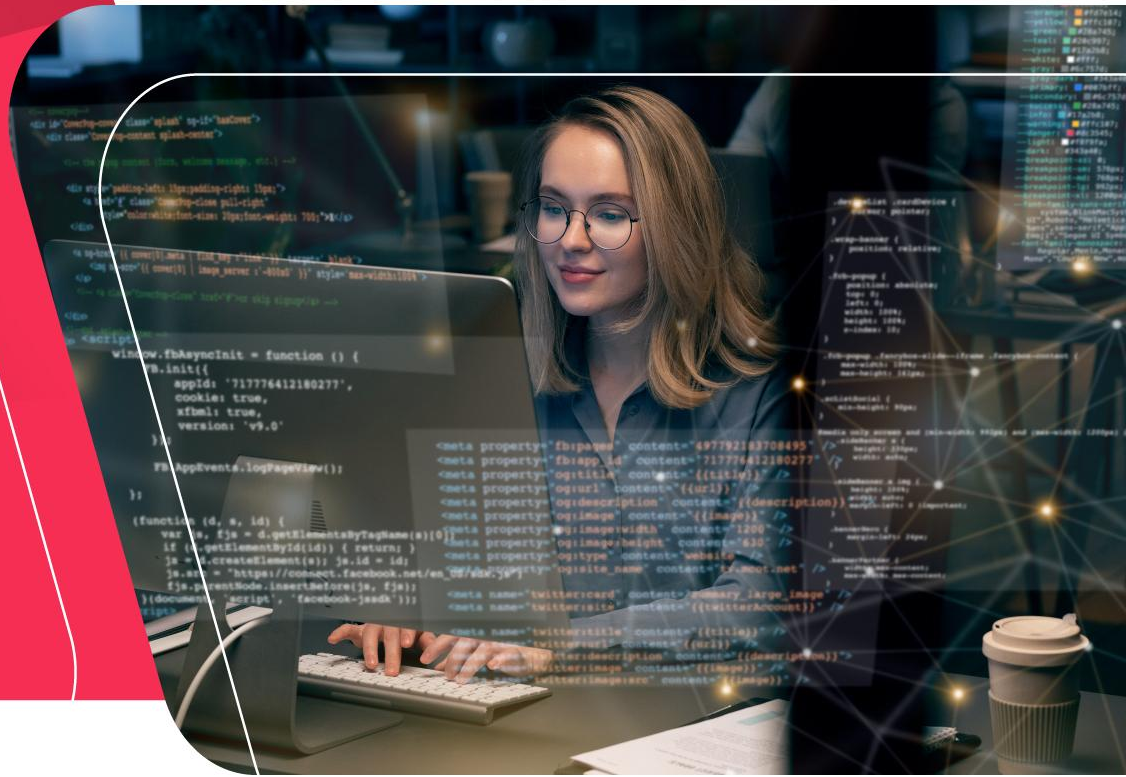
TIA-Academy

Java Bootcamp

Day 01



TIA-Academy

Unified Modelling Language (UML)

Unified Modeling Language

UML is a **standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems**.

- UML stands for Unified Modeling 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 modeling 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.

Unified Modeling Language

- **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.

Goals of UML

- **A picture is worth a thousand words**, this idiom absolutely fits describing UML. Object-oriented concepts were introduced much earlier than UML. At that point of time, there were no standard methodologies to organize and consolidate the object-oriented development. It was then that UML came into picture.
- There are a number of goals for developing UML but the most important is to define some general purpose modeling language, which all modelers can use and it also needs to be made simple to understand and use.

Goals of UML

- 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.
- In conclusion, the goal of UML can be defined as **a simple modeling mechanism to model all possible practical systems in today's complex environment.**

A Conceptual Model of UML

To understand the conceptual model of UML, first we need to clarify what is a conceptual model? and why a conceptual model is required?

- A conceptual model can be defined as **a model which is made of concepts and their relationships.**
- A conceptual model is the first step before drawing a UML diagram. **It helps to understand the entities in the real world and how they interact with each other**.

A Conceptual Model of UML

As UML describes the real-time systems, it is very important to **make a conceptual model and then proceed gradually**. The conceptual model of UML can be mastered by learning the following three major elements −

- UML building blocks
- Rules to connect the building blocks
- Common mechanisms of UML

UML-Building Blocks

UML is composed of three main building blocks, i.e., things, relationships, and diagrams. Building blocks generate one complete UML model diagram by rotating around several different blocks. It plays an essential role in developing UML diagrams. The basic UML building blocks are enlisted below:

1. Things
2. Relationships
3. Diagrams

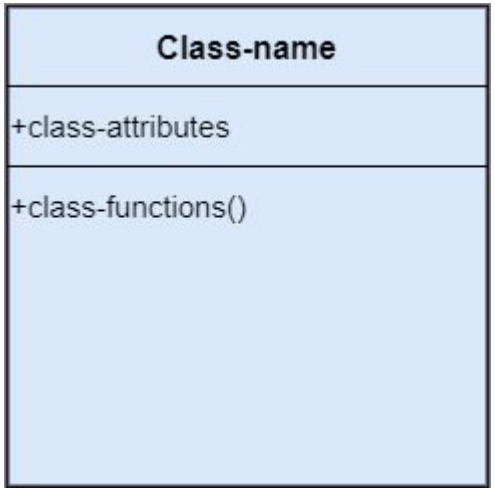
1. Things

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

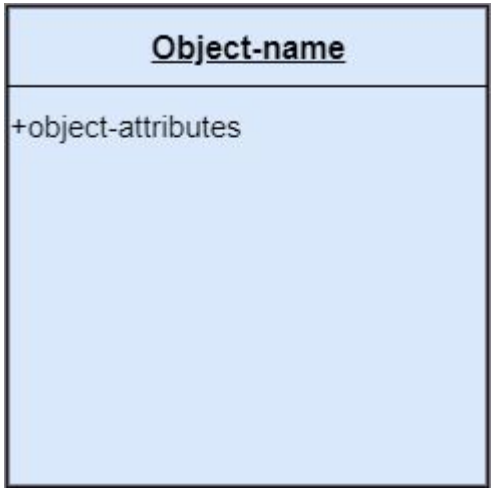
Structural things

- 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.
- **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. Its notation is as follows;



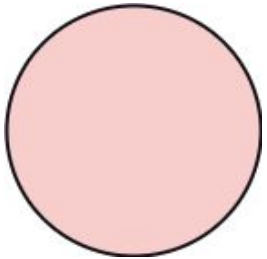
Structural things

- 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;

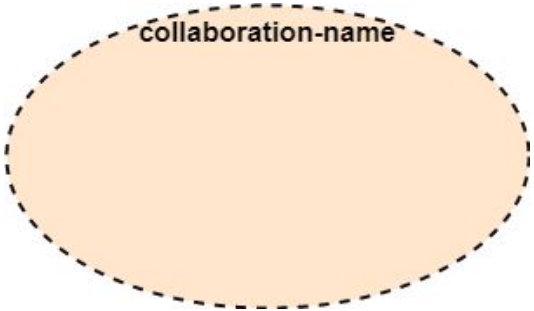


Structural things

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

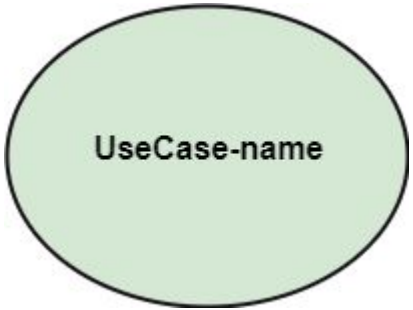


- **Collaboration**: It represents **the interaction between things that is done to meet the goal**. It is symbolized as a dotted ellipse with its name written inside it.

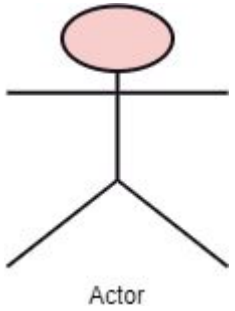


Structural things

- **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**.

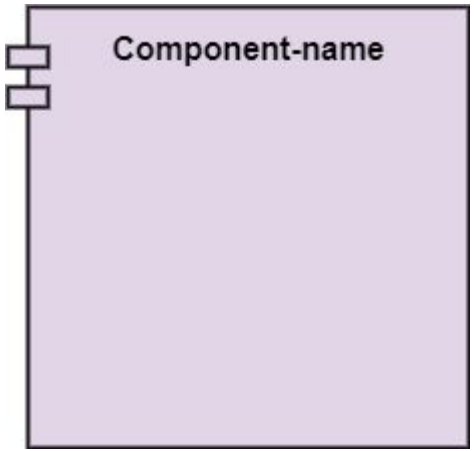


- **Actor**: It comes under the use case diagrams. It is an **object that interacts with the system**, for example, a user.

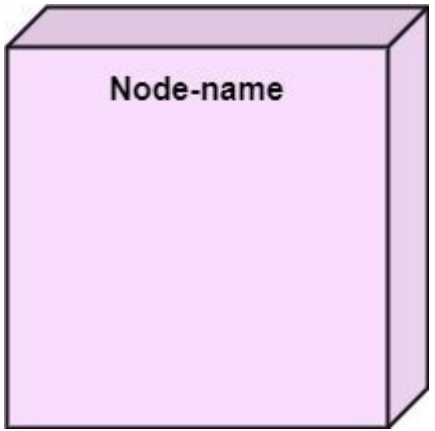


Structural things

Component: It represents the **physical part of the system.**

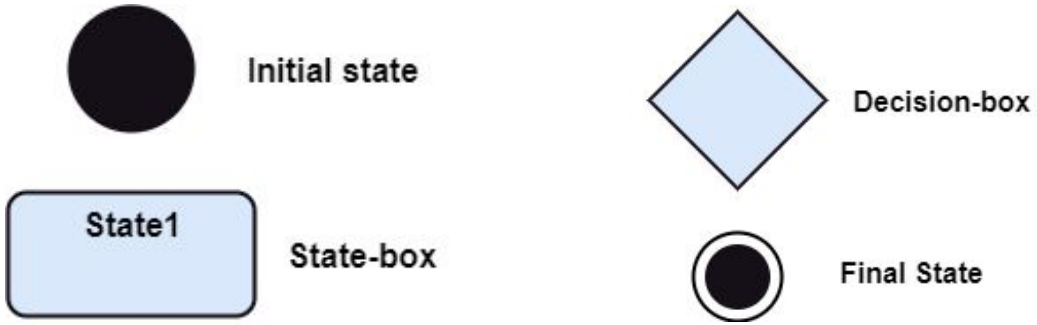


Node: A physical **element that exists at run time.**



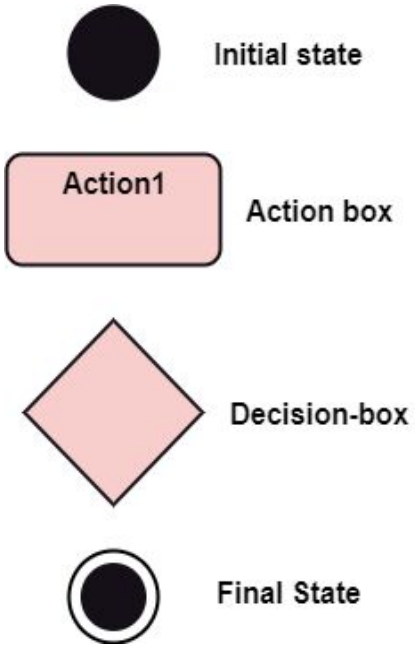
Structural things

- 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
- **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.



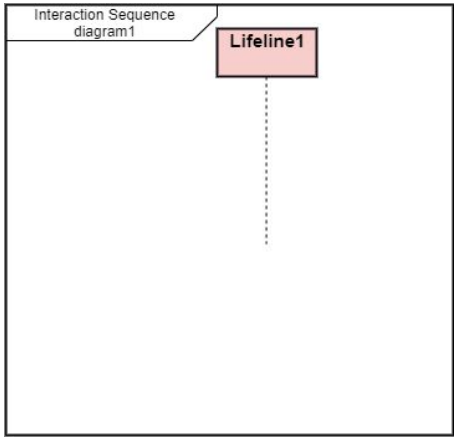
Behavioural Things

- **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.



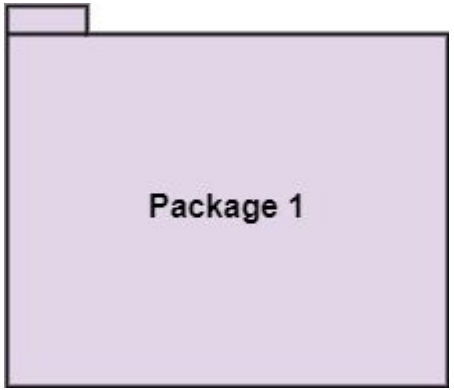
Behavioural Things

- **Interaction Diagram:** It is used to envision **the flow of messages between several components** in a system.



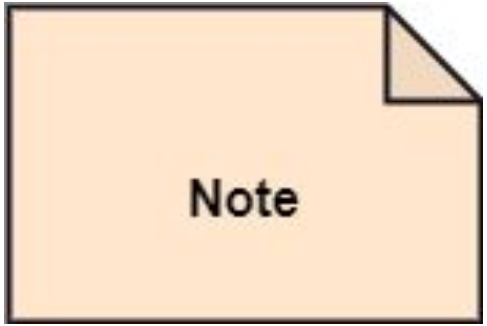
Behavioural Things

- 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.
- **Package**: Package is the only thing that is available for grouping behavioral and structural things.



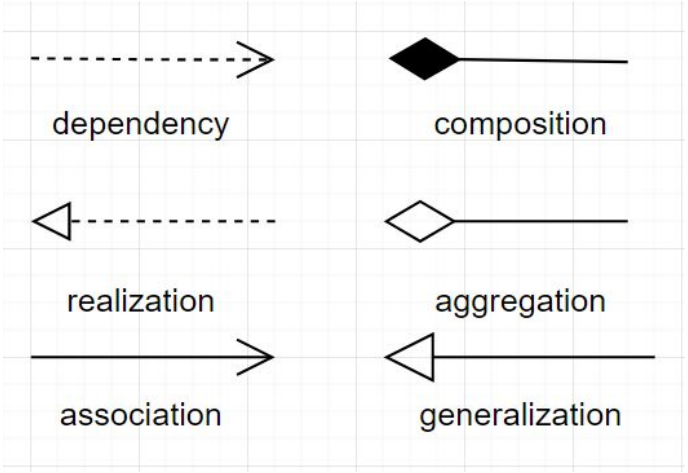
Structural things

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



2. 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 given below.



2. Relationships

- **Dependency**: Dependency is 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,

2. 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.

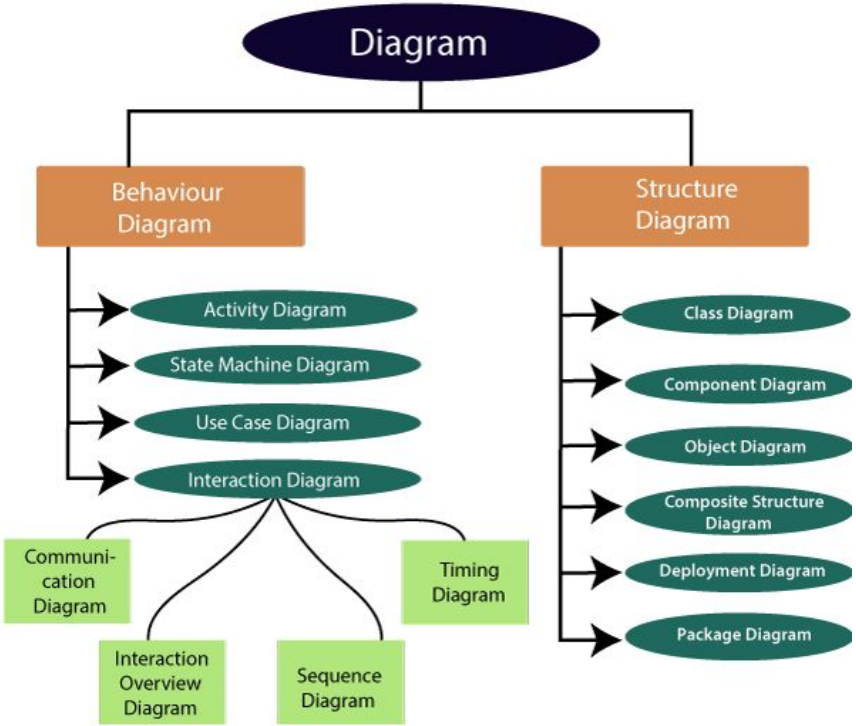
2. Relationships

- **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.

2. 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.

3. UML-Diagrams



- The UML diagrams are categorized into structural diagrams, behavioral diagrams, and also interaction overview diagrams. The diagrams are hierarchically classified in the following figure:

Structural 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. It presents an outline for the system. It stresses the elements to be present that are to be modeled.

- **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.
- **Object Diagram**: It 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**: It 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.
- **Deployment Diagram**: It 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.
- **Package Diagram**: It is 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.

- **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.
- **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.
- **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

Interaction diagrams are 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.

- **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.
- **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.
- **Timing Diagram**: It is 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 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.

Use Case Diagram

- A use case diagram is used to **represent the dynamic behavior of a system**.
- It **encapsulates the system's functionality** by incorporating use cases, actors, and their relationships.
- It models the tasks, services, and functions required by a system/subsystem of an application.
- It **depicts the high-level functionality of a system** and also tells how the user handles a system.

Purpose of Use Case Diagrams

- The main purpose of a use case diagram is to **portray the dynamic aspect of a system**.
- It **accumulates the system's requirement**, which includes both internal as well as external influences. It invokes persons, use cases, and several things that invoke the actors and elements accountable for the implementation of use case diagrams.
- It represents how an **entity from the external environment can interact** with a part of the system.
  - It gathers the system's needs.
  - It depicts the external view of the system.
  - It recognizes the internal as well as external factors that influence the system.
  - It represents the interaction between the actors.

How to draw a Use Case diagram?

- It is essential to **analyze the whole system before starting with drawing a use case diagram,** and then the system's functionalities are found. And once every single functionality is identified, they are then transformed into the use cases to be used in the use case diagram.
- After that, we will **enlist the actors that will interact with the system**. The actors are the person or a thing that invokes the functionality of a system. It may be a system or a private entity, such that it requires an entity to be pertinent to the functionalities of the system to which it is going to interact.
- Once both the actors and use cases are enlisted, **the relation between the actor and use case/ system is inspected**. It identifies the no of times an actor communicates with the system. Basically, an actor can interact multiple times with a use case or system at a particular instance of time.

How to draw a Use Case diagram?

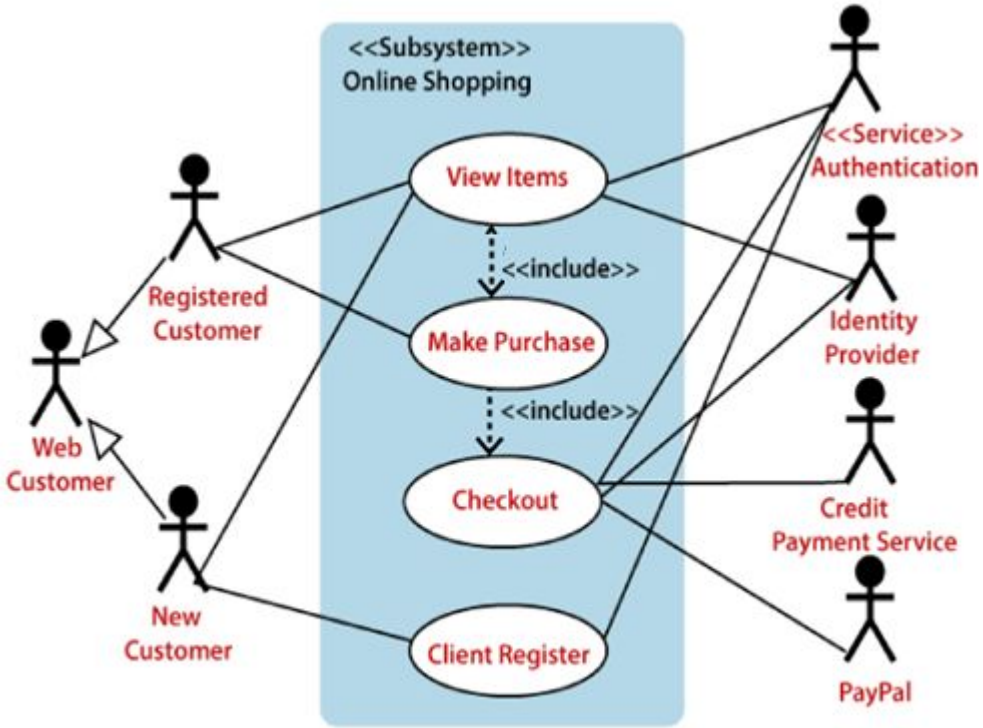
Following are some rules that must be followed while drawing a use case diagram:

- A pertinent and meaningful name should be assigned to the actor or a use case of a system.
- The communication of an actor with a use case must be defined in an understandable way.
- Specified notations to be used as and when required.
- The most significant interactions should be represented among the multiple no of interactions between the use case and actors.

Example of a Use Case Diagram

- A use case diagram depicting the **Online Shopping** website is given next.
- Here the Web Customer actor makes **use of any online shopping website to purchase online.**
- The top-level uses are as follows; **View Items, Make Purchase, Checkout, Client Register.**
- The **View Items** use case is utilized by the customer who searches and view products.
- The **Client Register** use case allows the customer to register itself with the website for availing gift vouchers, coupons, or getting a private sale invitation.
- It is to be noted that the **Checkout** is an included use case, which is part of **Making Purchase**, and it is not available by itself.

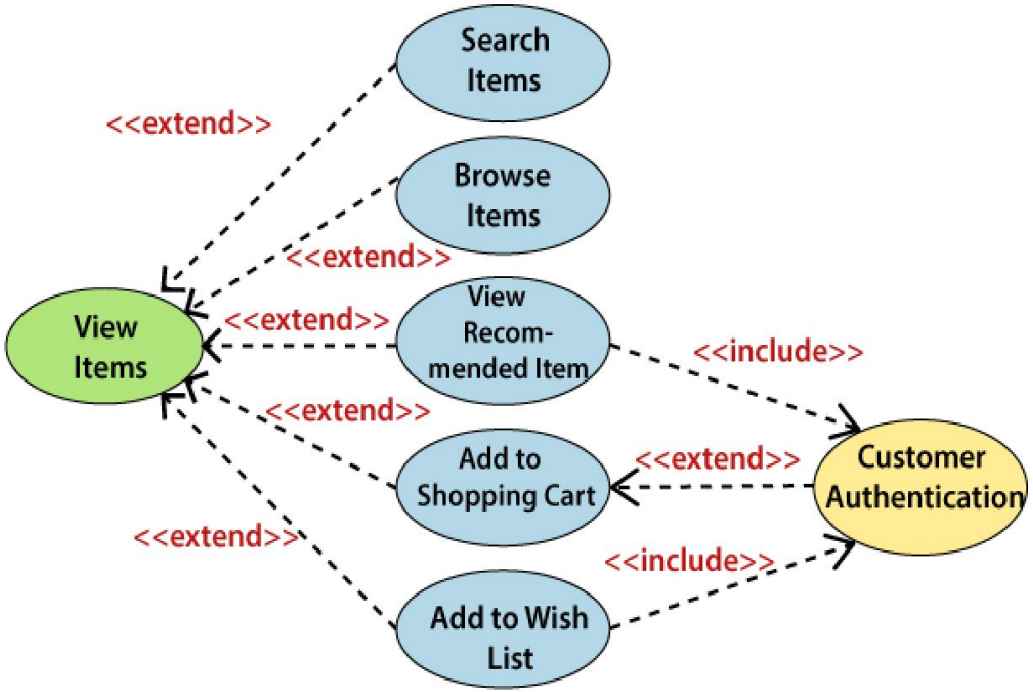
Example of a Use Case Diagram



Example of a Use Case Diagram

- The **View Items** is further extended by several use cases such as; Search Items, Browse Items, View Recommended Items, Add to Shopping Cart, Add to Wish list.
- All of these extended use cases provide some functions to customers, which allows them to search for an item.
- Both **View Recommended Item** and **Add to Wish List** include the Customer Authentication use case, as they necessitate authenticated customers, and simultaneously item can be added to the shopping cart without any user authentication.

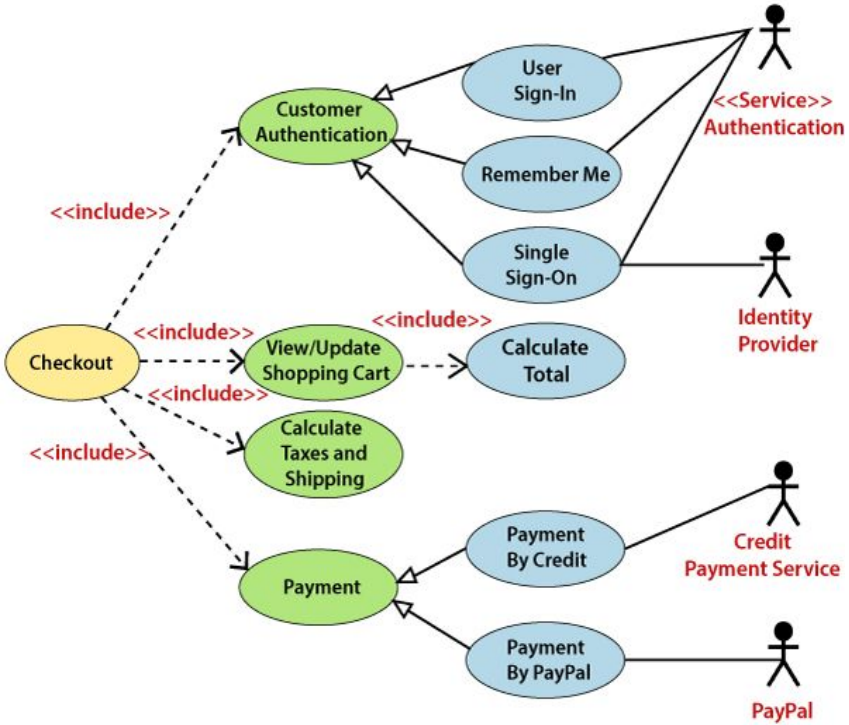
Example of a Use Case Diagram



Example of a Use Case Diagram

- Similarly, the **Checkout** use case also includes the following use cases, as shown below. It requires an authenticated Web Customer, which can be done by login page, user authentication cookie ("Remember me"), or Single Sign-On (SSO).
- SSO needs an external identity provider's participation, while Web site authentication service is utilized in all these use cases.
- The Checkout use case involves Payment use case that can be done either by the credit card and external credit payment services or with PayPal.

Example of a Use Case Diagram



Important tips for drawing a Use Case diagram

Following are some important tips that are to be kept in mind while drawing a use case diagram:

- A simple and complete use case diagram should be articulated.
- A use case diagram should represent the most significant interaction among the multiple interactions.
- At least one module of a system should be represented by the use case diagram.
- If the use case diagram is large and more complex, then it should be drawn more generalized.

TIA-Academy

ASSIGNMENT 01

UML Class Diagram

- The 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. 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 of Class Diagrams

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. Following are the purpose of class diagrams given below:

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

Benefits of Class Diagrams

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

Vital components of a Class Diagram

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.

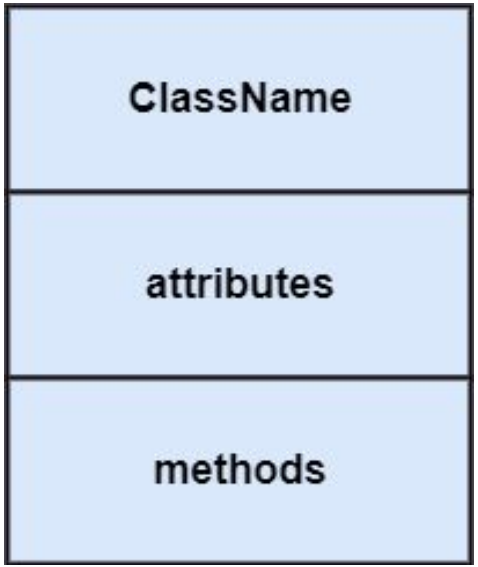
Vital components of a Class Diagram

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.

Vital components of a 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.

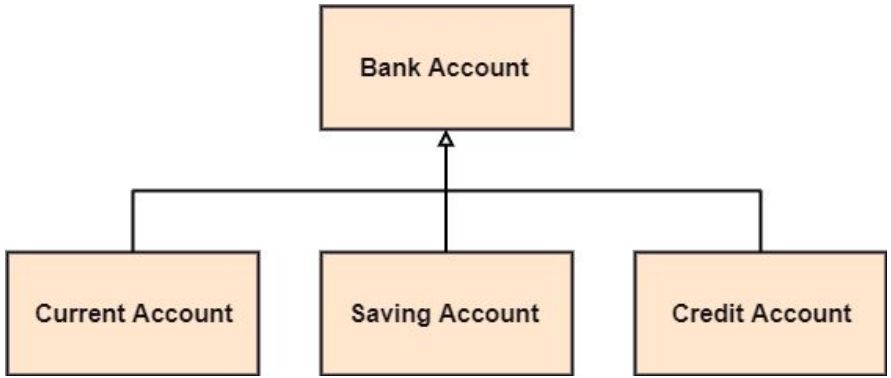


Relationships

In UML, relationships are of three types:

1. **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. In the following example, **Student_Name is dependent on the Student_Id.**

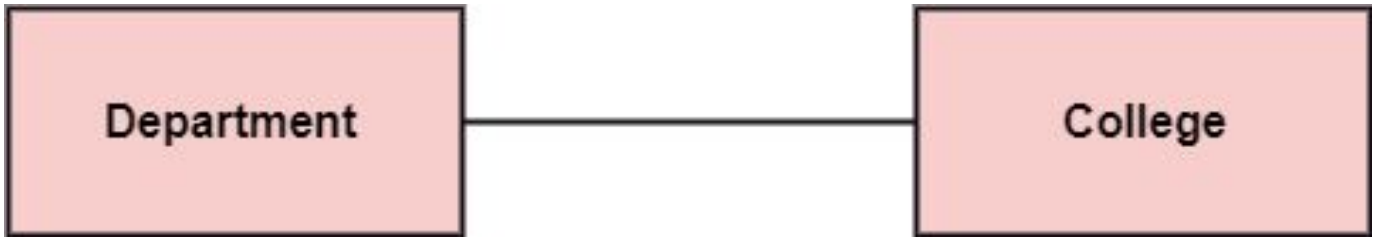
2. **Generalization: A generalization is a relationship between a parent class (superclass) and a child class (subclass).** In this, the child class is inherited from the parent class. For example, The Current Account, Saving Account, and Credit Account are the generalized form of Bank Account.



Relationships

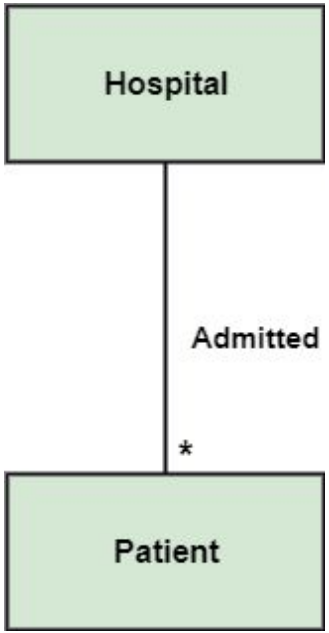
3. **Association**: It describes a static or physical connection between two or more objects. It depicts how many objects are there in the relationship.

   For example, a department is associated with the college.



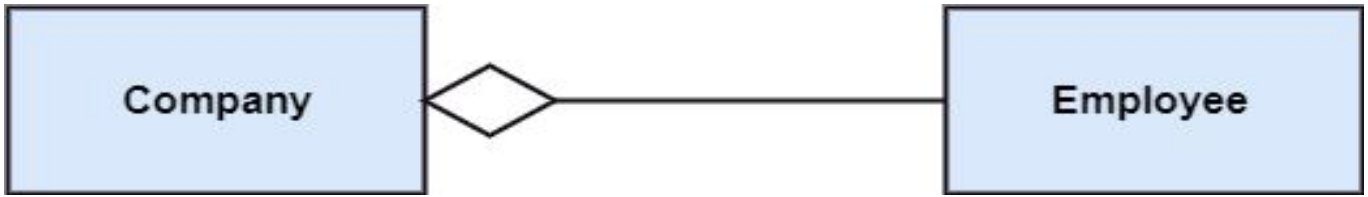
Relationships

- **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.



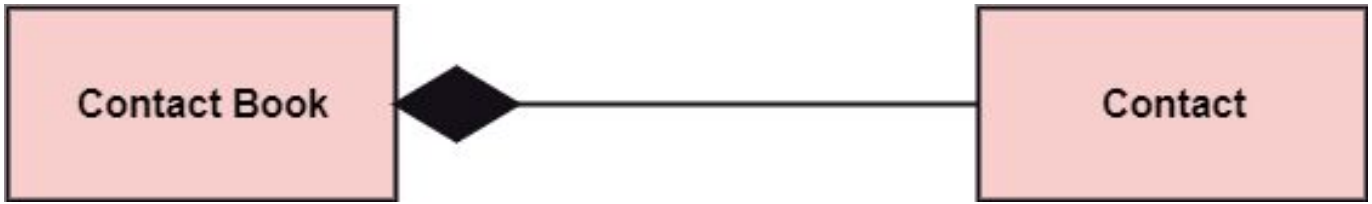
Relationships

- **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.



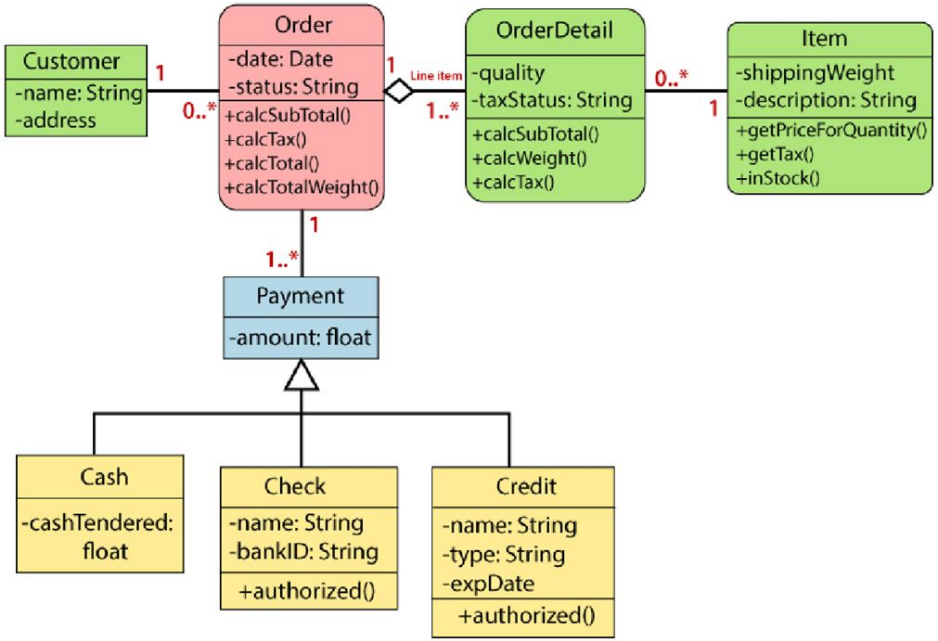
How to draw a Class Diagram?

The class diagram is used most widely to construct software applications. It not only represents a static view of the system but also all the major aspects of an application. A collection of class diagrams as a whole represents a system.Some key points that are needed to keep in mind while drawing a class diagram are given below:

- To describe a complete aspect of the system, it is suggested to give a meaningful name to the class diagram.
- The objects and their relationships should be acknowledged in advance.
- The attributes and methods (responsibilities) of each class must be known.
- A minimum number of desired properties should be specified as more number of the unwanted property will lead to a complex diagram.
- Notes can be used as and when required by the developer to describe the aspects of a diagram.
- The diagrams should be redrawn and reworked as many times to make it correct before producing its final version.

Class Diagram Example

A class diagram describing the sales order system is given below.

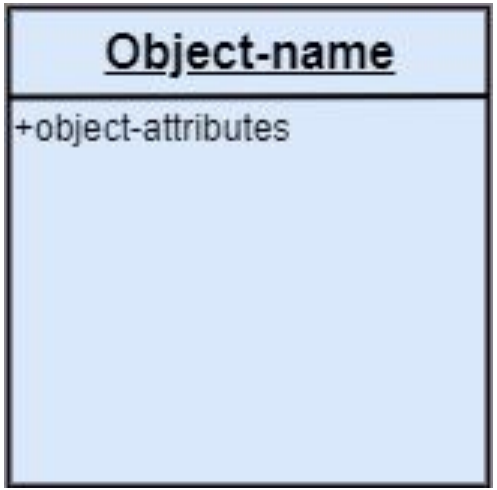


Usage of Class diagrams

- 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.
- Class diagrams can be used for the following purposes:
  - 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 languages.

Object Diagram

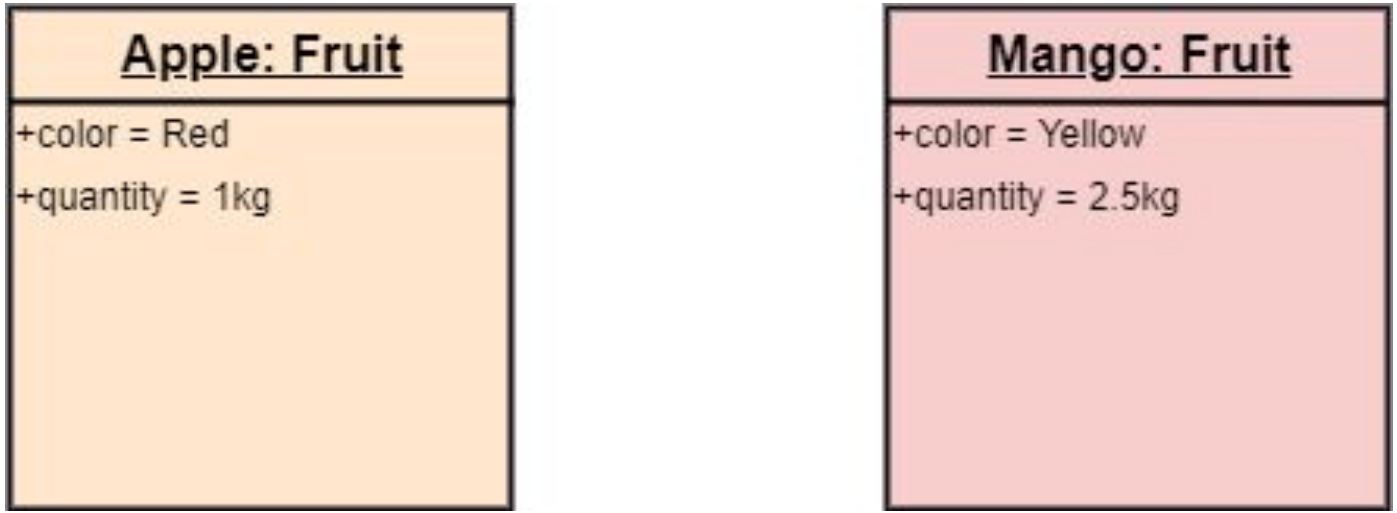
- Object diagrams are dependent on the class diagram as they are derived from the class diagram. It represents an instance of a class diagram. The objects help in portraying a static view of an object-oriented system at a specific instant.
- Both the object and class diagram are similar to some extent; the only difference is that the class diagram provides an abstract view of a system. It helps in visualizing a particular functionality of a system.
- Notation of an Object Diagram :



Purpose of Object Diagram

- The object diagram holds the same purpose as that of a class diagram. The class diagram provides an abstract view which comprises of classes and their relationships, whereas the object diagram represents an instance at a particular point of time.
- The object diagram is actually similar to the concrete (actual) system behavior. The main purpose is to depict a static view of a system.
- Following are the purposes enlisted below:
  - It is used to perform forward and reverse engineering.
  - It is used to understand object behavior and their relationships practically.
  - It is used to get a static view of a system.
  - It is used to represent an instance of a system.

Example of Object Diagram



How to draw an Object Diagram?

- All the objects present in the system should be examined before start drawing the object diagram.
- Before creating the object diagram, the relation between the objects must be acknowledged.
- The association relationship among the entities must be cleared already.
- To represent the functionality of an object, a proper meaningful name should be assigned.
- The objects are to be examined to understand its functionality.

Applications of Object diagrams

The following are the application areas where the object diagrams can be used.

- To build a prototype of a system.
- To model complex data structures.
- To perceive the system from a practical perspective.
- Reverse engineering.

Class vs. Object diagram

Serial No.	Class Diagram	Object Diagram
1.	It depicts the static view of a system.	It portrays the real-time behavior of a system.
2.	Dynamic changes are not included in the class diagram.	Dynamic changes are captured in the object diagram.
3.	The data values and attributes of an instance are not involved here.	It incorporates data values and attributes of an entity.
4.	The object behavior is manipulated in the class diagram.	Objects are the instances of a class.

Activity Diagram

- In UML, the activity diagram is used to demonstrate the flow of control within the system rather than the implementation. It models the concurrent and sequential activities.
- The activity diagram helps in envisioning the workflow from one activity to another. It put emphasis on the condition of flow and the order in which it occurs. The flow can be sequential, branched, or concurrent, and to deal with such kinds of flows, the activity diagram has come up with a fork, join, etc.
- It is also termed as an **object-oriented flowchart**. It encompasses activities composed of a set of actions or operations that are applied to model the behavioral diagram.

Components of an Activity Diagram

Following are the component of an activity diagram:

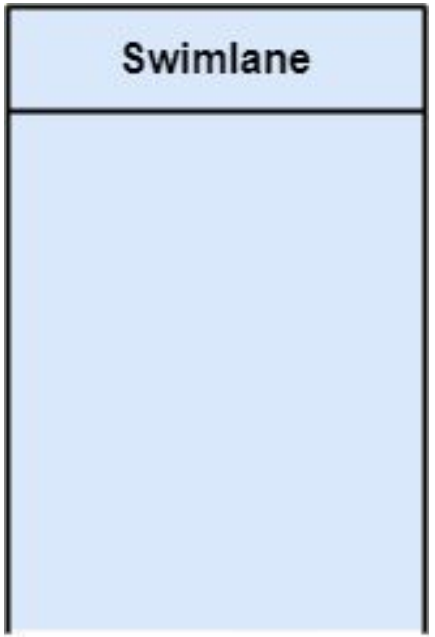
Activities

- The categorization of behavior into one or more actions is termed as an activity. In other words, it can be said that an activity is a network of nodes that are connected by edges. The edges depict the flow of execution. It may contain action nodes, control nodes, or object nodes.
- The control flow of activity is represented by control nodes and object nodes that illustrates the objects used within an activity. The activities are initiated at the initial node and are terminated at the final node.

Components of an Activity Diagram

Activity partition /swimlane

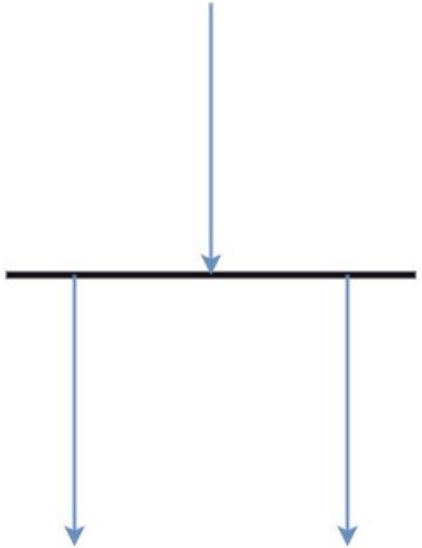
- The swimlane is used to cluster all the related activities in one column or one row. It can be either vertical or horizontal. It used to add modularity to the activity diagram.
- It is not necessary to incorporate swimlane in the activity diagram. But it is used to add more transparency to the activity diagram.



Components of an Activity Diagram

Forks

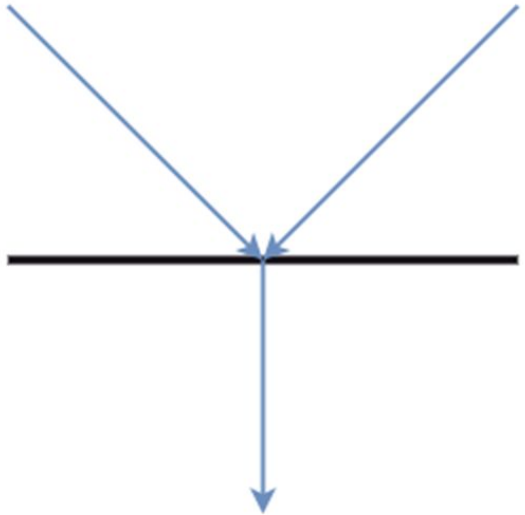
- Forks and join nodes generate the concurrent flow inside the activity. A fork node consists of one inward edge and several outward edges. It is the same as that of various decision parameters.
- Whenever a data is received at an inward edge, it gets copied and split crossways various outward edges. It split a single inward flow into multiple parallel flows.



Components of an Activity Diagram

Join Nodes

- Join nodes are the opposite of fork nodes. A Logical AND operation is performed on all of the inward edges as it synchronizes the flow of input across one single output (outward) edge.



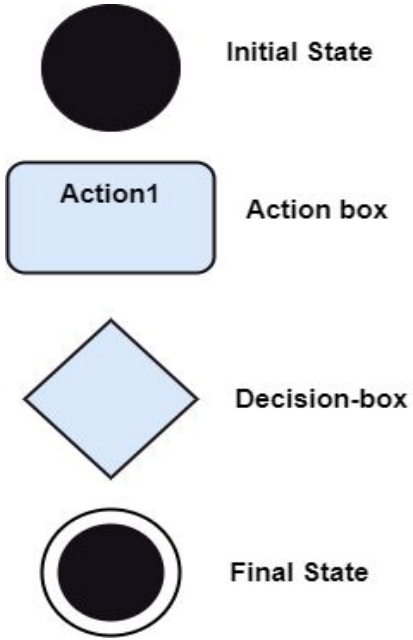
Pins

- It is a small rectangle, which is attached to the action rectangle. It clears out all the messy and complicated thing to manage the execution flow of activities. It is an object node that precisely represents one input to or output from the action.

Notation of an Activity diagram

Activity diagram constitutes following notations:

- **Initial State**: It depicts the initial stage or beginning of the set of actions.
- **Final State**: It is the stage where all the control flows and object flows end.
- **Decision Box**: It makes sure that the control flow or object flow will follow only one path.
- **Action Box**: It represents the set of actions that are to be performed.



Why use Activity Diagram?

- An event is created as an activity diagram encompassing a group of nodes associated with edges. To model the behavior of activities, they can be attached to any modeling element. It can model use cases, classes, interfaces, components, and collaborations.
- It mainly models processes and workflows. It envisions the dynamic behavior of the system as well as constructs a runnable system that incorporates forward and reverse engineering. It does not include the message part, which means message flow is not represented in an activity diagram.
- It is the same as that of a flowchart but not exactly a flowchart itself. It is used to depict the flow between several activities.

How to draw an Activity Diagram?

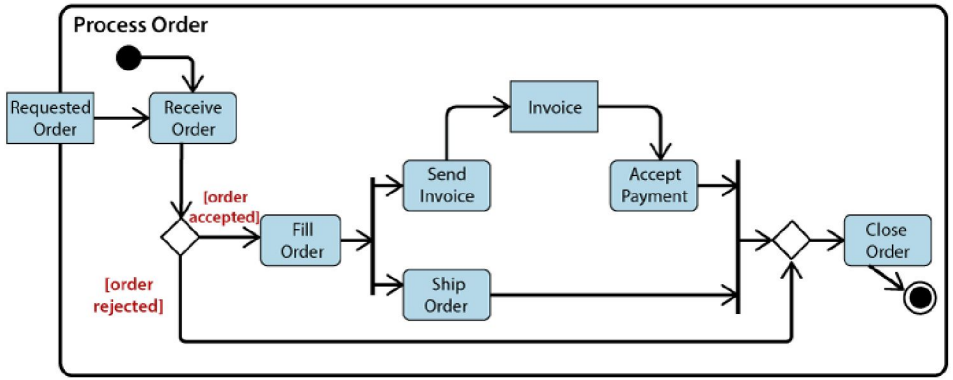
- An activity diagram is a flowchart of activities, as it represents the workflow among various activities. They are identical to the flowcharts, but they themself are not exactly the flowchart.
- In other words, it can be said that an activity diagram is an enhancement of the flowchart, which encompasses several unique skills.
- Since it incorporates swimlanes, branching, parallel flows, join nodes, control nodes, and forks, it supports exception handling. A system must be explored as a whole before drawing an activity diagram to provide a clearer view of the user.
- All of the activities are explored after they are properly analyzed for finding out the constraints applied to the activities. Each and every activity, condition, and association must be recognized.

How to draw an Activity Diagram?

- After gathering all the essential information, an abstract or a prototype is built, which is then transformed into the actual diagram.
- Following are the rules that are to be followed for drawing an activity diagram:
  - A meaningful name should be given to each and every activity.
  - Identify all of the constraints.
  - Acknowledge the activity associations.

Example of an Activity Diagram

- An example of an activity diagram showing the business flow activity of order processing is given below.
- Here the input parameter is the **Requested order**, and once the order is accepted, all of the required information is then filled, payment is also accepted, and then the order is shipped. It permits order shipment before an invoice is sent or payment is completed.



When to use an Activity Diagram?

An activity diagram can be used to portray business processes and workflows. Also, it used for modeling business as well as the software. An activity diagram is utilized for the followings:

- To graphically model the workflow in an easier and understandable way.
- To model the execution flow among several activities.
- To model comprehensive information of a function or an algorithm employed within the system.
- To model the business process and its workflow.
- To envision the dynamic aspect of a system.
- To generate the top-level flowcharts for representing the workflow of an application.
- To represent a high-level view of a distributed or an object-oriented system.

TIA-Academy

Thank You