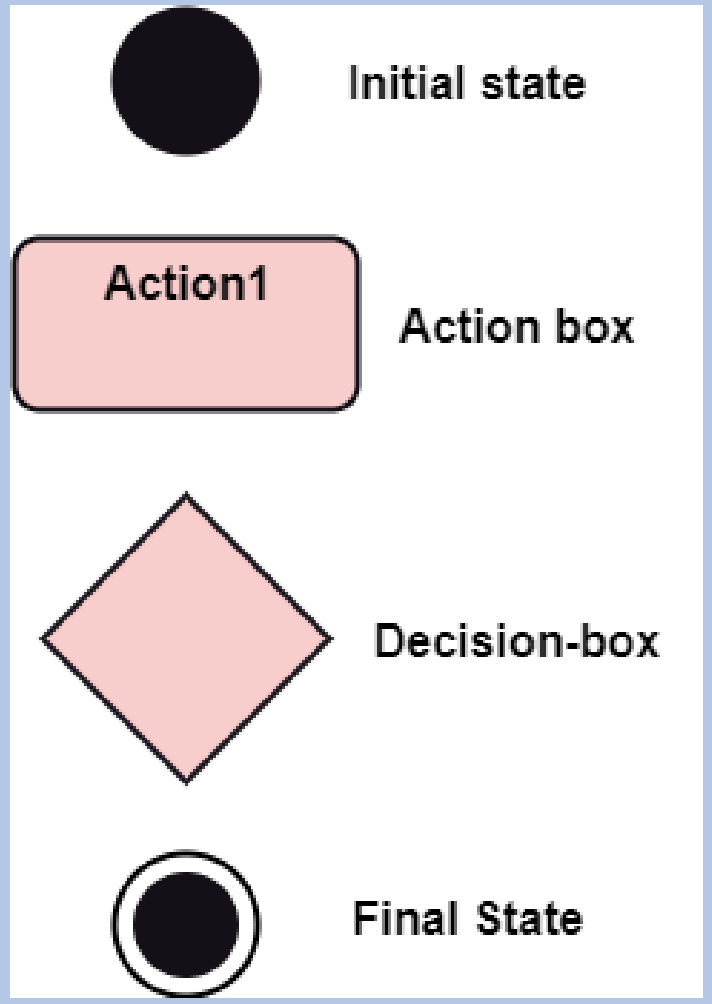
- It defines a sequence of states that an entity goes through in the software development lifecycle
- It keeps a record of several distinct states of a system component.



Things-Behavioral Things

Activity Diagram

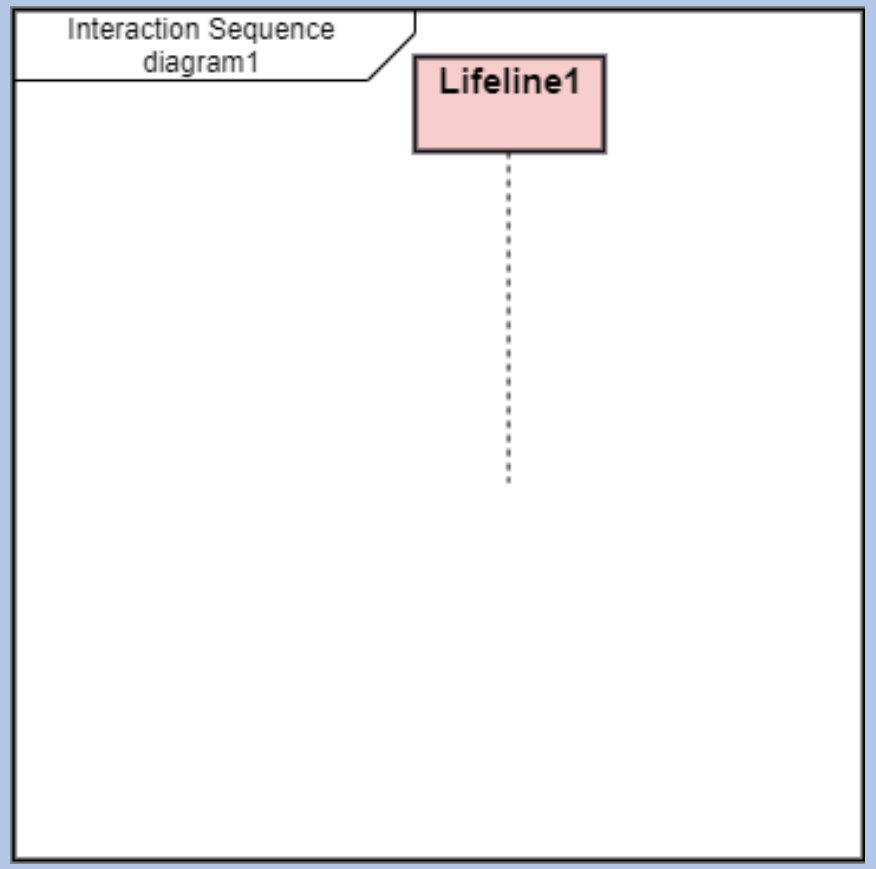
- It portrays all the activities accomplished by different entities of a system
- It is represented the same as that of a state machine diagram
- It consists of an initial state, final state, a decision box, and an action notation



Things-Behavioral Things

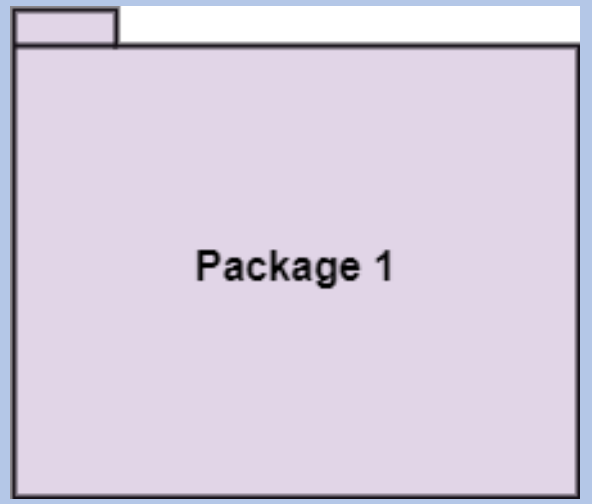
Interaction Diagram

- It is used to envision the flow of messages between several components in a system



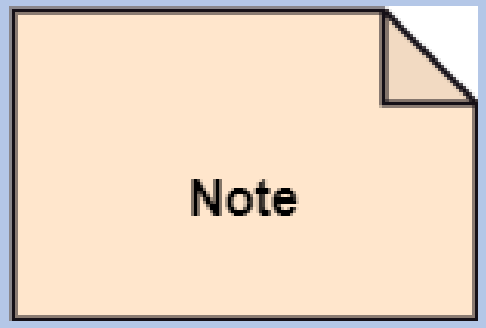
Things- Grouping Things

- It is a method that together binds the elements of the UML model
- In UML, the package is the only thing which is used for grouping of behavioral and structural things



Things- Annotation Things

- It is a mechanism that captures the remarks, descriptions, and comments of UML model elements
- In UML, a note is the only Annotational thing
- It is used to attach the constraints, comments, and rules to the elements of the model
- It is a kind of yellow sticky note.



Relationships

- It illustrates the meaningful connections between things
- It shows the association between the entities and defines the functionality of an application
- There are four types of relationships
  - Dependency
  - Association
  - Generalization
  - Realization

Relationships - Dependency

- A kind of relationship in which a change in target element affects the source element, or simply we can say the source element is dependent on the target element
- It is one of the most important notations in UML
- It depicts the dependency from one entity to another.
- It is denoted by a dotted line followed by an arrow at one side as shown below

Relationships - Association

- A set of links that associates the entities to the UML model
- It tells how many elements are actually taking part in forming that relationship
- It is denoted by a dotted line with arrowheads on both sides to describe the relationship with the element on both sides

Relationships - Generalization

- It portrays the relationship between a general thing (a parent class or superclass) and a specific kind of that thing (a child class or subclass)
- It is used to describe the concept of inheritance
- It is denoted by a straight line followed by an empty arrowhead at one side

Relationships - Realization

- It is a semantic kind of relationship between two things, where one defines the behavior to be carried out, and the other one implements the mentioned behavior
- It exists in interfaces
- It is denoted by a dotted line with an empty arrowhead at one side

Diagrams

- Graphical implementation of the models that incorporate symbols and text
- Each symbol has a different meaning in the context of the UML diagram
- There are thirteen different types of UML diagrams that are available in UML 2.0
- Each diagram has its own set of a symbol
- And each diagram manifests a different dimension, perspective, and view of the system

Diagrams

- UML diagrams are classified into three categories
  - Structural Diagram
  - Behavioral Diagram
  - Interaction Diagram

Diagrams - Structural Diagram

- Represents the static view of a system by portraying the structure of a system
- It shows several objects residing in the system
- Structural diagrams are:
  - Class diagram
  - Object diagram
  - Package diagram
  - Component diagram
  - Deployment diagram

Diagrams - Behavioral Diagram

- Depicts the behavioral features of a system
- Deals with dynamic parts of the system
- They are:
  - Activity diagram
  - State machine diagram
  - Use case diagram

Diagrams - Interaction diagram

- A subset of behavioral diagrams
- Depicts the interaction between two objects and the data flow between them
- They are :
  - Timing diagram
  - Sequence diagram
  - Collaboration diagram

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

UML- Architecture

- 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

UML- Architecture

- 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?

UML- Architecture

- 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

UML- Architecture

- 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

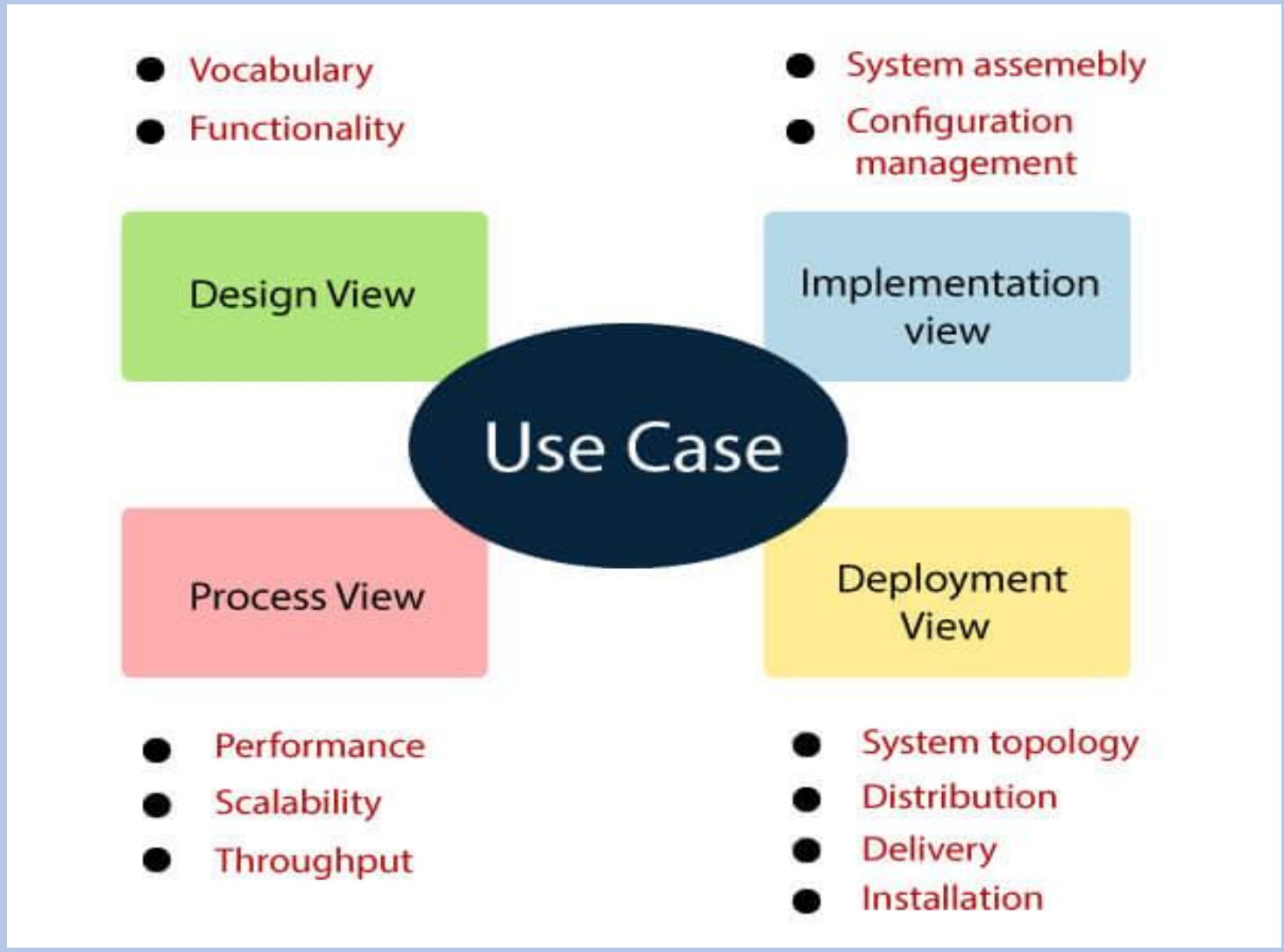
UML- Architecture

- 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

UML- Architecture

- Also, it provides a collection of several views. It can be best understood as a collection of five views:
  - Use case view
  - Design view
  - Implementation view
  - Process view
  - Development view

UML- Architecture



UML- Architecture

Use case view

- It is a view that shows the functionality of the system as perceived by external actors
- It reveals the requirements of the system
- With UML, it is easy to capture the static aspects of this view in the use case diagrams, whereas its dynamic aspects are captured in interaction diagrams, state chart diagrams, and activity diagrams

UML- Architecture

Design view

- It is a view that shows how the functionality is designed inside the system in terms of static structure and dynamic behavior
- It captures the vocabulary of the problem space and solution space
- 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

UML- Architecture

Implementation view

- It is the view that represents the organization of the core components and files
- It primarily addresses the configuration management of the system's releases
- 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

UML- Architecture

Process view

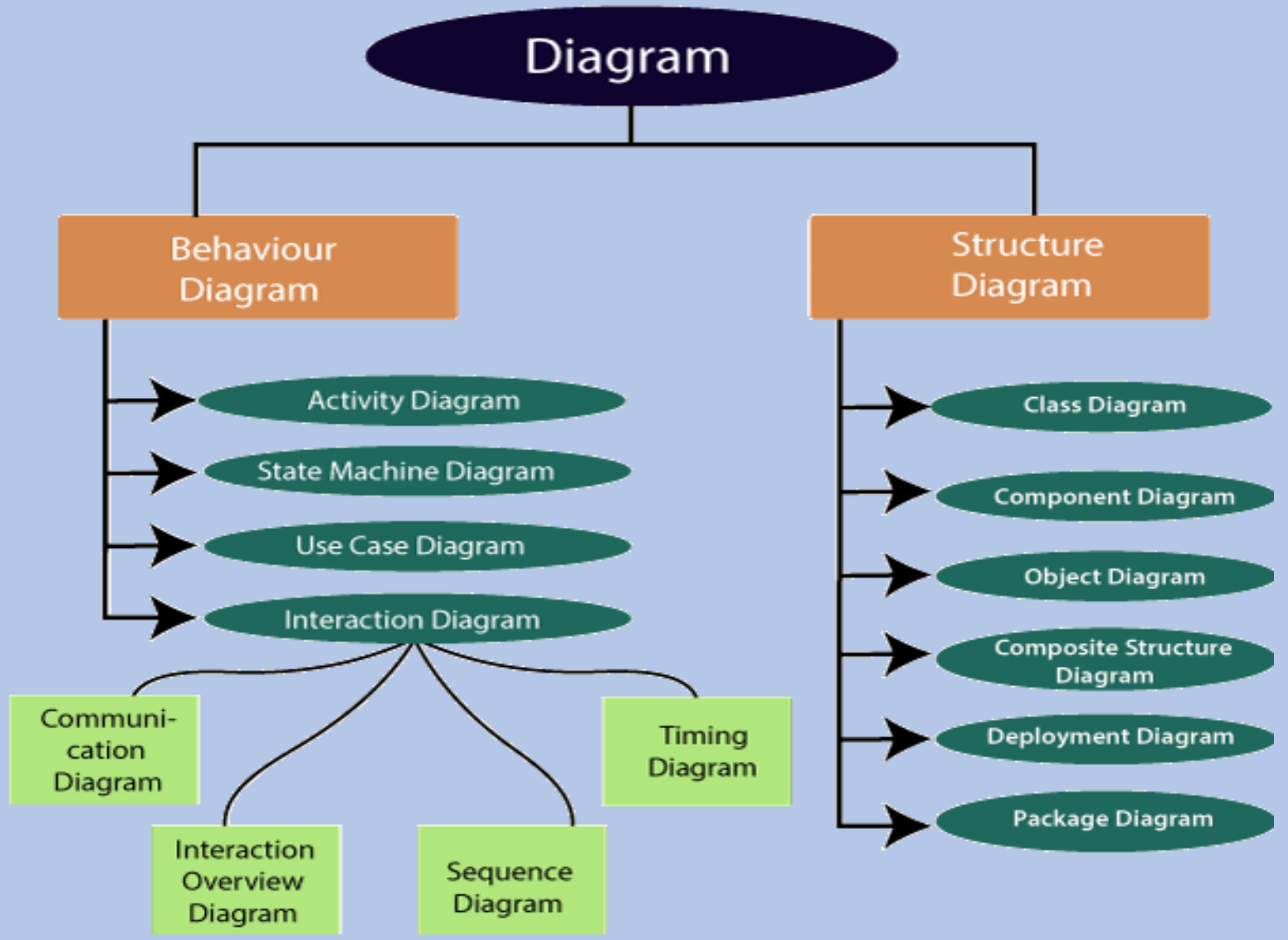
- It is the view that demonstrates the concurrency of the system
- It incorporates the threads and processes that make concurrent system and synchronized mechanisms
- It primarily addresses the system's scalability, throughput, and performance
- Its static and dynamic aspects are expressed the same way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

UML- Architecture

Deployment view

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

UML-Diagrams



Structural Diagrams

- Depict a static view or structure of a system
- It is widely used in the documentation of software architecture
- It embraces class diagrams, composite structure diagrams, component diagrams, deployment diagrams, object diagrams, and package diagrams
- Presents an outline for the system
- Stresses the elements to be present that are to be modeled

Structural Diagrams

Class Diagram

- Class diagrams are one of the most widely used diagrams
- It is the backbone of all the object-oriented software systems
- It depicts the static structure of the system
- It displays the system's class, attributes, and methods
- It is helpful in recognizing the relation between different objects as well as classes

Structural Diagrams

Composite Structure Diagram

- The composite structure diagrams show parts within the class
- It displays the relationship between the parts and their configuration that ascertain the behavior of the class
- It makes full use of ports, parts, and connectors to portray the internal structure of a structured classifier
- It is similar to class diagrams, just the fact it represents individual parts in a detailed manner when compared with class diagrams

Structural Diagrams

Object Diagram

- Describes the static structure of a system at a particular point in time
- It can be used to test the accuracy of class diagrams
- It represents distinct instances of classes and the relationship between them at a time

Structural Diagrams

Component Diagram

- Portrays the organization of the physical components within the system
- It is used for modeling execution details
- It determines whether the desired functional requirements have been considered by the planned development or not, as it depicts the structural relationships between the elements of a software system

Structural Diagrams

Deployment Diagram

- Presents the system's software and its hardware by telling what the existing physical components are and what software components are running on them
- It produces information about system software
- It is incorporated whenever software is used, distributed, or deployed across multiple machines with dissimilar configurations

Structural Diagrams

Package Diagram

- Used to illustrate how the packages and their elements are organized
- It shows the dependencies between distinct packages
- It manages UML diagrams by making it easily understandable
- It is used for organizing the class and use case diagrams

Behavioral Diagrams

- Behavioral diagrams portray a dynamic view of a system or the behavior of a system, which describes the functioning of the system
- It includes use case diagrams, state diagrams, and activity diagrams
- It defines the interaction within the system

Behavioral Diagrams

State Machine Diagram

- It is a behavioral diagram
- It portrays the system's behavior utilizing finite state transitions
- It is also known as the State-charts diagram
- It models the dynamic behavior of a class in response to external stimuli

Behavioral Diagrams

Activity Diagram

- It models the flow of control from one activity to the other
- With the help of an activity diagram, we can model sequential and concurrent activities
- It visually depicts the workflow as well as what causes an event to occur

Behavioral Diagrams

Use Case Diagram

- It represents the functionality of a system by utilizing actors and use cases
- It encapsulates the functional requirement of a system and its association with actors
- It portrays the use case view of a system

Interaction Diagrams

- A subclass of behavioral diagrams that give emphasis to object interactions and also depicts the flow between various use case elements of a system
- In simple words, it shows how objects interact with each other and how the data flows within them
- It consists of communication, interaction overview, sequence, and timing diagrams

Interaction Diagrams

Sequence Diagram

- It shows the interactions between the objects in terms of messages exchanged over time
- It delineates in what order and how the object functions are in a system

Interaction Diagrams

Communication Diagram

- It shows the interchange of sequence messages between the objects
- It focuses on objects and their relations
- It describes the static and dynamic behavior of a system

Interaction Diagrams

Timing Diagram

- A special kind of sequence diagram used to depict the object's behavior over a specific period of time
- It governs the change in state and object behavior by showing the time and duration constraints

Interaction Diagrams

Interaction Overview Diagram

- It is a mixture of activity and sequence diagram that depicts a sequence of actions to simplify the complex interactions into simple interactions

UML-Relationship

- Depict a connection between several things, such as structural, behavioral, or grouping things in the unified modeling language
- Since it is termed as a link, it demonstrates how things are interrelated to each other at the time of system execution
- It constitutes four types of relationships, i.e., dependency, association, generalization, and realization

UML-Relationship

Dependency

- Whenever there is a change in either the structure or the behavior of the class that affects the other class, such a relationship is termed as a dependency
- Simply, we can say a class contained in other class is known as dependency
- It is a unidirectional relationship

UML-Relationship

Association

- A structural relationship that represents how two entities are linked or connected to each other within a system
- It can form several types of associations, such as one-to-one, one-to-many, many-to-one, and many-to-many
- A ternary association is one that constitutes three links
- It portrays the static relationship between the entities of two classes.
- Four types of associations - bi-directional, unidirectional, aggregation (composition aggregation), and reflexive,
- Aggregation is a special form of association and composition is a special form of aggregation
- Mostly associations are unidirectional and bi-directional

UML-Relationship

Association-Aggregation

- A special form of association
- It portrays a part-of relationship
- It forms a binary relationship, which means it cannot include more than two classes
- It is also known as Has-a relationship
- It specifies the direction of an object contained in another object
- In aggregation, a child can exist independent of the parent

UML-Relationship

Association-Composition

- the child depends on the parent
- It forms a two-way relationship
- It is a special case of aggregation
- It is known as Part-of relationship

UML-Relationship

Aggregation VS Composition relationship

Features	Aggregation relationship	Composition relationship
Dependency	In an aggregation relationship, a child can exist independent of a parent.	In a composition relationship, the child cannot exist independent of the parent.
Type of Relationship	It constitutes a **Has-a** relationship.	It constitutes **Part-of**relationship.
Type of Association	It forms a **weak** association.	It forms a **strong** association.
Examples	A doctor has patients when the doctor gets transfer to another hospital, the patients do not accompany to a new workplace.	A hospital and its wards. If the hospital is destroyed, the wards also get destroyed.

UML-Relationship

Generalization

- The generalization relationship implements the object-oriented concept called inheritance or is-a relationship
- It exists between two objects (things or entities), such that one entity is a parent (superclass or base class), and the other one is a child (subclass or derived class)
- These are represented in terms of inheritance
- Any child can access, update, or inherit the functionality, structure, and behavior of the parent

UML-Relationship

Realization

- A kind of relationship in which one thing specifies the behavior or a responsibility to be carried out, and the other thing carries out that behavior
- It can be represented on a class diagram or component diagrams
- The realization relationship is constituted between interfaces, classes, packages, and components to link a client element to the supplier element

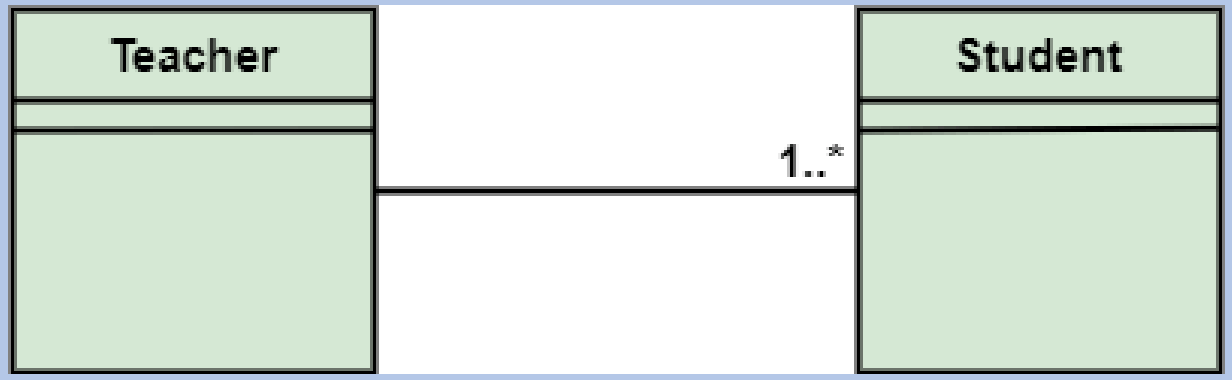
UML Association vs. Aggregation vs. Composition

Association

- A structural relationship in which different objects are linked within the system
- It exhibits a binary relationship between the objects representing an activity
- It depicts the relationship between objects, such as a teacher, can be associated with multiple teachers.
- It is represented by a line between the classes followed by an arrow that navigates the direction, and when the arrow is on both sides, it is then called a bidirectional association
- We can specify the multiplicity of an association by adding the adornments on the line that will denote the association

UML Association vs. Aggregation vs. Composition

A single teacher has multiple students



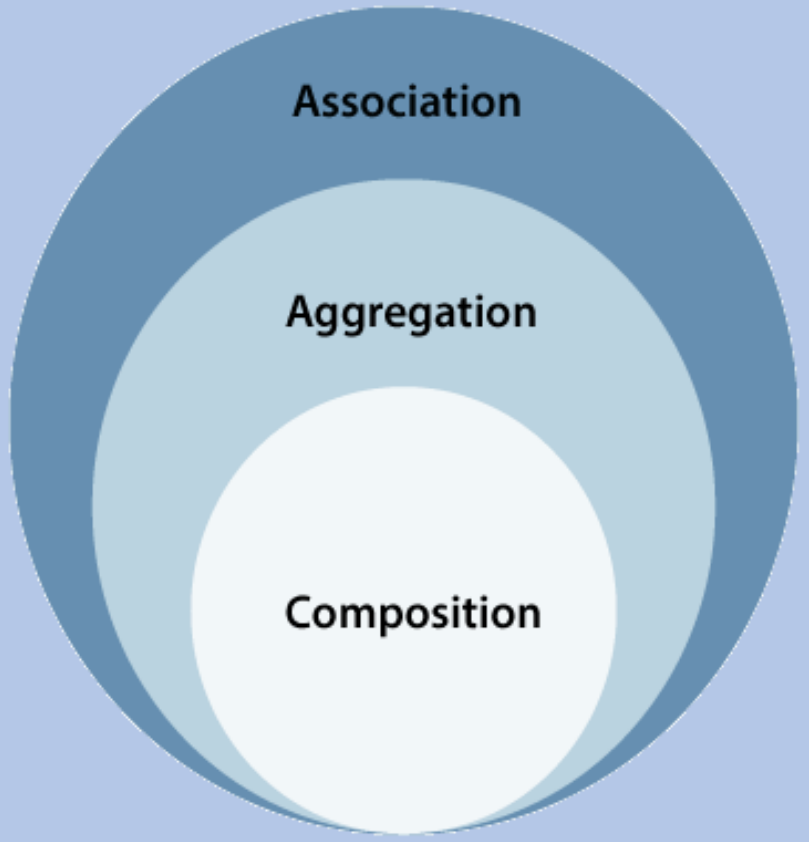
UML Association vs. Aggregation vs. Composition

A single student can associate with many teachers

UML Association vs. Aggregation vs. Composition

- Composition and aggregation are two subsets of association
- In both of the cases, the object of one class is owned by the object of another class
- The only difference is that in composition, the child does not exist independently of its parent, whereas in aggregation, the child is not dependent on its parent i.e., standalone
- An aggregation is a special form of association, and composition is the special form of aggregation

UML Association vs. Aggregation vs. Composition



UML Association vs. Aggregation vs. Composition

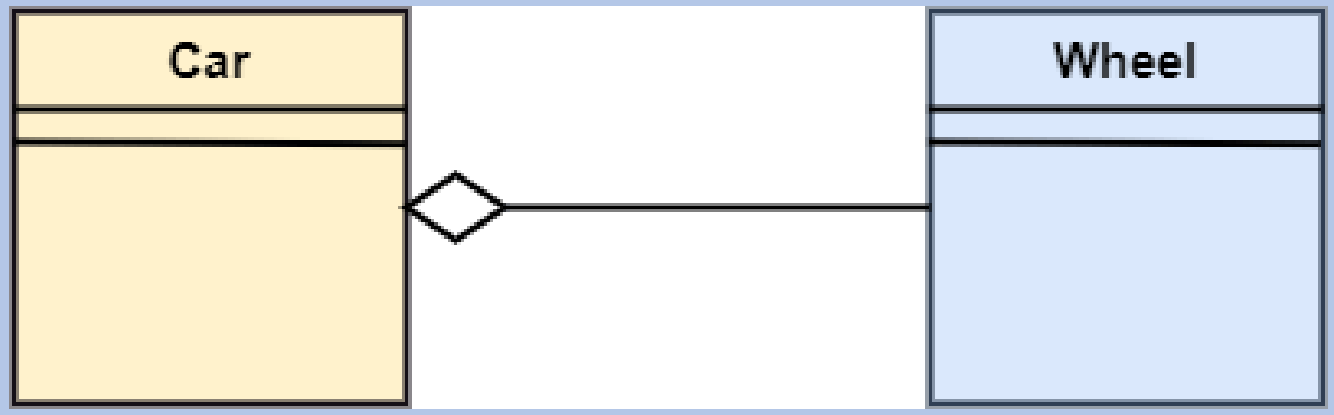
Aggregation

- A subset of association, is a collection of different things
- It represents “has a” relationship
- It is more specific than an association
- It describes a part-whole or part-of relationship
- It is a binary association, i.e., it only involves two classes
- It is a kind of relationship in which the child is independent of its parent

UML Association vs. Aggregation vs. Composition

Aggregation

- Ex. : Consider a car and a wheel example
- A car cannot move without a wheel. But the wheel can be independently used with the bike, scooter, cycle, or any other vehicle. The wheel object can exist without the car object, which proves to be an aggregation relationship



UML Association vs. Aggregation vs. Composition

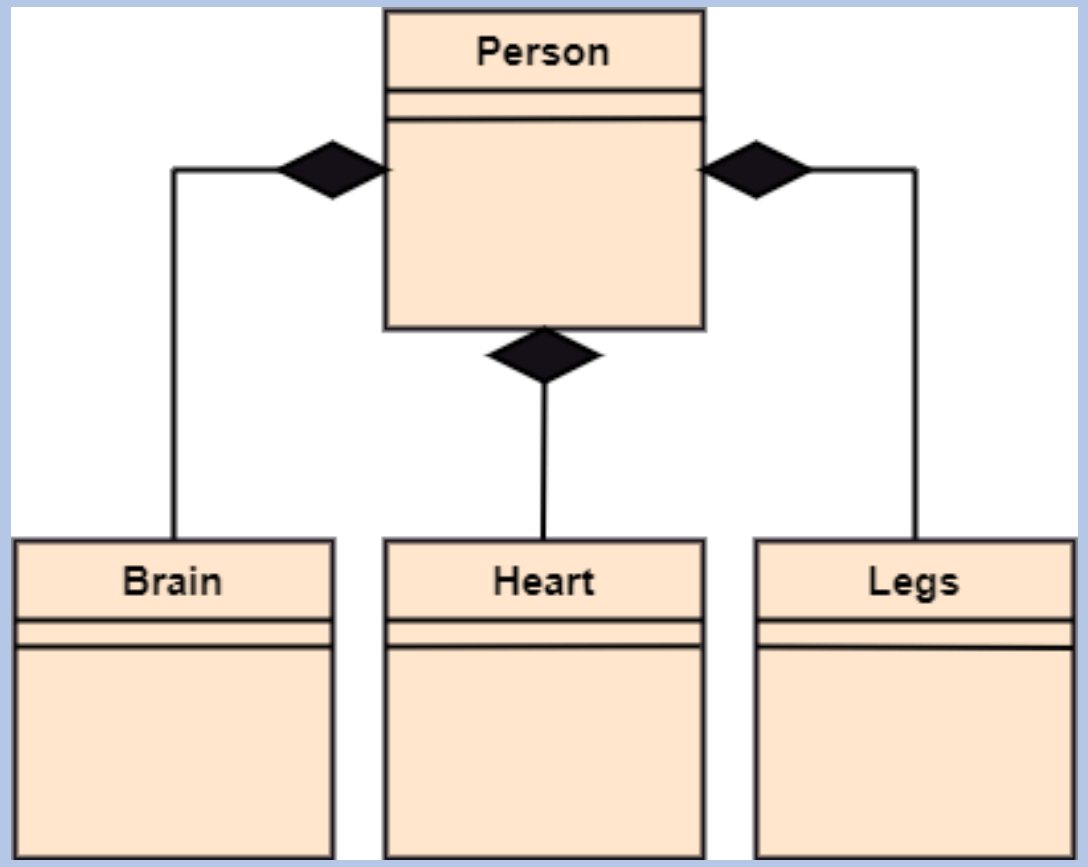
Composition

- Is a part of aggregation, and it portrays the whole-part relationship
- It depicts dependency between a composite (parent) and its parts (children), which means that if the composite is discarded, so will its parts get deleted
- It exists between similar objects

UML Association vs. Aggregation vs. Composition

Composition

- The composition association relationship connects the Person class with Brain class, Heart class, and Legs class. If the person is destroyed, the brain, heart, and legs will also get discarded



UML Association vs. Aggregation vs. Composition

Association	Aggregation	Composition
Association relationship is represented using an arrow.	Aggregation relationship is represented by a straight line with an empty diamond at one end.	The composition relationship is represented by a straight line with a black diamond at one end.
In UML, it can exist between two or more classes.	It is a part of the association relationship.	It is a part of the aggregation relationship.
It incorporates one-to-one, one-to-many, many-to-one, and many-to-many association between the classes.	It exhibits a kind of weak relationship.	It exhibits a strong type of relationship.
It can associate one more objects together.	In an aggregation relationship, the associated objects exist independently within the scope of the system.	In a composition relationship, the associated objects cannot exist independently within the scope of the system.

UML Association vs. Aggregation vs. Composition

Association	Aggregation	Composition
In this, objects are linked together.	In this, the linked objects are independent of each other.	Here the linked objects are dependent on each other.
It may or may not affect the other associated element if one element is deleted.	Deleting one element in the aggregation relationship does not affect other associated elements.	It affects the other element if one of its associated element is deleted.
Example: A tutor can associate with multiple students, or one student can associate with multiple teachers.	Example: A car needs a wheel for its proper functioning, but it may not require the same wheel. It may function with another wheel as well.	Example: If a file is placed in a folder and that is folder is deleted. The file residing inside that folder will also get deleted at the time of folder deletion.

UML- Association

- Semantic relationship between classes that shows how one instance is connected or merged with others in a system
- The objects are combined either logically or physically
- Since it connects the object of one class to the object of another class, it is categorized as a structural relationship

UML- Association

- The constraints applied to the association relationship are enlisted below:
- {implicit}: (ગર્ભિત) As the name suggests, the implicit constraints define that the relationship is not visible, but it is based on a concept
- {ordered}: It describes that the set of entities is in a particular way at one end in an association
- {changeable}: The changeable constraint ensures that the connections between several objects within a system are added, improved, and detached, as and when required.

UML- Association

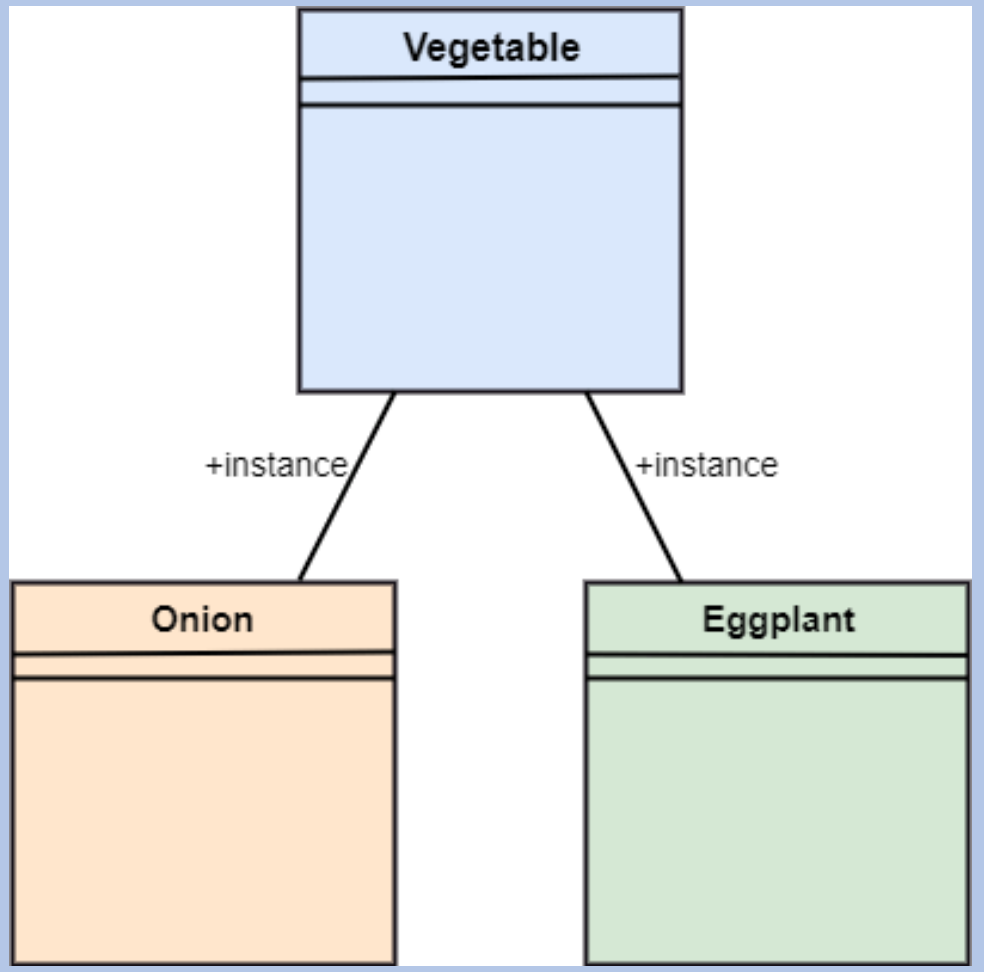
- {addOnly}: It specifies that any new connection can be added from an object located at the other end in an association.
- {frozen}: The frozen constraint specifies that whenever a link is added between objects, it cannot be altered by the time it is activated over the connection or given link.

UML- Association

- Reflexive Association : links are between the objects of the same classes. In other words, it can be said that the reflexive association consists of the same class at both ends. An object can also be termed as an instance

UML- Association

- Reflexive Association : The link between the onion and eggplant exist, as they belong to the same class



UML- Association

- Directed Association : Is concerned with the direction of flow inside association classes. The flow of association can be shown by employing a directed association. The directed association between two classes is represented by a line with an arrowhead, which indicates the navigation direction. The flow of association from one class to another is always in one direction.

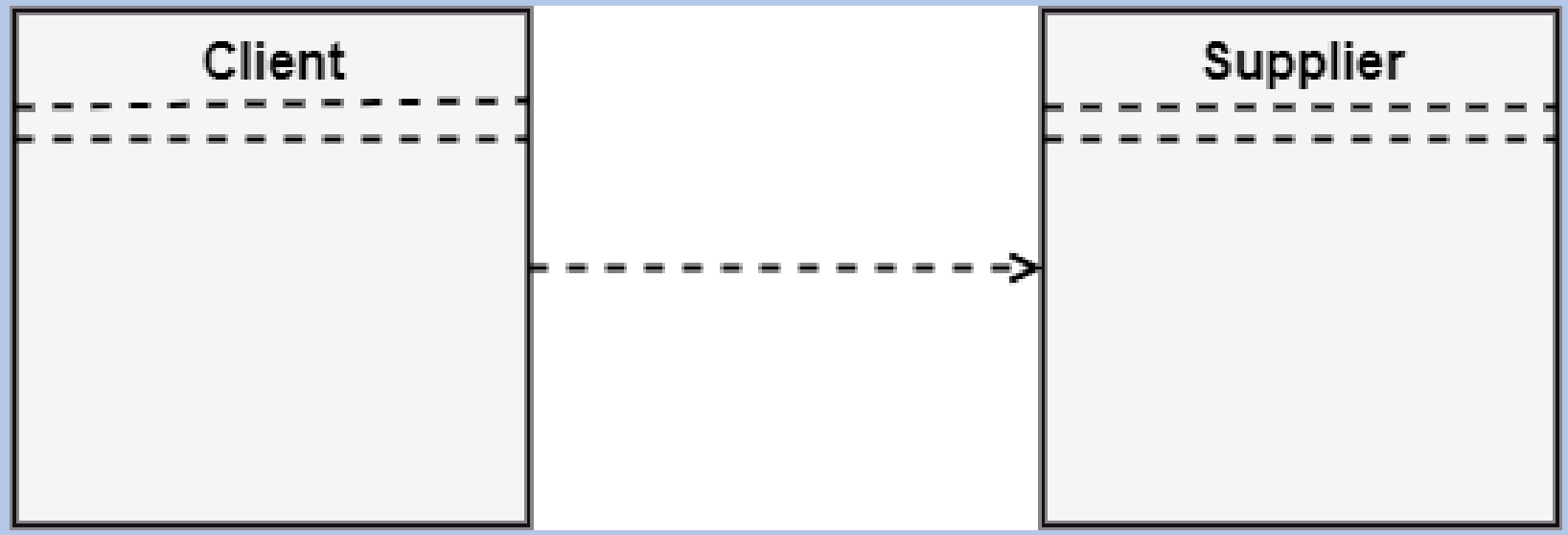
UML- Association

- Directed Association : Here the person works for the company, and not the company works for a person

UML-Dependency

- Dependency depicts how various things within a system are dependent on each other
- In UML, a dependency relationship is the kind of relationship in which a client (one element) is dependent on the supplier (another element)
- It is used in class diagrams, component diagrams, deployment diagrams, and use-case diagrams, which indicates that a change to the supplier necessitates a change to the client

UML-Dependency



UML-Dependency

- <<derive>> -It is a constraint that specifies the template can be initialized by the source at the target location utilizing given parameters.
- <<derive>> -It represents that the source object's location can be evaluated from the target object.
- <<friend>> -It states the uniqueness of the source in the target object.
- <<instanceOf>> -It states that an instance of a target classifier is the source object.
- <<instantiate>> -It defines the capability of the source object, creating instances of a target object.

UML-Dependency

- <<refine>> -It states that the source object comprises of exceptional abstraction than that of the target object.
- <<use>> -When the packages are created in UML, the use of stereotype is used as it describes that the elements of the source package can also exist in the target package. It specifies that the source package uses some of the elements of the target package.
- <<substitute>> -The substitute stereotype state that the client can be substituted at the runtime for the supplier.
- <<access>> -It is also called as private merging in which the source package accesses the element of the target package.
- <<import>> -It specifies that target imports the source package's element as they are defined within the target. It is also known as public merging.

UML-Dependency

- <<permit>> -It describes that the source element can access the supplier element or whatever visibility is provided by the supplier.
- <<extend>> -It states that the behavior of the source element can be extended by the target.
- <<include>> -It describes the source element, which can include the behavior of another element at a specific location, just like a function call in C/C++.
- <<become>> -It states that target is similar to the source with distinct roles and values.
- <<call>> -It specifies that the target object can be invoked by the source.
- <<copy>> -It states that the target is an independent replica of a source object.
- <<parameter>> -It describes that the supplier is a parameter of the client's actions.
- <<send>> -The client act as an operation, which sends some unspecified targets to the supplier.

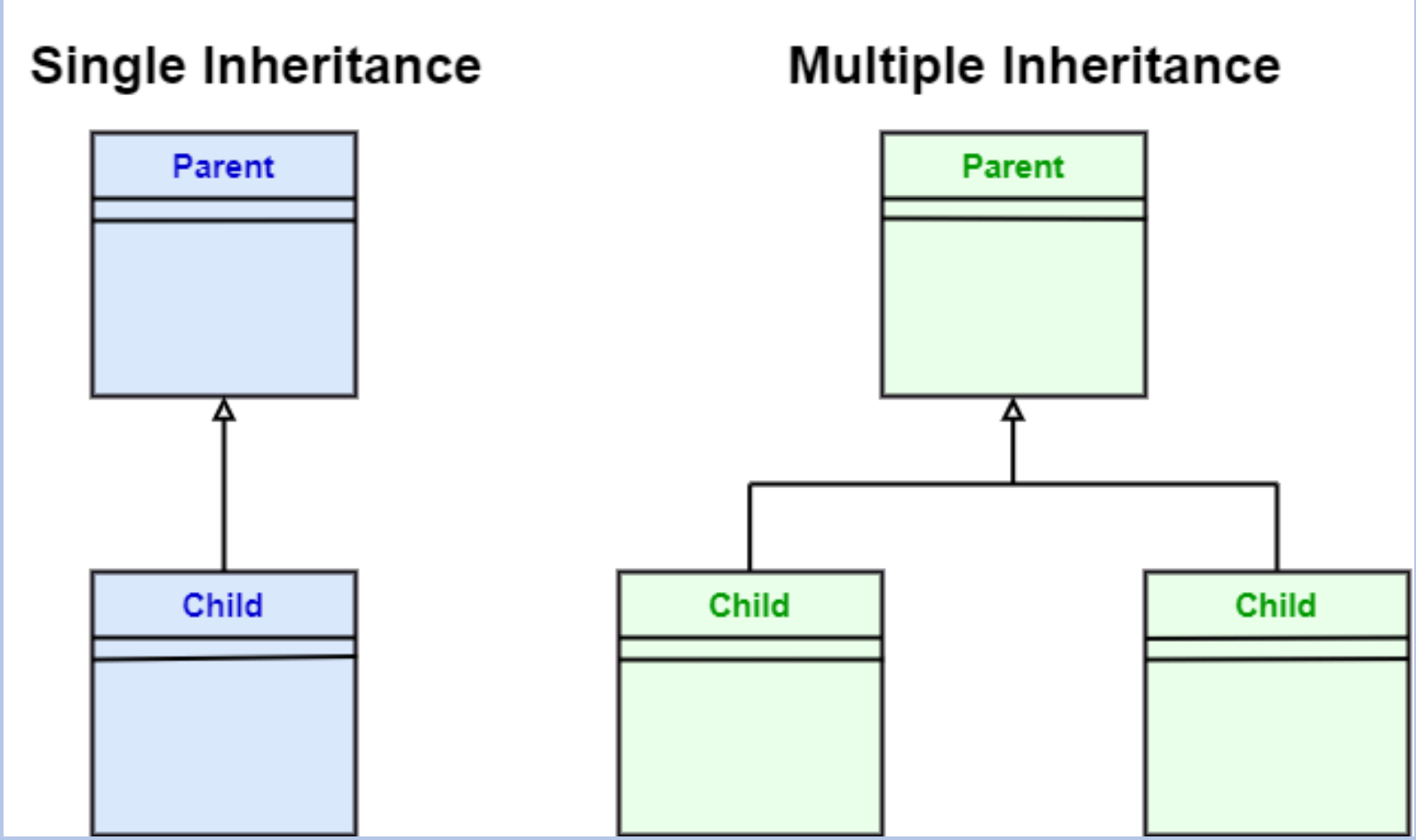
UML-Generalization

- A relationship that implements the concept of object orientation called inheritance
- The generalization relationship occurs between two entities or objects, such that one entity is the parent, and the other one is the child
- The child inherits the functionality of its parent and can access as well as update it.
- Generalization relationship is utilized in class, component, deployment, and use case diagrams to specify that the child inherits actions, characteristics, and relationships from its parent
- Generalization relation can either be used between actors or between use cases, but not between an actor and a use case

UML-Generalization

- The generalization relationship is incorporated to record attributes, operations, and relationships in a parent model element so that it can be inherited in one or more child model elements
- The parent model element can have as many children, and also, the child can have one or more parents
- But most commonly, it can be seen that there is one parent model element and multiple child model elements
- The generalization relationship does not consist of names
- It is represented by a solid line with a hollow arrowhead pointing towards the parent model element from the child model element

UML-Generalization



UML-Generalization

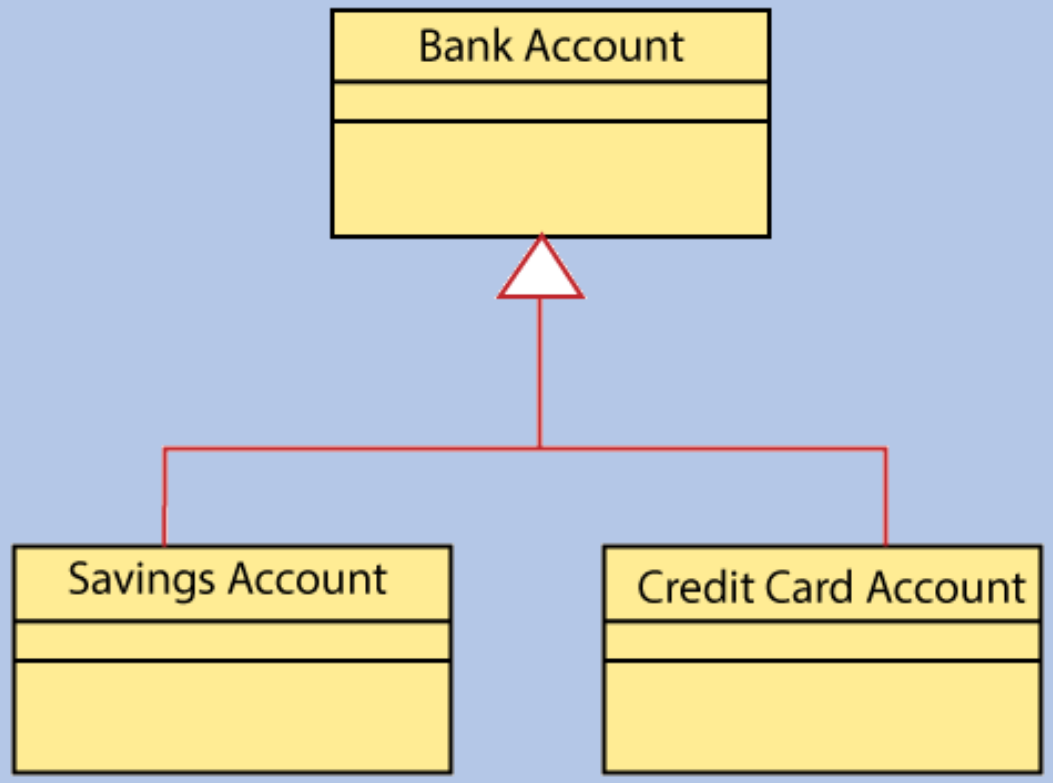
- Stereotypes and their constraints :
- <<implementation>> - It is used to show that the child is implemented by its parent, such that the child object inherits the structure and behavior of its parent object without disobeying the rules. The implementation of stereotype is mostly used in single inheritance.
- In the generalization stereotype, there are two types of constraints that are complete and incomplete to check if all the child objects are involved or not in the relationship

UML-Generalization

- Stereotypes and their constraints :
- Example: The bank account can be of two types; Savings Account and Credit Card Account. Both the savings and the credit card account inherits the generalized properties from the Bank Account, which is Account Number, Account Balance, etc

UML-Generalization

- Stereotypes and their constraints :



UML Class Diagram

- Class diagram depicts a static view of an application
- It represents the types of objects residing in the system and the relationships between them
- A class consists of its objects, and also it may inherit from other classes
- A class diagram is used to visualize, describe, document various different aspects of the system, and also construct executable software code
- It shows the attributes, classes, functions, and relationships to give an overview of the software system

UML Class Diagram

- It constitutes class names, attributes, and functions in a separate compartment that helps in software development
- Since it is a collection of classes, interfaces, associations, collaborations, and constraints, it is termed as a structural diagram
- **Purpose :**
- The main purpose of class diagrams is to build a static view of an application
- It is the only diagram that is widely used for construction, and it can be mapped with object-oriented languages
- It is one of the most popular UML diagrams.

UML Class Diagram

1. It analyses and designs a static view of an application
2. It describes the major responsibilities of a system
3. It is a base for component and deployment diagrams
4. It incorporates forward and reverse engineering

UML Class Diagram

- Benefits of Class Diagrams :

1. It can represent the object model for complex systems
2. It reduces the maintenance time by providing an overview of how an application is structured before coding
3. It provides a general schematic of an application for better understanding
4. It represents a detailed chart by highlighting the desired code, which is to be programmed
5. It is helpful for the stakeholders and the developers

UML Class Diagram

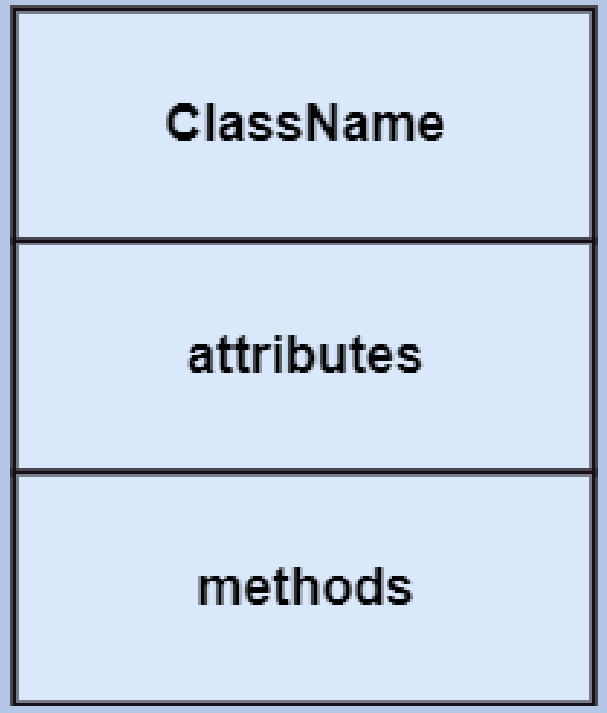
- Vital components of a Class Diagram: Made up of three sections
- **Upper Section**: The upper section encompasses the name of the class. A class is a representation of similar objects that shares the same relationships, attributes, operations, and semantics. Some of the following rules that should be taken into account while representing a class are given below:
- Capitalize the initial letter of the class name.
- Place the class name in the center of the upper section.
- A class name must be written in bold format.
- The name of the abstract class should be written in italics format

UML Class Diagram

- **Middle Section:** The middle section constitutes the attributes, which describe the quality of the class. The attributes have the following characteristics:
- The attributes are written along with its visibility factors, which are public (+), private (-), protected (#), and package (~).
- The accessibility of an attribute class is illustrated by the visibility factors.
- A meaningful name should be assigned to the attribute, which will explain its usage inside the class.

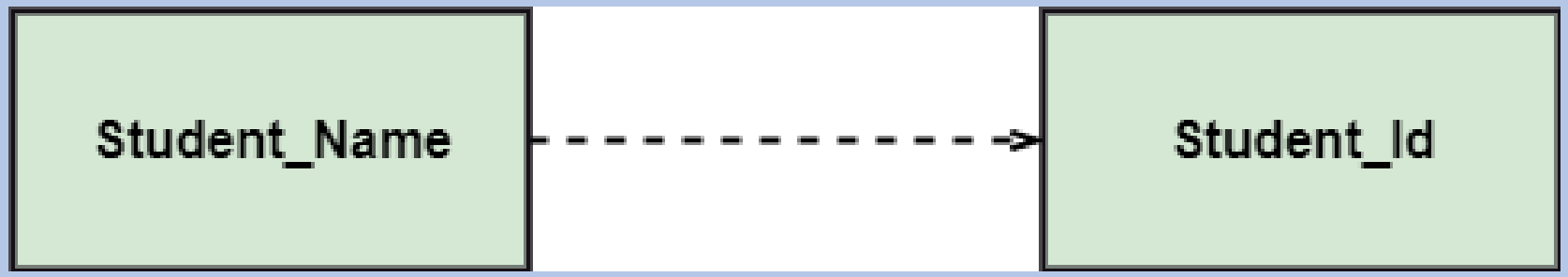
UML Class Diagram

- **Lower Section:** The lower section contain methods or operations. The methods are represented in the form of a list, where each method is written in a single line. It demonstrates how a class interacts with data



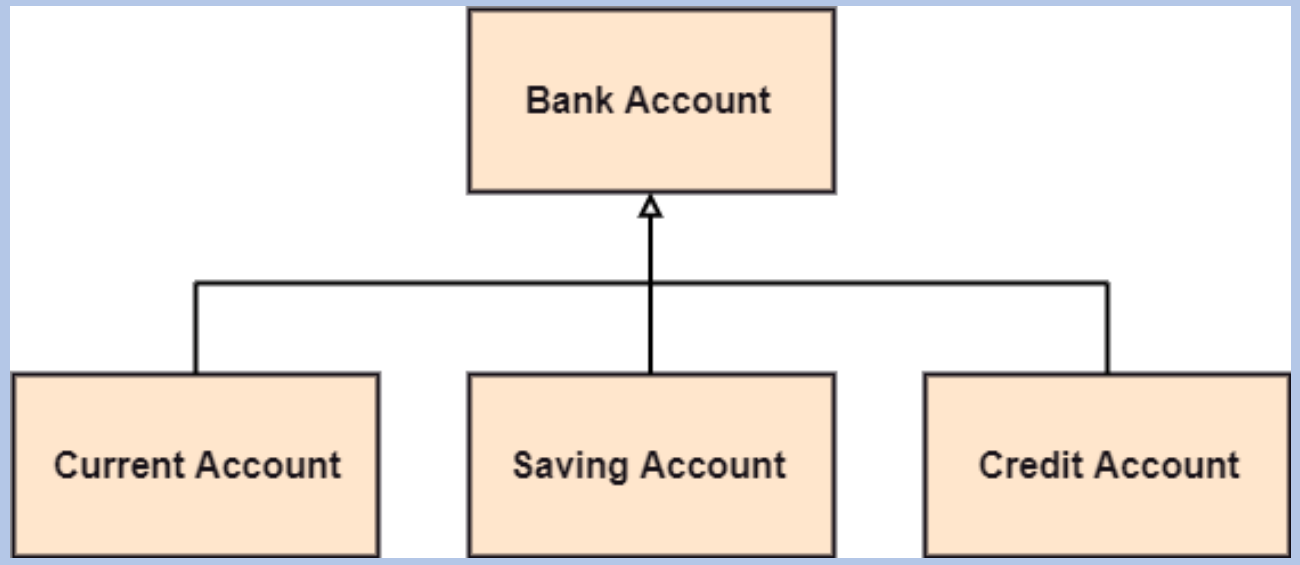
Relationships

- 3 types
- **Dependency:** A dependency is a semantic relationship between two or more classes where a change in one class cause changes in another class. It forms a weaker relationship.
- Student_Name is dependent on the Student_Id



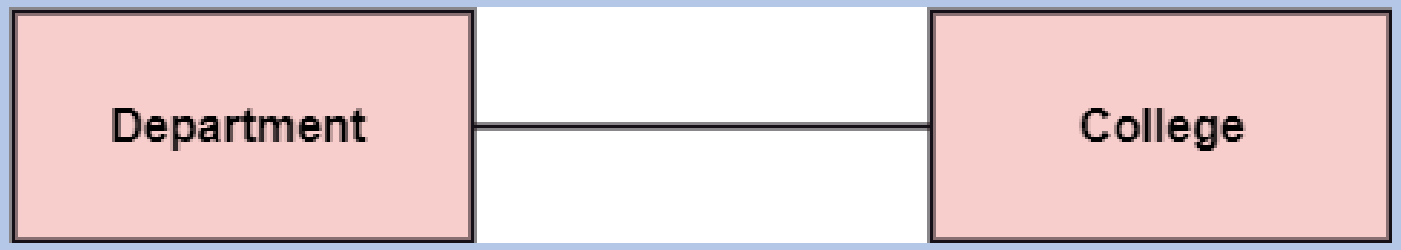
Relationships

- **Generalization**: A generalization is a relationship between a parent class (superclass) and a child class (subclass). In this, the child class is inherited from the parent class.
- For example, The Current Account, Saving Account, and Credit Account are the generalized form of Bank Account



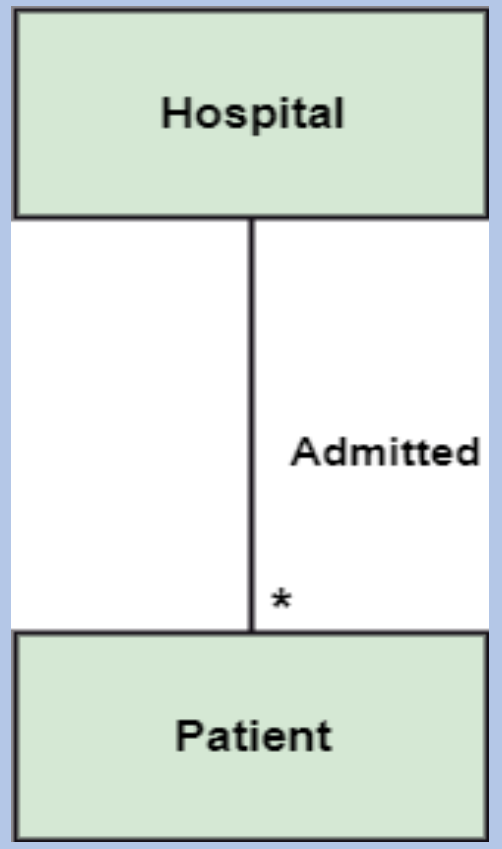
Relationships

- **Association:** It describes a static or physical connection between two or more objects. It depicts how many objects are there in the relationship.
- For example, a department is associated wih the college.



Relationships

- **Multiplicity:** It defines a specific range of allowable instances of attributes. In case if a range is not specified, one is considered as a default multiplicity.
- For example, multiple patients are admitted to one hospital.



Relationships

- **Aggregation:** An aggregation is a subset of association, which represents has a relationship. It is more specific then association. It defines a part-whole or part-of relationship. In this kind of relationship, the child class can exist independently of its parent class.
- The company encompasses a number of employees, and even if one employee resigns, the company still exists

Relationships

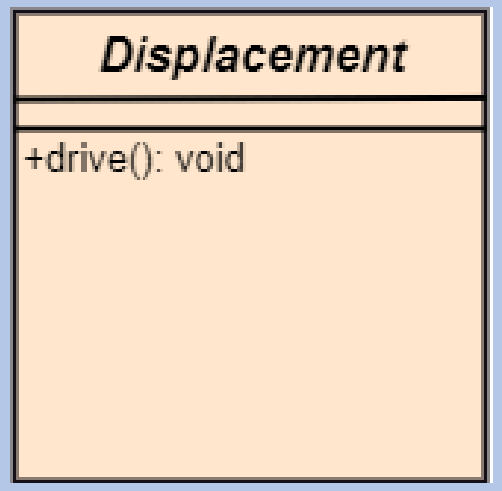
- **Composition:** The composition is a subset of aggregation. It portrays the dependency between the parent and its child, which means if one part is deleted, then the other part also gets discarded. It represents a whole-part relationship.
- A contact book consists of multiple contacts, and if you delete the contact book, all the contacts will be lost

Abstract Classes

- No objects can be a direct entity of the abstract class
- The abstract class can neither be declared nor be instantiated
- It is used to find the functionalities across the classes
- The notation of the abstract class is similar to that of class; the only difference is that the name of the class is written in italics
- Since it does not involve any implementation for a given function, it is best to use the abstract class with multiple objects

Abstract Classes

- Let us assume that we have an abstract class named **displacement** with a method declared inside it, and that method will be called as a drive ()
- Now, this abstract class method can be implemented by any object, for example, car, bike, scooter, cycle, etc



How to draw a Class Diagram?

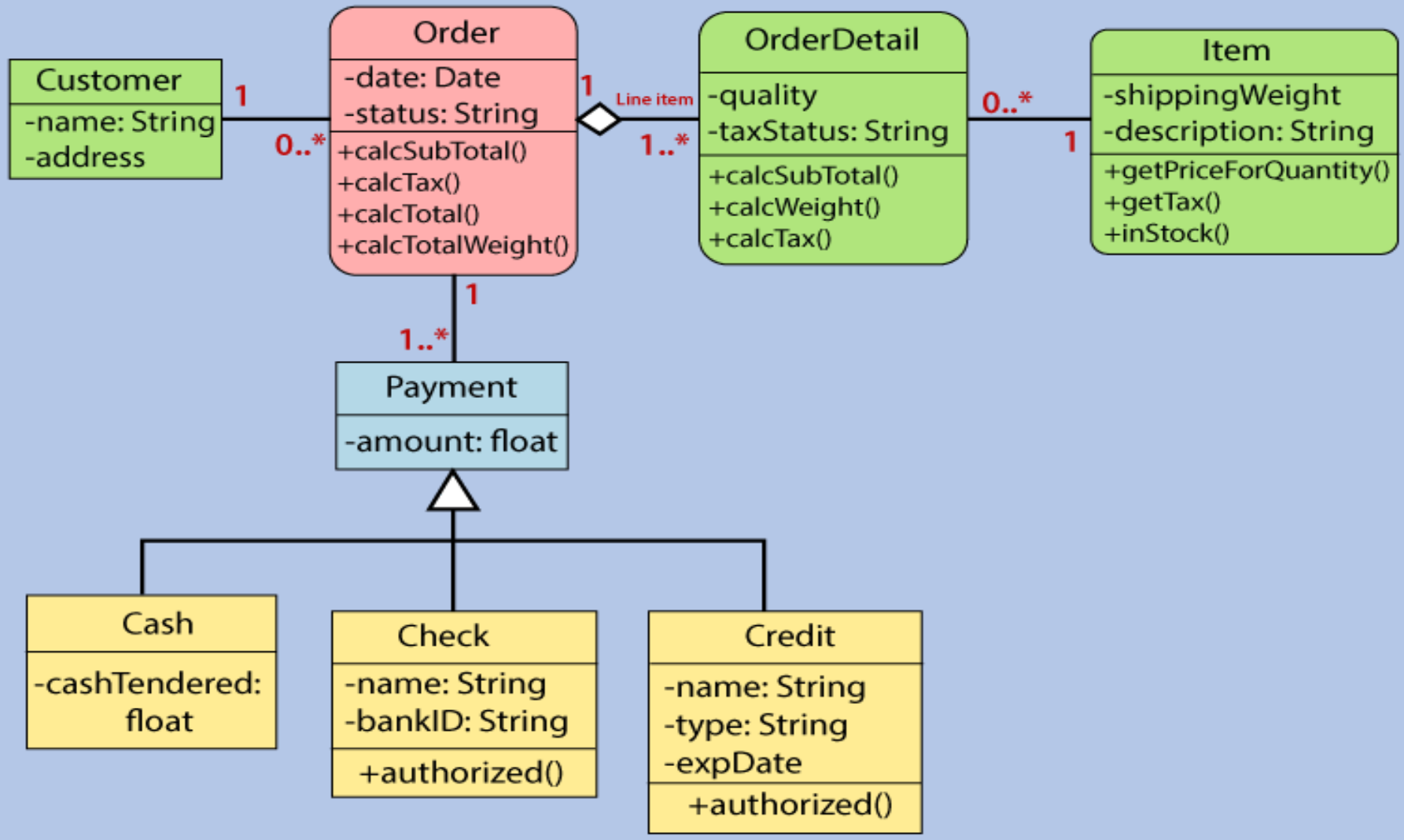
1. To describe a complete aspect of the system, it is suggested to give a meaningful name to the class diagram
2. The objects and their relationships should be acknowledged in advance
3. The attributes and methods (responsibilities) of each class must be known
4. A minimum number of desired properties should be specified as more number of the unwanted property will lead to a complex diagram

How to draw a Class Diagram?

5. Notes can be used as and when required by the developer to describe the aspects of a diagram
6. The diagrams should be redrawn and reworked as many times to make it correct before producing its final version

- Ex. Class Diagram for Sales order system

How to draw a Class Diagram?



Use of Class Diagram

- The class diagram is used to represent a static view of the system
- It plays an essential role in the establishment of the component and deployment diagrams
- It helps to construct an executable code to perform forward and backward engineering for any system, or we can say it is mainly used for construction
- It represents the mapping with object-oriented languages that are C++, Java, etc.

Use of Class Diagram

- To describe the static view of a system
- To show the collaboration among every instance in the static view
- To describe the functionalities performed by the system
- To construct the software application using object-oriented language