UNIT-4 UML

UML is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems.
UML was created by the Object Management Group (OMG) and UML 1.0 specification draft was proposed to the OMG in January 1997.
OMG is continuously making efforts to create a truly industry standard.

- 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.

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.
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.

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 −

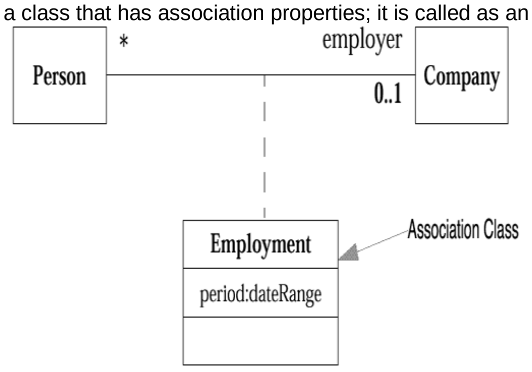
- UML building blocks
- Rules to connect the building blocks
- Common mechanisms of UML

1. Association

It is a structural relationship that represents objects can be connected or associated with another object inside the system. Following constraints can be applied to the association relationship.

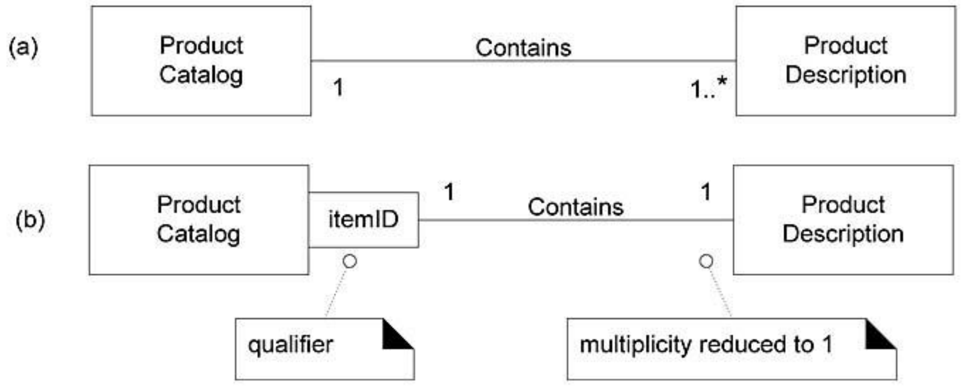
- **{implicit}** – Implicit constraints specify that the relationship is not manifest; it is based upon a concept.
- **{ordered}** – Ordered constraints specify that the set of objects at one end of an association are in a specific way.
- **{changeable}** – Changeable constraint specifies that the connection between various objects in the system can be added, removed, and modified as per the requirement.
- **{addOnly}** – It specifies that the new connections can be added from an object which is situated at the other end an association.
- **{frozen}** – It specifies that when a link is inserted between two objects, then it cannot be modified while the frozen constraint is active on the given link or a connection.

We can also create a class that has association properties; it is called as an association class.



2. Qualified Association

- A qualified association has a qualifier that is used to select an object (or objects) from a larger set of related objects, based upon the qualifier key. Informally, in a software perspective, it suggests looking things up by a key, such as objects in a HashMap. For example, if a ProductCatalog contains many ProductDescriptions, and each one can be selected by an itemID, then the UML notation in Figure 16.15 can be used to depict this.
- There's one subtle point about qualified associations: the change in multiplicity. For example, as contrasted in Figure 16.15 (a) vs. (b), qualification reduces the multiplicity at the target end of the association, usually down from many to one, because it implies the selection of usually one instance from a larger set.

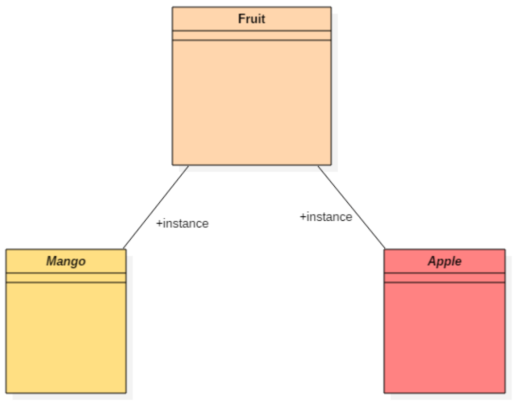


3. Reflexive association

- The reflexive association is a subtype of association relationship in UML. In a reflexive association, the instances of the same class can be related to each other. An instance of a class is also said to be an object.
- Reflexive association states that a link or a connection can be present within the objects of the same class.

Example:

- Let us consider an example of a class fruit.
- The fruit class has two instances, such as mango and apple. Reflexive association states that a link between mango and apple can be present as they are instances of the same class, such as fruit.



4. Multiplicity:

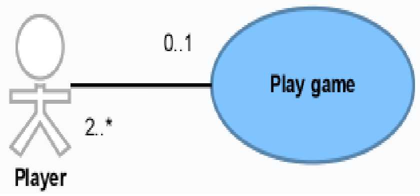
Multiplicity can be set for attributes, operations and associations in a UML class diagram, and for associations in a use case diagram. The multiplicity is an indication of how many objects may participate in the given relationship, or the allowable number of instances of the element.
In a use case diagram, multiplicity indicates how many actors can take part in how many occurrences of a use case. Multiplicity on a use case could mean that an actor interacts with multiple use cases, multiplicity on an actor could mean that one or more actors interact with a particular use case.

Example

The relationship between a player and the play game use case. A game may be played by many (two or more) players, but a player may not always participate in the game.
To show this relationship, the following multiplicity is set: on the side of the actor 2.., on the side of the use case *0..1. This is shown in the diagram as follows:

For a complete registration of a person's details, the person must at least have one address, but can have two addresses. Also, the person must have at least one telephone number, but may have many.
To show this, the following multiplicity is set: for the address attribute the multiplicity is 1..2, for the phone number attribute the multiplicity is set to 1..{*}. This is shown in the diagram as follows:

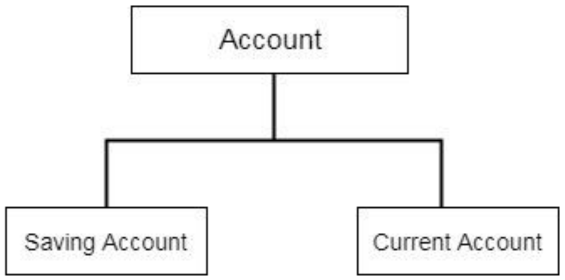
For a complete registration of a person's details, the person must at least have one address, but can have two addresses. Also, the person must have at least one telephone number, but may have many.
To show this, the following multiplicity is set: for the address attribute the multiplicity is 1..2, for the phone number attribute the multiplicity is set to 1..{*}. This is shown in the diagram as follows:



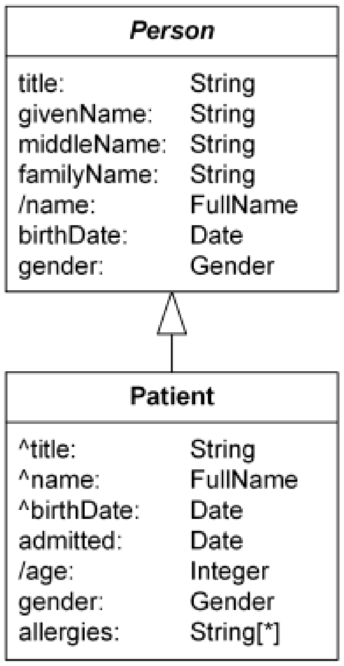
5. Generalization: It is also referred as „is-a" relationship. It is relationship between a class (super class) and one or more variations of the class (sub classes).It organizes classes by their similarities and differences, structuring the description of objects. The super class holds common attributes, operations and association. The subclasses add specific attributes, operations and associations. Each sub class inherits the features of its super class.

Notation: A large hollow arrowhead is used to show generalization. The arrowhead points towards the super class.

Example:-



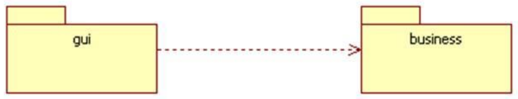
6.Inheritance: is usually explained in OOAD and in UML as some mechanism by which more specific classes (called subclasses or derived classes) incorporate structure and behavior of the more general classes (called superclasses, base classes, or parents). UML is inherently object-oriented modeling language and uses inheritance as one of its fundamental concepts while no UML specification provides appropriate details of how they actually define or interpret inheritance in UML.
UML 2.5 describes inherited members as some members of a parent classifier which were defined in the parent but behave as if they were defined in the inheriting classifier itself. An inherited member that is an attribute (or property in general) may have a value or collection of values in any instance of the inheriting classifier.
By default, the inherited members are members that do not have private visibility. In other words, all members of a parent are implicitly inherited except for those which have private visibility. This approach seemingly matches to inheritance definition in Java programming language but UML 2.5 specification provides no further details or explanations.
Until the UML 2.5 specification, inherited members had no specific notation. In UML 2.5 properties inherited by a classifier from a superclass may be shown on a diagram of the inheriting classifier by prepending a caret '^' symbol to the textual representation of the inherited property.
Example below shows Patient class with inherited attributes title, name, and birthDate with prepended caret '^' symbol. (See complete example at Hospital management domain UML diagram example.)

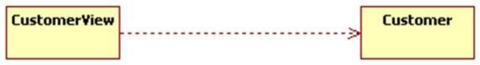


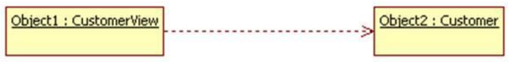
Patient class with inherited attributes
title, name, and birthDate.

<u>7.Dependency</u>

The most general relationship between two packages, classes, or objects is dependency, which is shown by a dashed arrow:







Strictly speaking, A depends on B is changes to B might necessitate changes to A. This is a bit stronger than UML dependency because it implies transitivity.
A better reading is A depends on B if A references B. This is a bit too weak because A might reference B in some implicit way.
Perhaps the simplest way around this is to say A depends on B is that A uses B.

Examples:

The gui package might contain a class such as CustomerView that depends on the Customer class, which belongs to the business package. We also say that gui imports from business or that business exports to gui.
The CustomerView class may have a method called display that expects a Customer object as input:

```
class CustomerView {
   void display(Customer c) { ... }
   ...
}
```

A CustomerView object may contain a pointer to a Customer object.

❖ Explain Different Component of UML:

1. What is Class?

A Class is a blueprint that is used to create Object. The Class defines what object can do.

What is Class Diagram?

UML CLASS DIAGRAM gives an overview of a software system by displaying classes, attributes, operations, and their relationships. This Diagram includes the class name, attributes, and operation in separate designated compartments.

Class Diagram defines the types of objects in the system and the different types of relationships that exist among them. It gives a high-level view of an application. This modeling method can run with almost all Object-Oriented Methods. A class can refer to another class. A class can have its objects or may inherit from other classes.

Class Diagram helps construct the code for the software application development.

Benefits of Class Diagram

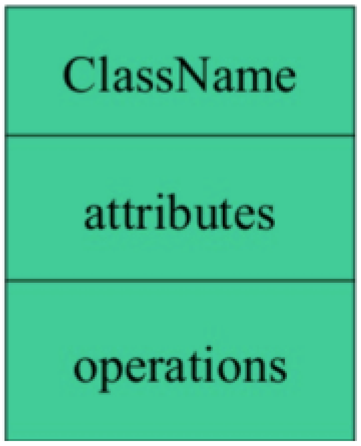
- Class Diagram Illustrates data models for even very complex information systems
- It provides an overview of how the application is structured before studying the actual code. This can easily reduce the maintenance time
- It helps for better understanding of general schematics of an application.
- Allows drawing detailed charts which highlights code required to be programmed
- Helpful for developers and other stakeholders.

Essential elements of A UML class diagram

Essential elements of UML class diagram are:

1. Class Name
2. Attributes
3. Operations

Class Name



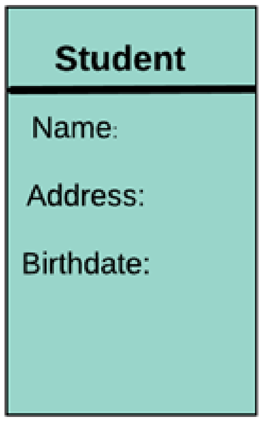
The name of the class is only needed in the graphical representation of the class. It appears in the topmost compartment. A class is the blueprint of an object which can share the same relationships, attributes, operations, & semantics. The class is rendered as a rectangle, including its name, attributes, and operations in sperate compartments.

Following rules must be taken care of while representing a class:

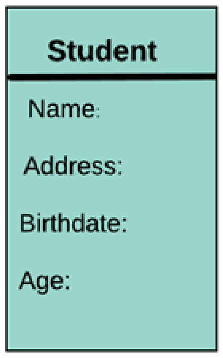
1. A class name should always start with a capital letter.
2. A class name should always be in the center of the first compartment.
3. A class name should always be written in **bold** format.
4. An abstract class name should be written in italics format.

Attributes:

An attribute is named property of a class which describes the object being modeled. In the class diagram, this component is placed just below the name-compartment.



A derived attribute is computed from other attributes. For example, an age of the student can be easily computed from his/her birth date.



Attributes characteristics

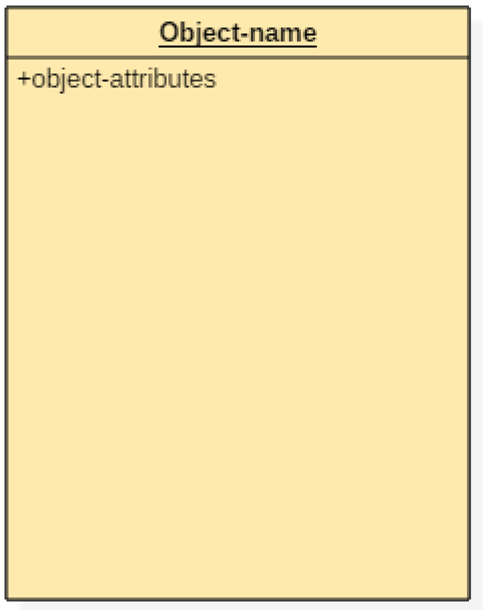
- The attributes are generally written along with the visibility factor.
- Public, private, protected and package are the four visibilities which are denoted by +, -, #, or ~ signs respectively.
- Visibility describes the accessibility of an attribute of a class.
- Attributes must have a meaningful name that describes the use of it in a class.

2.What is an Object Diagram?

Objects are the real-world entities whose behavior is defined by the classes. Objects are used to represent the static view of an object-oriented system. We cannot define an object without its class. Object and class diagrams are somewhat similar.

The difference between the class and object diagram is that the class diagram mainly represents the bird's eye view of a system which is also referred to as an abstract view. An object diagram describes the instance of a class. It visualizes the particular functionality of a system.

Notation of an object diagram:



Object Notation

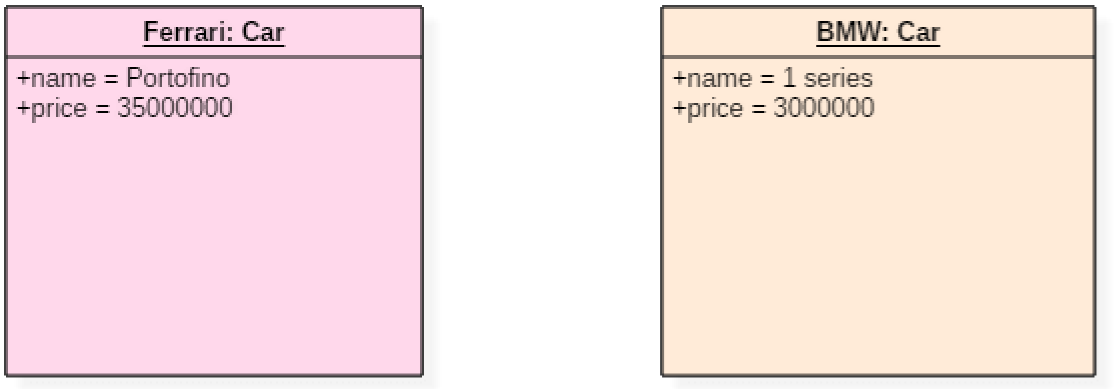
How to draw an object diagram?

1. Before drawing an object diagram, one should analyze all the objects inside the system.
2. The relations of the object must be known before creating the diagram.
3. Association between various objects must be cleared before.
4. An object should have a meaningful name that describes its functionality.
5. An object must be explored to analyze various functionalities of it.

Purpose of an object diagram:

1. It is used to describe the static aspect of a system.
2. It is used to represent an instance of a class.
3. It can be used to perform forward and reverse engineering on systems.
4. It is used to understand the behavior of an object.
5. It can be used to explore the relations of an object and can be used to analyze other connecting objects.

Object diagram example:



Object Diagram

The above UML object diagram contains two objects named Ferrari and BMW which belong to a class named as a Car. The objects are nothing but real-world entities that are the instances of a class.

Applications of Object Diagrams:

1. Object diagrams play an essential role while generating a blueprint of an object-oriented system.
2. Object diagrams provide means of modeling the classes, data and other information as a set or a single unit.
3. It is used for analyzing the online or offline system. The functioning of a system can be visualized using object diagrams.

3.What is the Use Case Diagram?

Use Case Diagram captures the system's functionality and requirements by using actors and use cases. Use Cases model the services, tasks, function that a system needs to perform. Use cases represent high-level functionalities and how a user will handle the system. Use-cases are the core concepts of Unified Modelling language modeling.

Why Use-Case diagram?

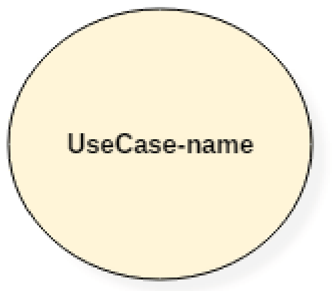
A Use Case consists of use cases, persons, or various things that are invoking the features called as actors and the elements that are responsible for implementing the use cases. Use case diagrams capture the dynamic behaviour of a live system. It models how an external entity interacts with the system to make it work. Use case diagrams are responsible for visualizing the external things that interact with the part of the system.

Use-case diagram notations

Following are the common notations used in a use case diagram:

Use-case:

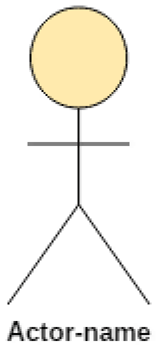
Use cases are used to represent high-level functionalities and how the user will handle the system. A use case represents a distinct functionality of a system, a component, a package, or a class. It is denoted by an oval shape with the name of a use case written inside the oval shape. The notation of a use case in UML is given below:



UML UseCase Notation

Actor:

It is used inside use case diagrams. The actor is an entity that interacts with the system. A user is the best example of an actor. An actor is an entity that initiates the use case from outside the scope of a use case. It can be any element that can trigger an interaction with the use case. One actor can be associated with multiple use cases in the system. The actor notation in UML is given below.



UML Actor Notation

How to draw a use-case diagram?

To draw a use case diagram in UML first one need to analyse the entire system carefully. You have to find out every single function that is provided by the system. After all the functionalities of a system are found out, then these functionalities are converted into various use cases which will be used in the use case diagram.

A use case is nothing but a core functionality of any working system. After organizing the use cases, we have to enlist the various actors or things that are going to interact with the system. These actors are responsible for invoking the functionality of a system. Actors can be a person or a thing. It can also be a private entity of a system. These actors must be relevant to the functionality or a system they are interacting with.

After the actors and use cases are enlisted, then you have to explore the relationship of a particular actor with the use case or a system. One must identify the total number of ways an actor could interact with the system. A single actor can interact with multiple use cases at the same time, or it can interact with numerous use cases simultaneously.

Following rules must be followed while drawing use-case for any system:

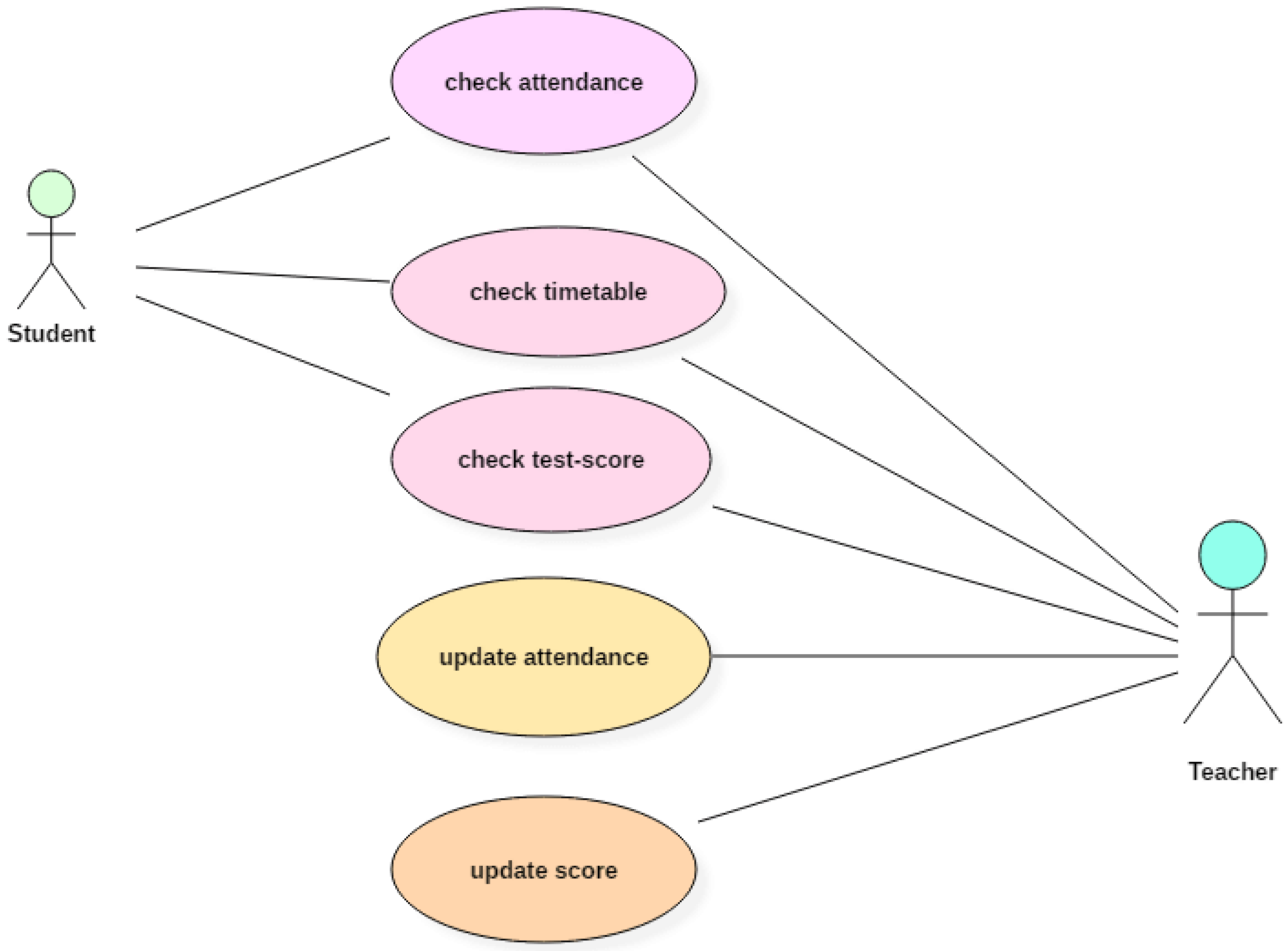
1. The name of an actor or a use case must be meaningful and relevant to the system.
2. Interaction of an actor with the use case must be defined clearly and in an understandable way.
3. Annotations must be used wherever they are required.
4. If a use case or an actor has multiple relationships, then only significant interactions must be displayed.

Tips for drawing a use-case diagram

1. A use case diagram should be as simple as possible.
2. A use case diagram should be complete.
3. A use case diagram should represent all interactions with the use case.
4. If there are too many use cases or actors, then only the essential use cases should be represented.
5. A use case diagram should describe at least a single module of a system.
6. If the use case diagram is large, then it should be generalized.

An example of a use-case diagram

Following use case diagram represents the working of the student management system:



UML UseCase Diagram

In the above use case diagram, there are two actors named student and a teacher. There are a total of five use cases that represent the specific functionality of a student management system. Each actor interacts with a particular use case. A student actor can check attendance, timetable as well as test marks on the application or a system. This actor can perform only these interactions with the system even though other use cases are remaining in the system.

It is not necessary that each actor should interact with all the use cases, but it can happen.

The second actor named teacher can interact with all the functionalities or use cases of the system. This actor can also update the attendance of a student and marks of the student. These interactions of both student and a teacher actor together sums up the entire student management application.

When to use a use-case diagram?

A use case is a unique functionality of a system which is accomplished by a user. A purpose of use case diagram is to capture core functionalities of a system and visualize the interactions of various things called as actors with the use case. This is the general use of a use case diagram.

The use case diagrams represent the core parts of a system and the workflow between them. In use case, implementation details are hidden from the external use only the event flow is represented.

With the help of use case diagrams, we can find out pre and post conditions after the interaction with the actor. These conditions can be determined using various test cases.

In general use case diagrams are used for:

1. Analyzing the requirements of a system
2. High-level visual software designing
3. Capturing the functionalities of a system
4. Modeling the basic idea behind the system
5. Forward and reverse engineering of a system using various test cases.

Use cases are intended to convey desired functionality so the exact scope of a use case may vary according to the system and the purpose of creating UML model.

4.What is an Activity Diagram in UML?

ACTIVITY DIAGRAM is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system. The basic purpose of activity diagrams is to capture the dynamic behavior of the system.. It is also called object-oriented flowchart.

This UML diagram focuses on the execution and flow of the behavior of a system instead of implementation. Activity diagrams consist of activities that are made up of actions that apply to behavioral modeling technology.

Components of Activity Diagram

Activities

It is a behavior that is divided into one or more actions. Activities are a network of nodes connected by edges. There can be action nodes, control nodes, or object nodes. Action nodes represent some action. Control nodes represent the control flow of an activity. Object nodes are used to describe objects used inside an activity. Edges are used to show a path or a flow of execution. Activities start at an initial node and terminate at a final node.

Activity partition/swimlane

An activity partition or a swimlane is a high-level grouping of a set of related actions. A single partition can refer to many things, such as classes, use cases, components, or interfaces.

If a partition cannot be shown clearly, then the name of a partition is written on top of the name of an activity.

Fork and Join nodes

Using a fork and join nodes, concurrent flows within an activity can be generated. A fork node has one incoming edge and numerous outgoing edges. It is similar to one too many decision parameters. When data arrives at an incoming edge, it is duplicated and split across numerous outgoing edges simultaneously. A single incoming flow is divided into multiple parallel flows.

A join node is opposite of a fork node as It has many incoming edges and a single outgoing edge. It performs logical AND operation on all the incoming edges. This helps you to synchronize the input flow across a single output edge.

Pins

An activity diagram that has a lot of flows gets very complicated and messy.

Pins are used to clearing up the things. It provides a way to manage the execution flow of activity by sorting all the flows and cleaning up messy thins. It is an object node that represents one input to or an output from an action.

Both input and output pins have precisely one edge.

Why use Activity Diagrams?

Activity diagram in UML allows you to create an event as an activity which contains a collection of nodes joined by edges. An activity can be attached to any modeling element to model its behavior. Activity diagrams are used to model,

- Use cases
- Classes
- Interfaces
- Components
- Collaborations

Activity diagrams are used to model processes and workflows. The essence of a useful activity diagram is focused on communicating a specific aspect of a system's dynamic behavior. Activity diagrams capture the dynamic elements of a system.

Activity diagram is similar to a flowchart that visualizes flow from one activity to another activity. Activity diagram is identical to the flowchart, but it is not a flowchart. The flow of activity can be controlled using various control elements in the UML flow diagram. In simple words, an activity diagram is used to activity diagrams that describe the flow of execution between multiple activities.

Activity Diagram Notations

Activity diagrams symbols can be generated by using the following notations:

- Initial states: The starting stage before an activity takes place is depicted as the initial state
- Final states: The state which the system reaches when a specific process ends is known as a Final State
- State or an activity box:
- Decision box: It is a diamond shape box which represents a decision with alternate paths. It represents the flow of control.

Activity Digram Notation and Symbol

How to draw an activity diagram?

Activity diagram is a flowchart of activities. It represents the workflow between various system activities. Activity diagrams are similar to the flowcharts, but they are not flowcharts. Activity diagram is an advancement of a flowchart that contains some unique capabilities.

Activity diagrams include swimlanes, branching, parallel flow, control nodes, expansion nodes, and object nodes. Activity diagram also supports exception handling.

To draw an activity diagram, one must understand and explore the entire system. All the elements and entities that are going to be used inside the diagram must be known by the user. The central concept which is nothing but an activity must be clear to the user. After analyzing all activities, these activities should be explored to find various constraints that are applied to activities. If there is such a constraint, then it should be noted before developing an activity diagram.

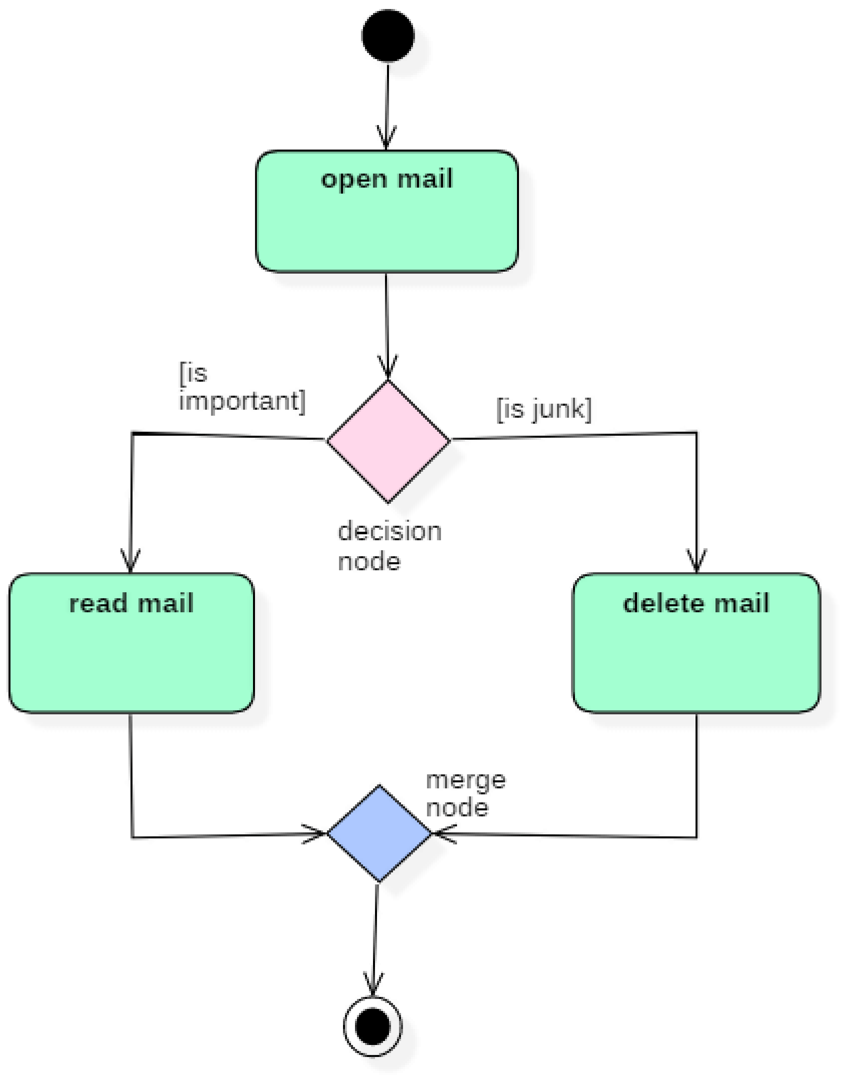
All the activities, conditions, and associations must be known. Once all the necessary things are gathered, then an abstract or a prototype is generated, which is later converted into the actual diagram.

Following rules must be followed while developing an activity diagram,

1. All activities in the system should be named.
2. Activity names should be meaningful.
3. Constraints must be identified.
4. Activity associations must be known.

Example of Activity Diagram

Let us consider mail processing activity as a sample for Activity Diagram. Following diagram represents activity for processing e-mails.



activity diagram

In the above activity diagram, three activities are specified. When the mail checking process begins user checks if mail is important or junk. Two guard conditions [is essential] and [is junk] decides the flow of execution of a process. After performing the activity, finally, the process is terminated at termination node.

When Use Activity Diagram

Activity diagram is used to model business processes and workflows. These diagrams are used in software modeling as well as business modeling.

Most commonly activity diagrams are used to,

1. Model the workflow in a graphical way, which is easily understandable.
2. Model the execution flow between various entities of a system.
3. Model the detailed information about any function or an algorithm which is used inside the system.
4. Model business processes and their workflows.
5. Capture the dynamic behavior of a system.
6. Generate high-level flowcharts to represent the workflow of any application.
7. Model high-level view of an object-oriented or a distributed system.