# UML - Overview

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

UML was created by Object Management Group (OMG) and UML 1.0 specification draft was proposed to the OMG in January 1997.

OMG is continuously putting effort to make a truly industry standard.

* UML stands for **U**nified **M**odeling **L**anguage.
* UML is different from the other common programming languages like C++, Java, and COBOL etc.
* UML is a pictorial language used to make software blue prints.

So 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 but it is not limited within this boundary. It is also used to model non software systems as well like 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 is become an OMG (Object Management Group) standard.

**Goals of UML:**

*A picture is worth a thousand words*, this absolutely fits while discussing about UML. Object oriented concepts were introduced much earlier than UML. So at that time there were no standard methodologies to organize and consolidate the object oriented development. At that point of time 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 also it 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. So it must be clear that UML is not a development method rather it accompanies with processes to make a successful system.

At the 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 conceptual model of UML first we need to clarify *what is a conceptual model?* And *why a conceptual model is at all 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. 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

**Object oriented concepts:**

UML can be described as the successor of object oriented analysis and design.

An object contains both data and methods that control the data. The data represents the state of the object. A class describes an object and they also form hierarchy to model real world system. The hierarchy is represented as inheritance and the classes can also be associated in different manners as per the requirement.

The objects are the real world entities that exist around us and the basic concepts like abstraction, encapsulation, inheritance, polymorphism all can be represented using UML.

So UML is powerful enough to represent all the concepts exists in object oriented analysis and design. UML diagrams are representation of object oriented concepts only. So before learning UML, it becomes important to understand OO concepts in details.

Following are some fundamental concepts of object oriented world:

* **Objects:** Objects represent an entity and the basic building block.
* **Class:** Class is the blue print of an object.
* **Abstraction:** Abstraction represents the behavior of a real world entity.
* **Encapsulation:** Encapsulation is the mechanism of binding the data together and hiding them from outside world.
* **Inheritance:** Inheritance is the mechanism of making new classes from existing one.
* **Polymorphism:** It defines the mechanism to exist in different forms.

**OO Analysis and Design**

Object oriented analysis can be defined as investigation and to be more specific it is the investigation of objects. Design means collaboration of identified objects.

So it is important to understand the OO analysis and design concepts. Now the most important purpose of OO analysis is to identify objects of a system to be designed. This analysis is also done for an existing system. Now an efficient analysis is only possible when we are able to start thinking in a way where objects can be identified. After identifying the objects their relationships are identified and finally the design is produced.

So the purpose of OO analysis and design can described as:

* Identifying the objects of a system.
* Identify their relationships.
* Make a design which can be converted to executables using OO languages.

There are three basic steps where the OO concepts are applied and implemented. The steps can be defined as

OO Analysis --> OO Design --> OO implementation using OO languages

Now the above three points can be described in details:

* During object oriented analysis the most important purpose is to identify objects and describing them in a proper way. If these objects are identified efficiently then the next job of design is easy. The objects should be identified with responsibilities. Responsibilities are the functions performed by the object. Each and every object has some type of responsibilities to be performed. When these responsibilities are collaborated the purpose of the system is fulfilled.
* The second phase is object oriented design. During this phase emphasis is given upon the requirements and their fulfilment. In this stage the objects are collaborated according to their intended association. After the association is complete the design is also complete.
* The third phase is object oriented implementation. In this phase the design is implemented using object oriented languages like Java, C++ etc.

**Role of UML in OO design:**

UML is a modeling language used to model software and non-software systems. Although UML is used for non software systems the emphasis is on modeling object oriented software applications. Most of the UML diagrams discussed so far are used to model different aspects like static, dynamic etc. Now what ever be the aspect the artifacts are nothing but objects.

If we look into class diagram, object diagram, collaboration diagram, interaction diagrams all would basically be designed based on the objects.

So the relation between OO design and UML is very important to understand. The OO design is transformed into UML diagrams according to the requirement. Before understanding the UML in details the OO concepts should be learned properly. Once the OO analysis and design is done the next step is very easy. The input from the OO analysis and design is the input to the UML diagrams.

# UML Building Blocks

As UML describes the real time systems it is very important to make a conceptual model and then proceed gradually. 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

This chapter describes all the UML building blocks. The building blocks of UML can be defined as: (TRD)

* Things
* Relationships
* Diagrams

## (1) Things:

**Things** are the most important building blocks of UML. Things can be

(S=Subrata BoGrA)

* Structural
* Behavioral
* Grouping
* Annotational( টীকা)

### Structural things:

The **Structural things** define the static part of the model. They represent physical and conceptual elements. Following are the brief descriptions of the structural things.

**Class, interface, Collaboration, use case, component, Node.**

**Class:**

Class represents set of objects having similar responsibilities.

class

**Interface:**

Interface defines a set of operations which specify the responsibility of a class.

Interface

**Collaboration:**

Collaboration defines interaction between elements.

Collaboration

**Use case:**

Use case represents a set of actions performed by a system for a specific goal.

Use case

**Component:**

Component describes physical part of a system.

Component

**Node:**

A node can be defined as a physical element that exists at run time.



### Behavioral things:

**A behavioral thing** consists of the dynamic parts of UML models. Following are the behavioral things:

**Interaction:**

Interaction is defined as a behavior that consists of a group of messages exchanged among elements to accomplish a specific task.

Interaction

**State machine:**

State machine is useful when the state of an object in its life cycle is important. It defines the sequence of states an object goes through in response to events. Events are external factors responsible for state change.



### Grouping things:

**Grouping things** can be defined as a mechanism to group elements of a UML model together. There is only one grouping thing available:

**Package:**

Package is the only one grouping thing available for gathering structural and behavioral things.



### Annotational things:

**Annotational things** can be defined as a mechanism to capture remarks, descriptions, and comments of UML model elements. **Note** is the only one Annotational thing available.

## Note:

* A note is used to render comments, constraints etc. of an UML element.

Note

## (2) Relationship:

**Relationship** is another most important building block of UML. It shows how elements are associated with each other and this association describes the functionality of an application.

* There are four kinds of relationships available. (RGDA)

### Dependency:

Dependency is a relationship between two things in which change in one element also affects the other one.

Dependency

### Association:

Association is basically a set of links that connects elements of an UML model. It also describes how many objects are taking part in that relationship.

Association

### Generalization:

Generalization can be defined as a relationship which connects a specialized element with a generalized element. It basically describes inheritance relationship in the world of objects.

Generalization

### Realization:

Realization can be defined as a relationship in which two elements are connected. One element describes some responsibility which is not implemented and the other one implements them. This relationship exists in case of interfaces.

Realization

## (3) UML Diagrams:

UML diagrams are the ultimate output of the entire discussion. All the elements, relationships are used to make a complete UML diagram and the diagram represents a system.

The visual effect of the UML diagram is the most important part of the entire process. All the other elements are used to make it a complete one.

UML includes the following nine diagrams and the details are described in the following chapters.

* Class diagram
* Object diagram
* Use case diagram
* Sequence diagram
* Collaboration diagram
* Activity diagram
* State chart diagram
* Deployment diagram
* Component diagram

We would discuss all these diagrams in subsequent chapters of this tutorial.

# UML - Basic Notations

UML is popular for its diagrammatic notations. We all know that UML is for visualizing, specifying, constructing and documenting the components of software and non-software systems. Here the *Visualization* is the most important part which needs to be understood and remembered by heart.

UML notations are the most important elements in modeling. Efficient and appropriate use of notations is very important for making a complete and meaningful model. The model is useless unless its purpose is depicted properly.

So learning notations should be emphasized from the very beginning. Different notations are available for things and relationships. And the UML diagrams are made using the notations of things and relationships. Extensibility is another important feature which makes UML more powerful and flexible.

The chapter describes the UML Basic Notations in more details. This is just an extension to the UML building block section I have discussed in previous chapter.

## Structural Things:

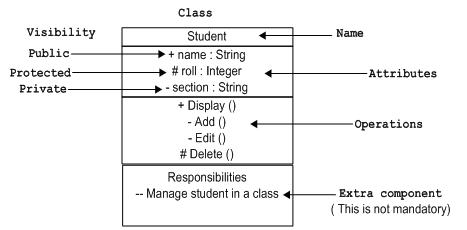
Graphical notations used in structural things are the most widely used in UML. These are considered as the nouns of UML models. Following are the list of structural things.

* Classes
* object
* Interface
* Collaboration
* Use case
* Active classes
* Components
* Nodes

### Class Notation:

UML *class* is represented by the diagram shown below. The diagram is divided into four parts.

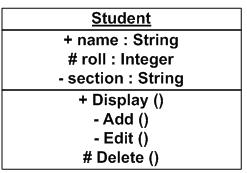
* The top section is used to name the class.
* The second one is used to show the attributes of the class.
* The third section is used to describe the operations performed by the class.
* The fourth section is optional to show any additional components.



Classes are used to represent objects. Objects can be anything having properties and responsibility.

### Object Notation:

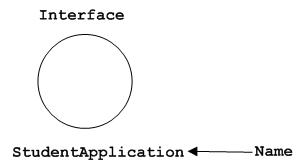
The *object* is represented in the same way as the class. The only difference is the *name* which is underlined as shown below.



As object is the actual implementation of a class which is known as the instance of a class. So it has the same usage as the class.

### Interface Notation:

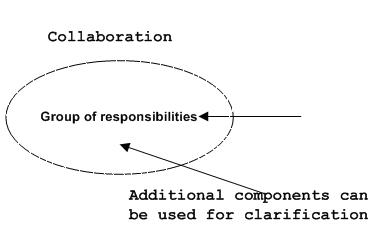
Interface is represented by a circle as shown below. It has a name which is generally written below the circle.



Interface is used to describe functionality without implementation. Interface is the just like a template where you define different functions not the implementation. When a class implements the interface it also implements the functionality as per the requirement.

### Collaboration Notation:

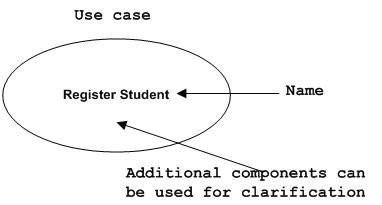
Collaboration is represented by a dotted eclipse as shown below. It has a name written inside the eclipse.



Collaboration represents responsibilities. Generally responsibilities are in a group.

### Use case Notation:

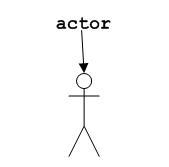
Use case is represented as an eclipse with a name inside it. It may contain additional responsibilities.



Use case is used to capture high level functionalities of a system.

### Actor Notation:

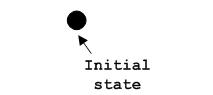
An actor can be defined as some internal or external entity that interacts with the system.



Actor is used in a use case diagram to describe the internal or external entities.

### Initial State Notation:

Initial state is defined to show the start of a process. This notation is used in almost all diagrams.



The usage of Initial State Notation is to show the starting point of a process.

### Final State Notation:

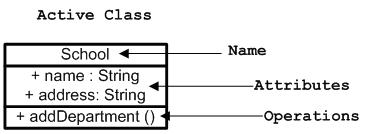
Final state is used to show the end of a process. This notation is also used in almost all diagrams to describe the end.



The usage of Final State Notation is to show the termination point of a process.

### Active class Notation:

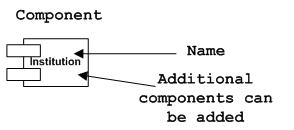
Active class looks similar to a class with a solid border. Active class is generally used to describe concurrent behavior of a system.



Active class is used to represent concurrency in a system.

### Component Notation:

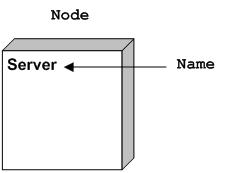
A component in UML is shown as below with a name inside. Additional elements can be added wherever required.



Component is used to represent any part of a system for which UML diagrams are made.

### Node Notation:

A node in UML is represented by a square box as shown below with a name. A node represents a physical component of the system.



Node is used to represent physical part of a system like server, network etc.

## Behavioral Things:

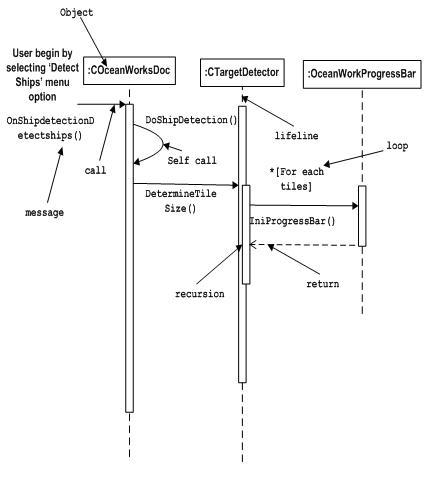
Dynamic parts are one of the most important elements in UML. UML has a set of powerful features to represent the dynamic part of software and non-software systems. These features include *interactions* and *state machines*.

Interactions can be of two types:

* Sequential (Represented by sequence diagram)
* Collaborative (Represented by collaboration diagram)

### Interaction Notation:

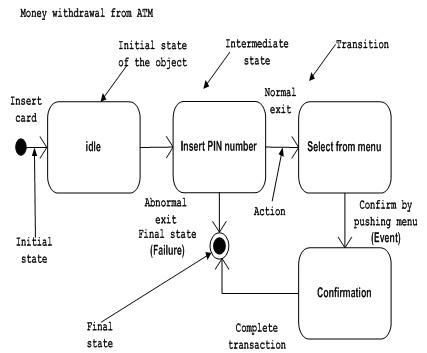
Interaction is basically message exchange between two UML components. The following diagram represents different notations used in an interaction.



Interaction is used to represent communication among the components of a system.

### State machine Notation:

State machine describes the different states of a component in its life cycle. The notations are described in the following diagram.



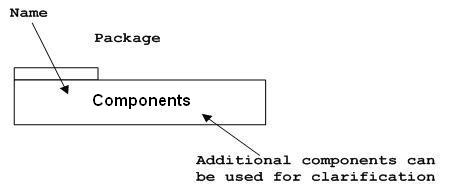
State machine is used to describe different states of a system component. The state can be active, idle or any other depending upon the situation.

## Grouping Things:

Organizing the UML models are one of the most important aspects of the design. In UML there is only one element available for grouping and that is package.

### Package Notation:

Package notation is shown below and this is used to wrap the components of a system.

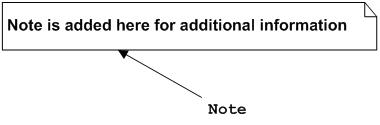


## Annotational Things:

In any diagram explanation of different elements and their functionalities are very important. So UML has *notes* notation to support this requirement.

### Note Notation:

This notation is shown below and they are used to provide necessary information of a system.



## Relationships

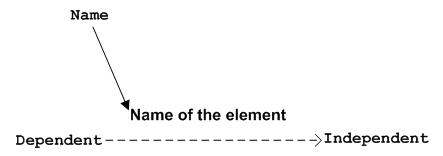
A model is not complete unless the relationships between elements are described properly. The *Relationship* gives a proper meaning to an UML model. Following are the different types of relationships available in UML.

* Dependency
* Association
* Generalization
* Extensibility

### Dependency Notation:

Dependency is an important aspect in UML elements. It describes the dependent elements and the direction of dependency.

Dependency is represented by a dotted arrow as shown below. The arrow head represents the independent element and the other end the dependent element.

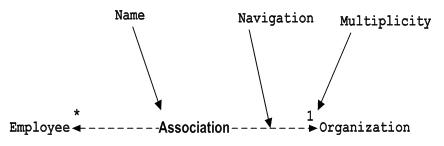


Dependency is used to represent dependency between two elements of a system.

### Association Notation:

Association describes how the elements in an UML diagram are associated. In simple word it describes how many elements are taking part in an interaction.

Association is represented by a dotted line with (without) arrows on both sides. The two ends represent two associated elements as shown below. The multiplicity is also mentioned at the ends (1, \* etc) to show how many objects are associated.



Association is used to represent the relationship between two elements of a system.

### Generalization Notation:

Generalization describes the inheritance relationship of the object oriented world. It is parent and child relationship.

Generalization is represented by an arrow with hollow arrow head as shown below. One end represents the parent element and the other end child element.

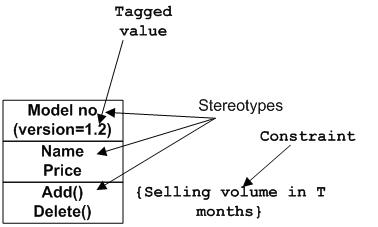
Generalization Notation

Generalization is used to describe parent-child relationship of two elements of a system.

### Extensibility Notation:

All the languages (programming or modeling) have some mechanism to extend its capabilities like syntax, semantics etc. UML is also having the following mechanisms to provide extensibility features.

* Stereotypes (Represents new elements)
* Tagged values (Represents new attributes)
* Constraints (Represents the boundaries)



Extensibility notations are used to enhance the power of the language. It is basically additional elements used to represent some extra behaviour of the system. These extra behaviours are not covered by the standard available notations.

# UML - Class Diagram

## Overview:

The class diagram is a static diagram. It represents the static view of an application. Class diagram is not only used for visualizing, describing and documenting different aspects of a system but also for constructing executable code of the software application.

The class diagram describes the attributes and operations of a class and also the constraints imposed on the system. The class diagrams are widely used in the modelling of object oriented systems because they are the only UML diagrams which can be mapped directly with object oriented languages.

***The class diagram shows a collection of classes, interfaces, associations, collaborations and constraints. It is also known as a structural diagram.***

## Purpose:

The purpose of the class diagram is to model the static view of an application. The class diagrams are the only diagrams which can be directly mapped with object oriented languages and thus widely used at the time of construction.

The UML diagrams like activity diagram, sequence diagram can only give the sequence flow of the application but class diagram is a bit different. So it is the most popular UML diagram in the coder community.

So the purpose of the class diagram can be summarized as:

* Analysis and design of the static view of an application.
* Describe responsibilities of a system.
* Base for component and deployment (প্রয়োগ করা) diagrams.
* Forward and reverse engineering.

## How to draw Class Diagram?

Class diagrams are the most popular UML diagrams used for construction of software applications. So it is very important to learn the drawing procedure of class diagram.

Class diagrams have lot of properties to consider while drawing but here the diagram will be considered from a top level view.

Class diagram is basically a graphical representation of the static view of the system and represents different aspects of the application. So a collection of class diagrams represent the whole system.

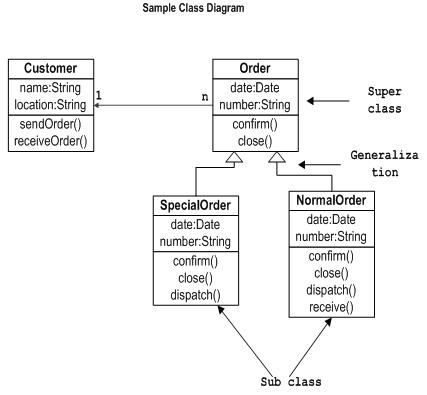
The following points should be remembered while drawing a class diagram:

* The name of the class diagram should be meaningful to describe the aspect of the system.
* Each element and their relationships should be identified in advance.
* Responsibility (attributes and methods) of each class should be clearly identified.
* For each class minimum number of properties should be specified. Because unnecessary properties will make the diagram complicated.
* Use notes whenever required to describe some aspect of the diagram. Because at the end of the drawing it should be understandable to the developer/coder.
* Finally, before making the final version, the diagram should be drawn on plain paper and rework as many times as possible to make it correct.

Now the following diagram is an example of an *Order System* of an application. So it describes a particular aspect of the entire application.

* First of all *Order* and *Customer* are identified as the two elements of the system and they have a *one to many* relationship because a customer can have multiple orders.
* We would keep *Order* class is an abstract class and it has two concrete classes (inheritance relationship) *SpecialOrder* and *NormalOrder*.
* The two inherited classes have all the properties as the *Order* class. In addition they have additional functions like *dispatch ()* and *receive ()*.

So the following class diagram has been drawn considering all the points mentioned above:



## Where to use Class Diagrams?

Class diagram is a static diagram and it is used to model static view of a system. The static view describes the vocabulary of the system.

Class diagram is also considered as the foundation for component and deployment diagrams. Class diagrams are not only used to visualize the static view of the system but they are also used to construct the executable code for forward and reverse engineering of any system.

Generally UML diagrams are not directly mapped with any object oriented programming languages but the class diagram is an exception.

Class diagram clearly shows the mapping with object oriented languages like Java, C++ etc. So from practical experience class diagram is generally used for construction purpose.

So in a brief, class diagrams are used for:

* Describing the static view of the system.
* Showing the collaboration among the elements of the static view.
* Describing the functionalities performed by the system.
* Construction of software applications using object oriented languages.

# UML - Use Case Diagrams

## Overview:

*Use Case Diagrams is a dynamic behaviour means the behaviour of the system when it is running /operating.*

To model a system the most important aspect is to capture the dynamic behaviour. To clarify a bit in details, *dynamic behaviour* means the behaviour of the system when it is running /operating.

So only static behaviour is not sufficient to model a system rather dynamic behaviour is more important than static behaviour. In UML there are five diagrams available to model dynamic nature and use case diagram is one of them. Now as we have to discuss that the use case diagram is dynamic in nature there should be some internal or external factors for making the interaction.

These internal and external agents are known as actors. So use case diagrams are consists of actors, use cases and their relationships. The diagram is used to model the system/subsystem of an application. A single use case diagram captures a particular functionality of a system.

So to model the entire system numbers of use case diagrams are used.

## Purpose:

The purpose of use case diagram is to capture the dynamic aspect of a system. But this definition is too generic to describe the purpose.

Because other four diagrams (activity, sequence, collaboration and State chart) are also having the same purpose. So we will look into some specific purpose which will distinguish it from other four diagrams.

Use case diagrams are used to gather the requirements of a system including internal and external influences. These requirements are mostly design requirements. So when a system is analyzed to gather its functionalities use cases are prepared and actors are identified.

Now when the initial task is complete use case diagrams are modelled to present the outside view.

So in brief, the purposes of use case diagrams can be as follows:

* Used to gather requirements of a system.
* Used to get an outside view of a system.
* Identify external and internal factors influencing the system.
* Show the interacting among the requirements are actors.

## How to draw Use Case Diagram?

Use case diagrams are considered for high level requirement analysis of a system. So when the requirements of a system are analyzed the functionalities are captured in use cases.

So we can say that uses cases are nothing but the system functionalities written in an organized manner. Now the second things which are relevant to the use cases are the actors. Actors can be defined as something that interacts with the system.

The actors can be human user, some internal applications or may be some external applications. So in a brief when we are planning to draw a use case diagram we should have the following items identified.

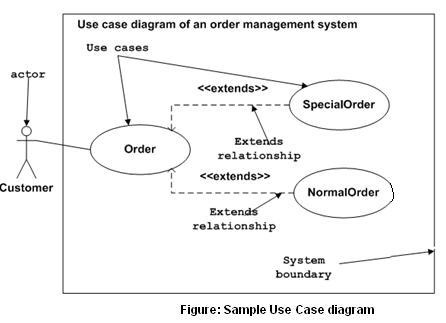
* Functionalities to be represented as an use case
* Actors
* Relationships among the use cases and actors.

Use case diagrams are drawn to capture the functional requirements of a system. So after identifying the above items we have to follow the following guidelines to draw an efficient use case diagram.

* The name of a use case is very important. So the name should be chosen in such a way so that it can identify the functionalities performed.
* Give a suitable name for actors.
* Show relationships and dependencies clearly in the diagram.
* Do not try to include all types of relationships. Because the main purpose of the diagram is to identify requirements.
* Use note whenever required to clarify note some important points.

The following is a sample use case diagram representing the order management system. So if we look into the diagram then we will find three use cases (Order, SpecialOrder and NormalOrder) and one actor which is customer.

The *SpecialOrder* and *NormalOrder* use cases are extended from *Order* use case. So they have extends relationship. Another important point is to identify the system boundary which is shown in the picture. The actor *Customer* lies outside the system as it is an external user of the system.



## Where to Use Case Diagrams?

As we have already discussed there are five diagrams in UML to model dynamic view of a system. Now each and every model has some specific purpose to use. Actually these specific purposes are different angles of a running system.

So to understand the dynamics of a system we need to use different types of diagrams. Use case diagram is one of them and its specific purpose is to gather system requirements and actors.

Use case diagrams specify the events of a system and their flows. But use case diagram never describes how they are implemented. Use case diagram can be imagined as a black box where only the input, output and the function of the black box is known.

These diagrams are used **at a very high level of design**. Then this high level design is refined again and again to get a complete and practical picture of the system. A well-structured use case also describes the pre-condition, post condition, exceptions. And these extra elements are used to make test cases when performing the testing.

Although the use cases are not a good candidate for forward and reverse engineering but still they are used in a slight different way to make forward and reverse engineering. And the same is true for reverse engineering. Still use case diagram is used differently to make it a candidate for reverse engineering.

In-forward engineering use case diagrams are used to make test cases and in reverse engineering use cases are used to prepare the requirement details from the existing application.

**So the following are the places where use case diagrams are used:**

* **Requirement analysis and high level design.**
* **Model the context of a system.**
* **Reverse engineering.**
* **Forward engineering.**