**What is**[**UML**](https://www.geeksforgeeks.org/unified-modeling-language-uml-introduction/)**?**
It is the general purpose modeling language used to visualize the system. It is a graphical language that is standard to the software industry for specifying, visualizing, constructing and documenting the artifacts of the software systems, as well as for business modeling.

**Benefits of UML:**

* Simplifies complex software design, can also implement OOPs like concept which is widely used.
* It reduces thousands of words of explanation in a few graphical diagrams that may reduce time consumption to understand.
* It makes communication more clear and real.
* It helps to acquire the entire system in a view.
* It becomes very much easy for the software programmer to implement the actual demand once they have the clear picture of the problem.

UML models basically three types of diagrams, namely, structure diagrams, interaction diagrams, and behavior diagrams. An activity diagram is a behavioral diagram i.e. it depicts the behavior of a system.

**Activity Diagram**.( <https://www.smartdraw.com/activity-diagram/>) helping link

1. We use Activity Diagrams to illustrate the flow of control in a system and refer to the steps involved in the execution of a use case. Primary use is to depict the dynamic aspects of a system.
2. Used to represent dynamic aspect of the system, basic nature of activity diagram is scenario specific which represents a set of activities performed in a particular scenario.
3. We can draw an activity diagram for a particular scenario or use case but cannot draw it for entire system.
4. It may have more than one final nodes based on different exit condition but it must have only one initial condition.
5. An activity diagram is very similar to a flowchart. So let us understand if an activity diagrams or a flowcharts are any different :

**Difference between an Activity diagram and a Flowchart –**

Flowcharts were typically invented earlier than activity diagrams. Non-programmers use Flow charts to model workflows. For example: A manufacturer uses a flow chart to explain and illustrate how a particular product is manufactured. We can call a flowchart a primitive version of an activity diagram. Business processes where decision making is involved is expressed using a flow chart.

So, programmers use activity diagrams (advanced version of a flowchart) to depict workflows. An activity diagram is **used by developers** to understand the flow of programs on a high level. It also enables them to figure out constraints and conditions that cause particular events. A flow chart converges into being an activity diagram if complex decisions are being made.

1. **Difference between a Use case diagram and an Activity diagram**

An activity diagram is used to model the workflow depicting conditions, constraints, sequential and concurrent activities. On the other hand, the purpose of a Use Case is to just depict the functionality i.e. what the system does and not how it is done. So in simple terms, an activity diagram shows ‘How’ while a Use case shows ‘What’ for a particular system.

## 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.

### Relationships

There are mainly three kinds of relationships in UML:

1. Dependencies
2. Generalizations
3. Associations

**Dependency**

A dependency means the relation between two or more classes in which a change in one may force changes in the other. However, it will always create a weaker relationship. Dependency indicates that one class depends on another.

In the following example, Student has a dependency on College



**Generalization:**



A generalization helps to connect a subclass to its superclass. A sub-class is inherited from its superclass. Generalization relationship can't be used to model interface implementation. Class diagram allows inheriting from multiple superclasses.

In this example, the class Student is generalized from Person Class.

**Association:**

This kind of relationship represents static relationships between classes A and B. For example; an employee works for an organization.

Here are some rules for Association:

* Association is mostly verb or a verb phrase or noun or noun phrase.
* It should be named to indicate the role played by the class attached at the end of the association path.
* Mandatory for reflexive associations

In this example, the relationship between student and college is shown which is studies.



**Multiplicity**



A multiplicity is a factor associated with an attribute. It specifies how many instances of attributes are created when a class is initialized. If a multiplicity is not specified, by default one is considered as a default multiplicity.

Let's say that that there are 100 students in one college. The college can have multiple students.

**Aggregation**

Aggregation is a special type of association that models a whole- part relationship between aggregate and its parts.



For example, the class college is made up of one or more student. In aggregation, the contained classes are never totally dependent on the lifecycle of the container. Here, the college class will remain even if the student is not available.

**Composition:**



The composition is a special type of aggregation which denotes strong ownership between two classes when one class is a part of another class.

For example, if college is composed of classes student. The college could contain many students, while each student belongs to only one college. So, if college is not functioning all the students also removed.

## Aggregation vs. Composition

|  |  |
| --- | --- |
| **Aggregation** | **Composition** |
| Aggregation indicates a relationship where the child can exist separately from their parent class. Example: Automobile (Parent) and Car (Child). So, If you delete the Automobile, the child Car still exist. | Composition display relationship where the child will never exist independent of the parent. Example: House (parent) and Room (child). Rooms will never separate into a House. |

## Abstract Classes

It is a class with an operation prototype, but not the implementation. It is also possible to have an abstract class with no operations declared inside of it. An abstract is useful for identifying the functionalities across the classes. Let us consider an example of an abstract class. Suppose we have an abstract class called as a motion with a method or an operation declared inside of it. The method declared inside the abstract class is called a **move ()**.

This abstract class method can be used by any object such as a car, an animal, robot, etc. for changing the current position. It is efficient to use this abstract class method with an object because no implementation is provided for the given function. We can use it in any way for multiple objects.

In UML, the abstract class has the same notation as that of the class. The only difference between a class and an abstract class is that the class name is strictly written in an italic font.

An abstract class cannot be initialized or instantiated.

Abstract Class Notation

In the above abstract class notation, there is the only a single abstract method which can be used by multiple objects of classes.

## Example of UML Class Diagram

Creating a class diagram is a straightforward process. It does not involve many technicalities. Here, is an example:

ATMs system is very simple as customers need to press some buttons to receive cash. However, there are multiple security layers that any ATM system needs to pass. This helps to prevent fraud and provide cash or need details to banking customers.

Below given is a UML Class Diagram example:



## Best practices of Designing of the Class Diagram

Class diagrams are the most important UML diagrams used for software application development. There are many properties which should be considered while drawing a Class Diagram. They represent various aspects of a software application.

Here, are some points which should be kept in mind while drawing a class diagram:

* The name given to the class diagram must be meaningful. Moreover, It should describe the real aspect of the system.
* The relationship between each element needs to be identified in advance.
* The responsibility for every class needs to be identified.
* For every class, minimum number of properties should be specified. Therefore, unwanted properties can easily make the diagram complicated.
* User notes should be included whenever you need to define some aspect of the diagram. At the end of the drawing, it must be understandable for the software development team.
* Lastly, before creating the final version, the diagram needs to be drawn on plain paper. Moreover, It should be reworked until it is ready for final submission.

### What is a Sequence Diagram?

Sequence diagrams, commonly used by developers, model the interactions between objects in a single use case. They illustrate how the different parts of a system interact with each other to carry out a function, and the order in which the interactions occur when a particular use case is executed.

In simpler words, a sequence diagram shows different parts of a system work in a ‘sequence’ to get something done.

### Sequence Diagram Notations

A sequence diagram is structured in such a way that it represents a timeline which begins at the top and descends gradually to mark the sequence of interactions. Each object has a column and the messages exchanged between them are represented by arrows.

**A Quick Overview of the Various Parts of a Sequence Diagram**

**Lifeline Notation**

A sequence diagram is made up of several of these lifeline notations that should be arranged horizontally across the top of the diagram. No two lifeline notations should overlap each other. They represent the different objects or parts that interact with each other in the system during the sequence.

A lifeline notation with an actor element symbol is used when the particular sequence diagram is owned by a use case.



A lifeline with an entity element represents system data. For example, in a customer service application, the Customer entity would manage all data related to a customer.



A lifeline with a boundary element indicates a system boundary/ software element in a system; for example, user interface screens, database gateways or menus that users interact with, are boundaries.



And a lifeline with a control element indicates a controlling entity or manager. It organizes and schedules the interactions between the boundaries and entities and serves as the mediator between them.



**Activation Bars**

Activation bar is the box placed on the lifeline.  It is used to indicate that an object is active (or instantiated) during an interaction between two objects. The length of the rectangle indicates the duration of the objects staying active.

In a sequence diagram, an interaction between two objects occurs when one object sends a message to another. The use of the activation bar on the lifelines of the Message Caller (the object that sends the message) and the Message Receiver (the object that receives the message) indicates that both are active/is instantiated during the exchange of the message.



**Message Arrows**

An arrow from the Message Caller to the Message Receiver specifies a message in a sequence diagram.   A message can flow in any direction; from left to right, right to left or back to the Message Caller itself. While you can describe the message being sent from one object to the other on the arrow, with different arrowheads you can indicate the type of message being sent or received.

The message arrow comes with a description, which is known as a message signature, on it. The format for this message signature is below. All parts except the message\_name are optional.

*attribute = message\_name (arguments): return\_type*

* *Synchronous message*

As shown in the activation bars example, a synchronous message is used when the sender waits for the receiver to process the message and return before carrying on with another message.  The arrowhead used to indicate this type of message is a solid one, like the one below.



* *Asynchronous message*

An asynchronous message is used when the message caller does not wait for the receiver to process the message and return before sending other messages to other objects within the system. The arrowhead used to show this type of message is a line arrow like shown in the example below.



* *Return message*

A return message is used to indicate that the message receiver is done processing the message and is returning control over to the message caller. Return messages are optional notation pieces, for an activation bar that is triggered by a synchronous message always implies a return message.

Tip: You can avoid cluttering up your diagrams by minimizing the use of return messages since the return value can be specified in the initial message arrow itself.


* *Participant  creation message*

Objects do not necessarily live for the entire duration of the sequence of events. Objects or participants can be created according to the message that is being sent.

The dropped participant box notation can be used when you need to show that the particular participant did not exist until the create call was sent.  If the created participant does something immediately after its creation, you should add an activation box right below the participant box.



* Participant destruction message

Likewise, participants when no longer needed can also be deleted from a sequence diagram. This is done by adding an ‘X’ at the end of the lifeline of the said participant.

* *Reflexive message*

When an object sends a message to itself, it is called a reflexive message. It is indicated with a message arrow that starts and ends at the same lifeline as shown in the example below.



**Comment**

[UML diagrams](https://creately.com/lp/uml-diagram-tool/) generally permit the annotation of comments in all [UML diagram types](https://creately.com/blog/diagrams/uml-diagram-types-examples/). The comment object is a rectangle with a folded-over corner as shown below. The comment can be linked to the related object with a dashed line.

