UNIT-3

Design Using UML

What is UML

The UML stands for Unified modeling language, is a standardized general-purpose visual modeling language in the field of Software Engineering. It is used for specifying, visualizing, constructing, and documenting the primary artifacts of the software system. It helps in designing and characterizing, especially those software systems that incorporate the concept of Object orientation. It describes the working of both the software and hardware systems.

Characteristics of UML

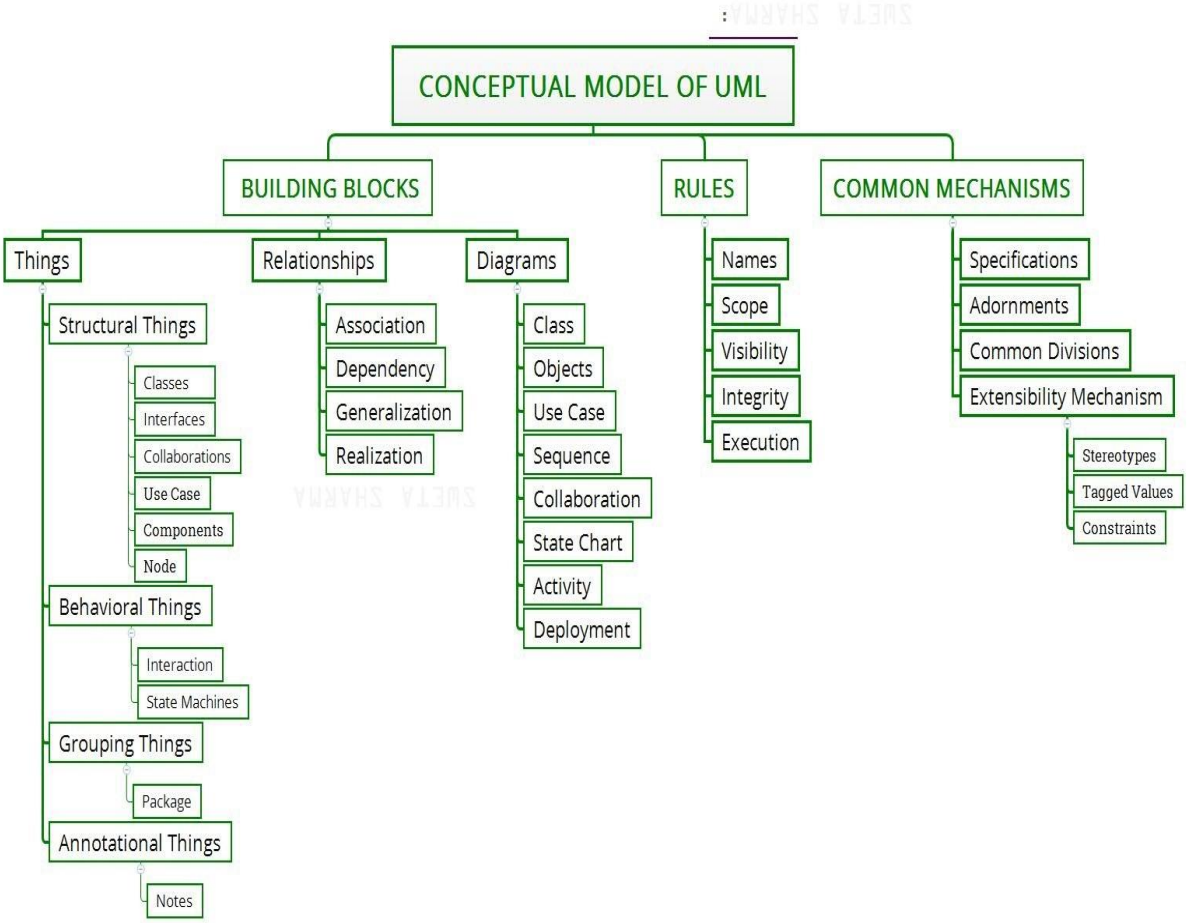
The UML has the following features:

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

Conceptual Modeling

Before moving ahead with the concept of UML, we should first understand the basics of the conceptual model.

A conceptual model is composed of several interrelated concepts. It makes it easy to understand the objects and how they interact with each other. This is the first step before drawing UML diagrams.



The conceptual model of UML contains the fundamentals of UML. The conceptual model consists of three parts. They are:

1) Building blocks of UML (syntax / vocabulary)

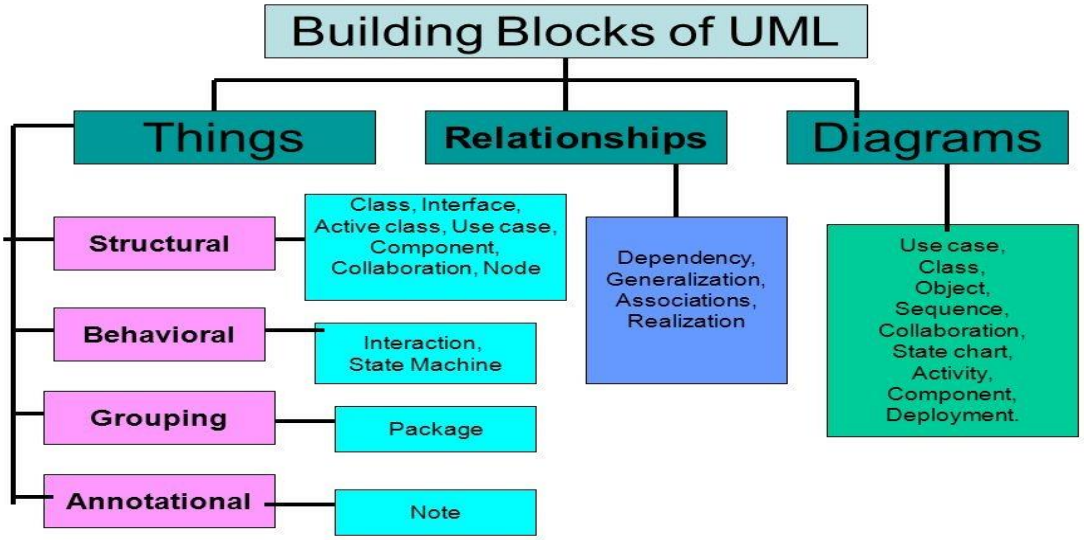
2) Rules (semantics)

3) Common Mechanisms

Building Blocks of UML

The building blocks of UML can be defined as –

- Things
- Relationships
- Diagrams



Things

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

- Structural
- Behavioral
- Grouping
- Annotational

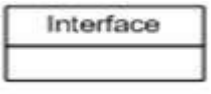
Structural Things

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

Class − Class represents a set of objects having similar responsibilities.

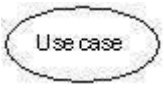
Interface − Interface defines a set of operations, which specify the responsibility of a class.

OR

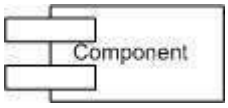


Collaboration −Collaboration defines an interaction between elements.

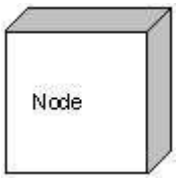
Use case −Use case represents a set of actions performed by a system for a specific goal.



Component −Component describes the physical part of a system.



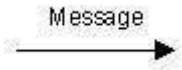
Node − A node can be defined as a physical element that exists at run time.



Behavioral Things

A behavioral thing consists of the dynamic parts of UML models. Following are the behavioural things −

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



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

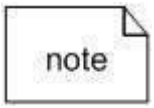
Grouping things can be defined as a mechanism to group elements of a UML model together. There is only one grouping thing available −

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 - It is the only one Annotational thing available. A note is used to render comments, constraints, etc. of an UML element.



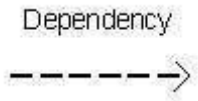
Relationship

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

There are four kinds of relationships available.

Dependency

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

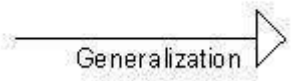


Association

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

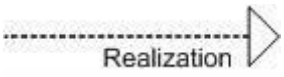
Generalization

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



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.



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 complete.

UML includes the following nine diagrams, the details of which are described in the subsequent chapters.

- Class diagram
- Object diagram
- Use case diagram
- Sequence diagram
- Collaboration diagram
- Activity diagram
- Statechart diagram
- Deployment diagram
- Component diagram

Rules of UML

The rules of UML specify how the UMLs building blocks come together to develop diagrams. The rules enable the users to create well-formed models. A well-formed model is self-consistent and also consistent with the other models.

UML has rules for:

Names – What elements can be called as things, relationships and diagrams

Scope – The context that gives a specific meaning to a name

Visibility – How these names are seen and can be used by the other names

Integrity – How things properly relate to one another

Execution – What it means to run or simulate a model

Common Mechanisms in UML

Why UML is easy to learn and use? It's because of the four common mechanisms that apply throughout the UML. They are:

1. Specifications
2. Adornments
3. Common divisions
4. Extensibility mechanisms

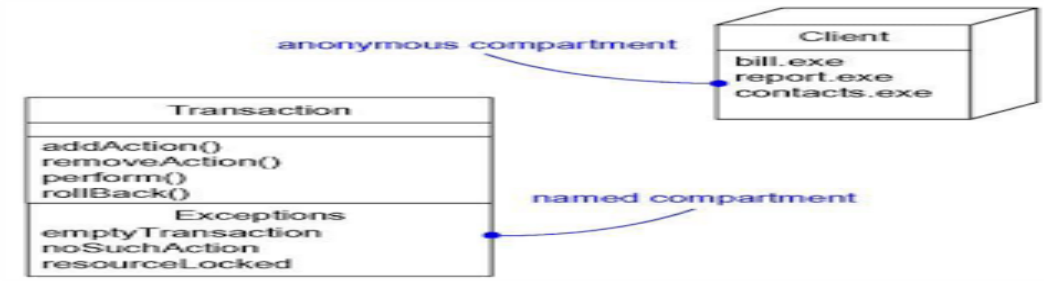
Specifications:

Behind every graphical notation in UML there is a precise specification of the details that element represents. For example, a class icon is a rectangle and it specifies the name, attributes and operations of the class.

Adornments:

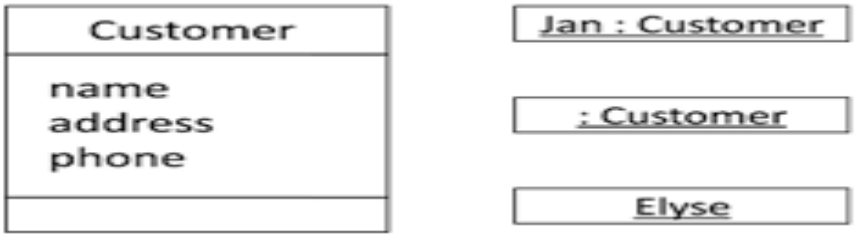
The mechanism in UML which allows the users to specify extra information with the basic notation of an element is the adornments.

In the above example, the access specifiers: + (public), # (protected) and – (private) represent the visibility of the attributes which is extra information over the basic attribute representation.



Common Divisions

In UML there is clear division between semantically related elements like: separation between a class and an object and the separation between an interface and its implementation.



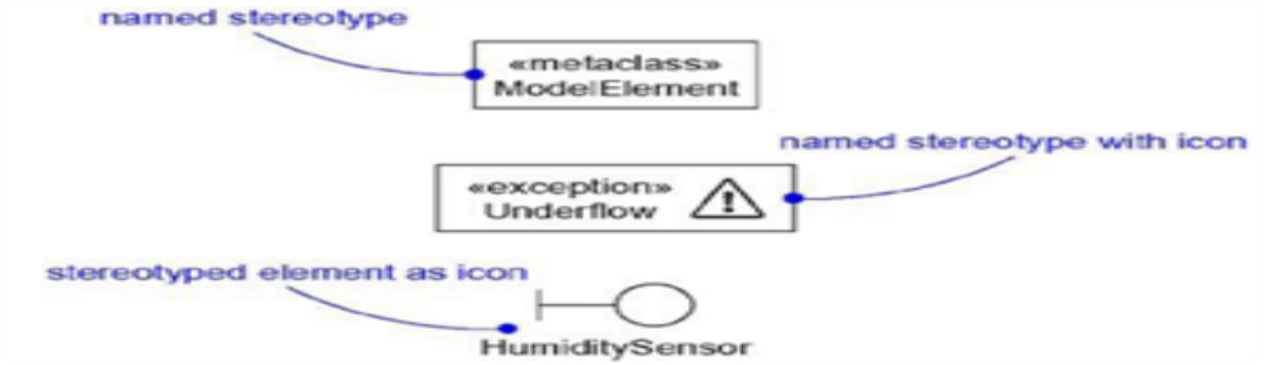
Extensibility Mechanisms

UMLs extensibility mechanisms allow the user to extend (new additions) the language in a controlled way. The extensibility mechanisms in UML are:

- Stereotypes
- Tagged Values
- Constraints

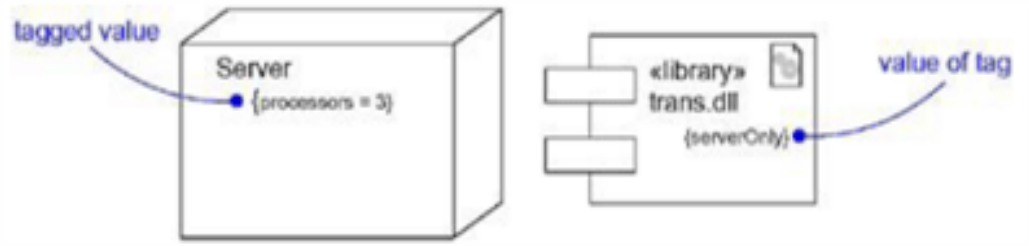
Stereotypes

Stereotypes allows us to create new building blocks. Using stereotypes we can use the basic elements but with special properties, semantics and notation. Stereotypes are rendered as text inside guillemets(<< >>) or we can create new icons for stereotypes.



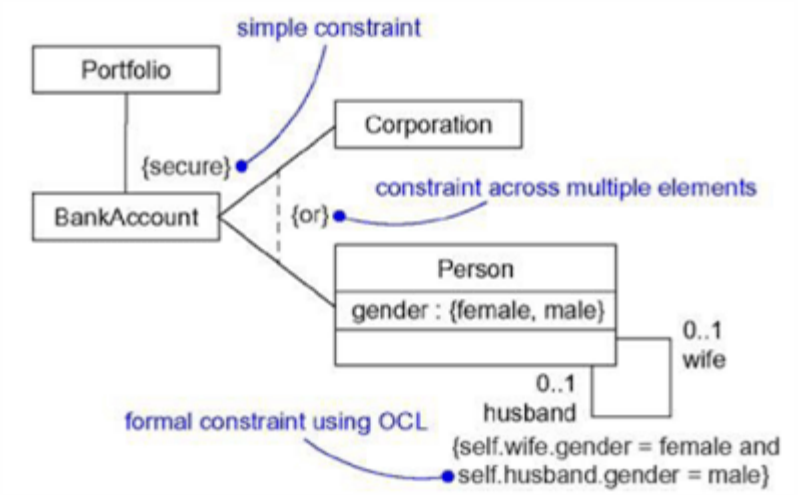
Tagged Values

Every element in UML has its own properties. For example a class has its own attributes and operations. Using tagged values, we can represent new properties also called as metadata. These properties apply to the element itself rather than its instance. Tagged values are enclosed in braces { } and are written under the element name.



Constraints

A constraint is used to add new semantics or change existing rules. The constraints specify rules that must be followed by the elements in the model. Represented as text inside braces { } and placed near to the associated element.



Defining Things

Things

A diagram can be viewed as a graph containing vertices and edges. In UML, vertices are replaced by things, and the edges are replaced by relationships. There are four types of things in UML. They are:

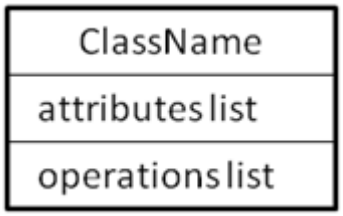
1) Structural things (static parts)
2) Behavioral things (dynamic parts)
3) Grouping things (organizational parts)
4) Annotational things (explanatory parts)

Structural things

Represents the static aspects of a software system. There are seven structural things in UML. They are:

Class: A class is a collection of similar objects having similar attributes, behavior, relationships and semantics. Graphically class is represented as a rectangle with three compartments.

Graphical representation:

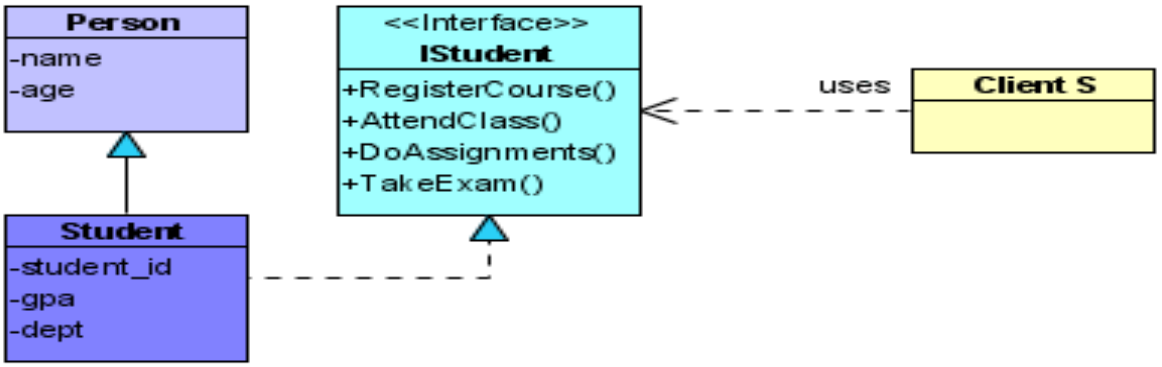


Example:

Interface: An interface is a collection of operation signatures and/or attribute definitions that ideally define a cohesive set of behavior. Graphically interface is represented as a circle or a class symbol stereotyped with interface.

Graphical representation:

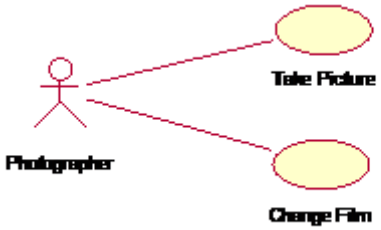
or



Use Case: A use case is a collection of actions, defining the interactions between a role (actor) and the system. Graphically use case is represented as a solid ellipse with its name written inside or below the ellipse.

Graphical representation:

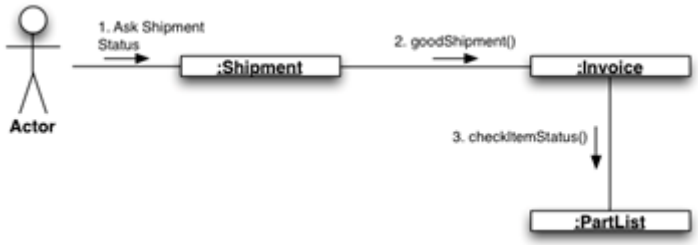
Example:



Collaboration: A collaboration is the collection of interactions among objects to achieve a goal. Graphically collaboration is represented as a dashed ellipse. A collaboration can be a collection of classes or other elements.

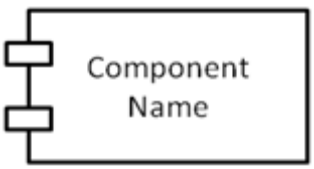
Graphical representation:

Example:

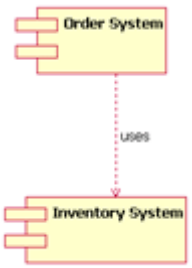


Component: A component is a physical and replaceable part of a system. Graphically component is represented as a tabbed rectangle. Examples of components are executable files, dll files, database tables, files and documents.

Graphical representation:

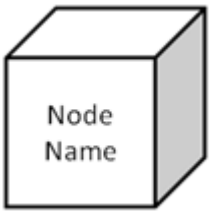


Example:

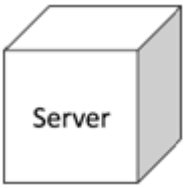


Node: A node is a physical element that exists at run time and represents a computational resource. Graphically node is represented as a cube. Examples of nodes are PCs, laptops, smartphones or any embedded system.

Graphical representation:

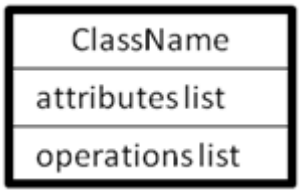


Example:

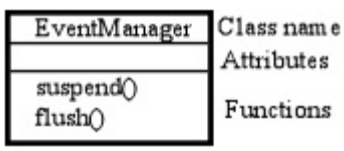


Active Class: A class whose objects can initiate its own flow of control (threads) and work in parallel with other objects. Graphically active class is represented as a rectangle with thick borders.

Graphical representation:



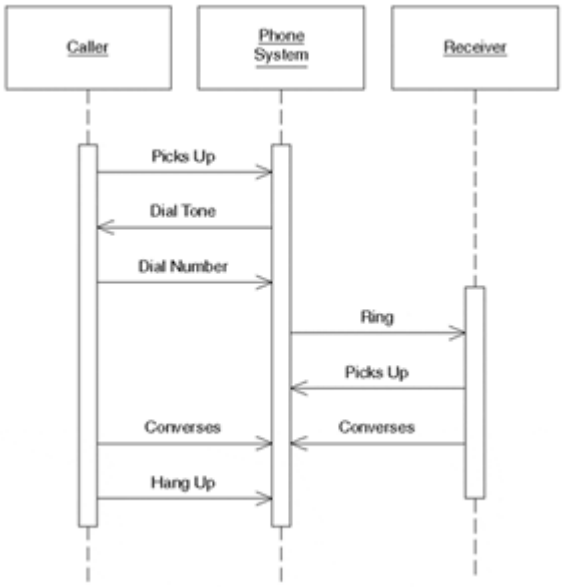
Example:



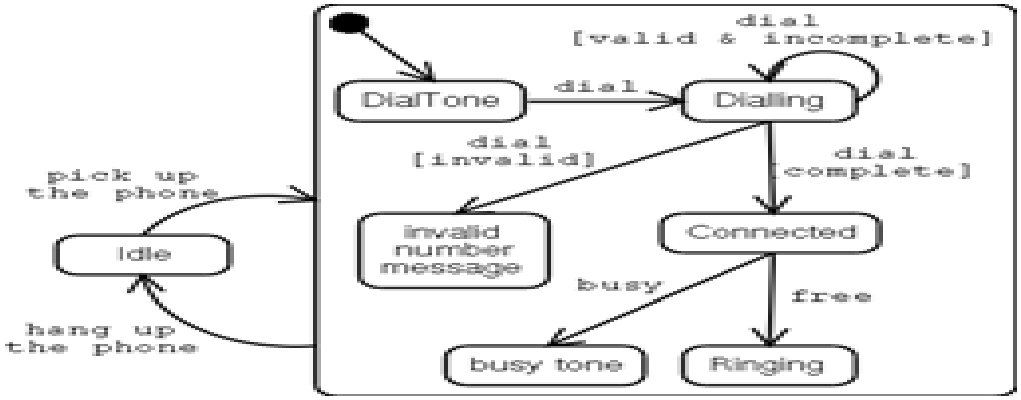
Behavioral things

Represents the dynamic aspects of a software system. Behavior of a software system can be modeled as interactions or as a sequence of state changes.

Interaction: A behavior made up of a set of messages exchanged among a set of objects to perform a particular task. A message is represented as a solid arrow. Below is an example of interaction representing a phone conversation:



State Machine: A behavior that specifies the sequences of states an object or interaction goes through during its' lifetime in response to events. A state is represented as a rectangle with rounded corners. Below is an example of state machine representing the states of a phone system:

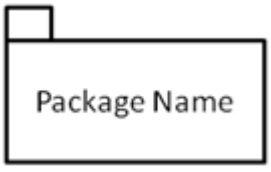


Grouping things

Elements which are used for organizing related things and relationships in models.

Package: A general purpose mechanism for organizing elements into groups. Graphically package is represented as a tabbed folder. When the diagrams become large and cluttered, related are grouped into a package so that the diagram can become less complex and easy to understand.

Graphical representation:

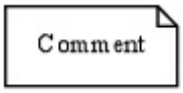


Example:

Annotational things

Note: A symbol to display comments. Graphically note is represented as a rectangle with a dog ear at the top right corner.

Graphical representation:

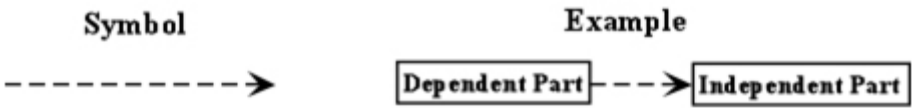


Relationships

The things in a diagram are connected through relationships. So, a relationship is a connection between two or more things.

Dependency: A semantic relationship, in which a change in one thing (the independent thing) may cause changes in the other thing (the dependent thing). This relationship is also known as "using" relationship. Graphically represented as dashed line with stick arrow head.

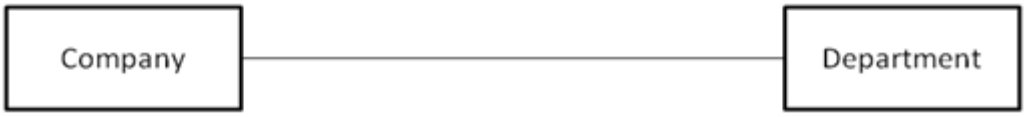
Graphical representation:



Example:

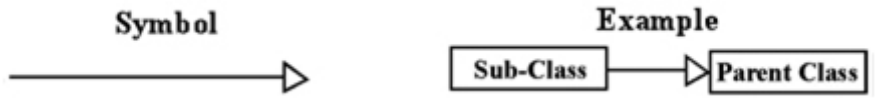
Association: A structural relationship describing connections between two or more things. Graphically represented as a solid line with optional stick arrow representing navigation.

Example:

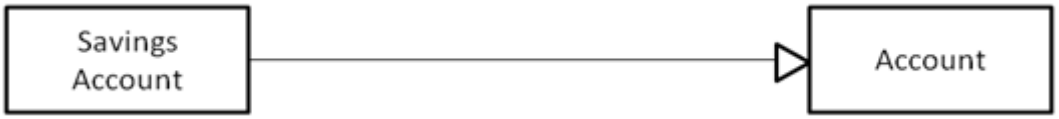


Generalization: Is a generalization-specialization relationship. Simply put this describes the relationship of a parent class (generalization) to its subclasses (specializations). Also known as "is-a" relationship.

Graphical representation:

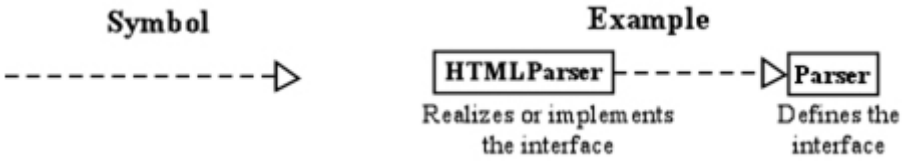


Example:



Realization: Defines a semantic relationship in which one class specifies something that another class will perform. Example: The relationship between an interface and the class that realizes or executes that interface.

Graphical representation and Example:



UML DIAGRAMS

We prepare UML diagrams to understand the system in a better and simple way. A single diagram is not enough to cover all the aspects of the system. UML defines various kinds of diagrams to cover most of the aspects of a system.

You can also create your own set of diagrams to meet your requirements. Diagrams are generally made in an incremental and iterative way.

There are two broad categories of diagrams and they are again divided into subcategories −

- Structural Diagrams
- Behavioral Diagrams

Structural Diagrams

The structural diagrams represent the static aspect of the system. These static aspects represent those parts of a diagram, which forms the main structure and are therefore stable.

These static parts are represented by classes, interfaces, objects, components, and nodes. Thefour structural diagrams are −

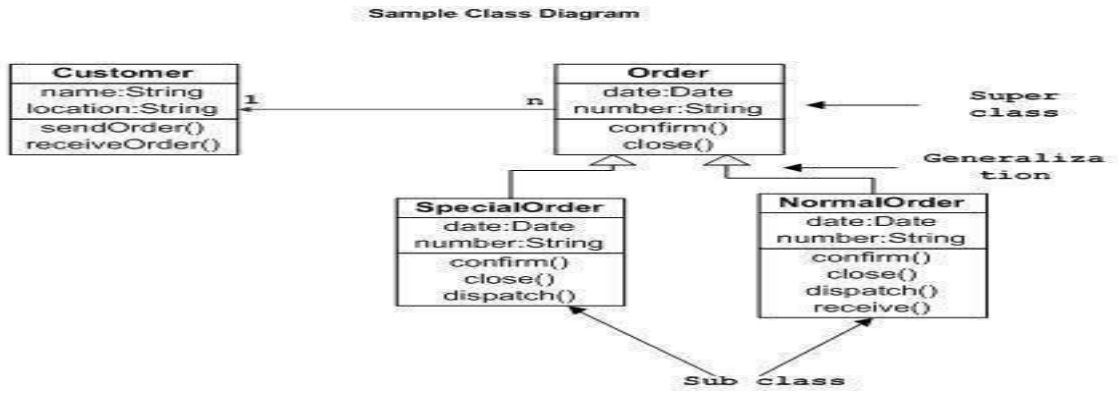
- Class diagram
- Object diagram
- Component diagram
- Deployment diagram

Class Diagram

Class diagrams are the most common diagrams used in UML. Class diagram consists of classes, interfaces, associations, and collaboration. Class diagrams basically represent the object-oriented view of a system, which is static in nature.

Active class is used in a class diagram to represent the concurrency of the system.

Class diagram represents the object orientation of a system. Hence, it is generally used for development purpose. This is the most widely used diagram at the time of system construction.

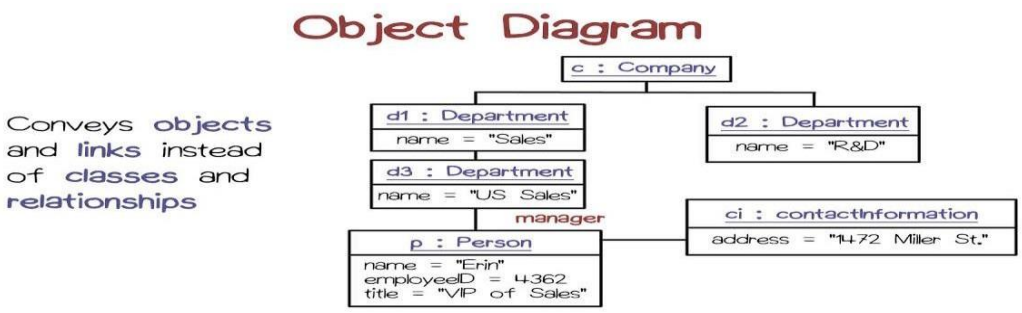


Object Diagram

Object diagrams can be described as an instance of class diagram. Thus, these diagrams are more close to real-life scenarios where we implement a system.

Object diagrams are a set of objects and their relationship is just like class diagrams. They also represent the static view of the system.

The usage of object diagrams is similar to class diagrams but they are used to build prototype of a system from a practical perspective.

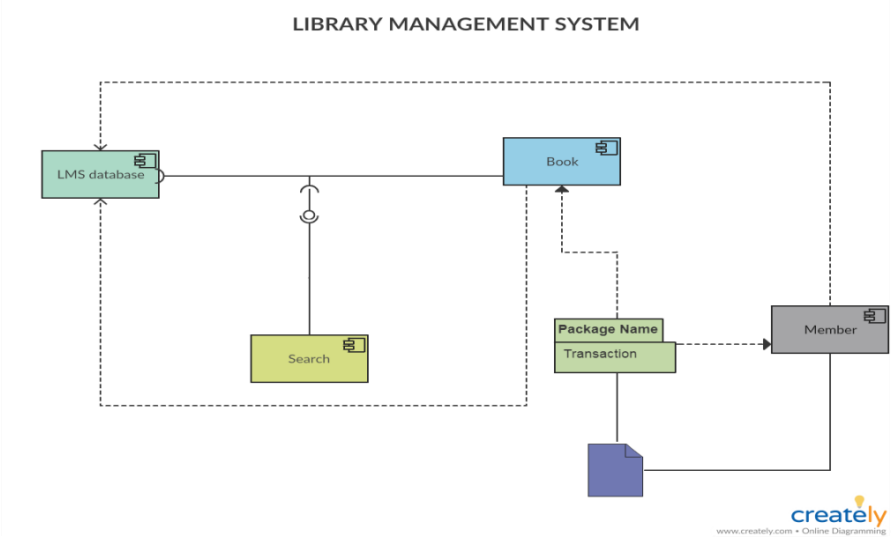


Component Diagram

Component diagrams represent a set of components and their relationships. These components consist of classes, interfaces, or collaborations. Component diagrams represent the implementation view of a system.

During the design phase, software artifacts (classes, interfaces, etc.) of a system are arrangedin different groups depending upon their relationship. Now, these groups are known as components.

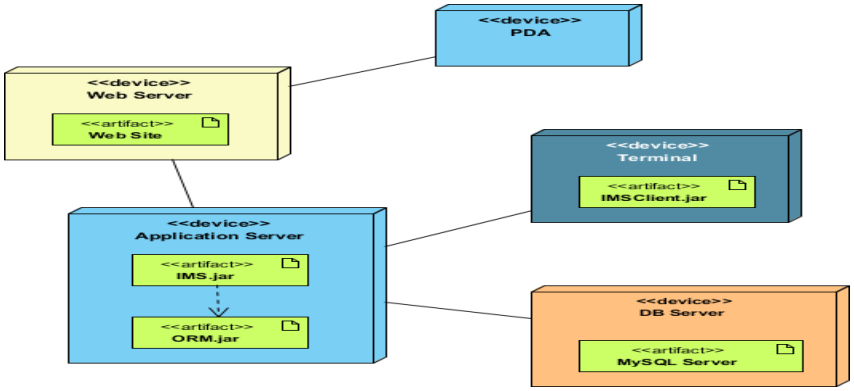
Finally, it can be said component diagrams are used to visualize the implementation.



Deployment Diagram

Deployment diagrams are a set of nodes and their relationships. These nodes are physicalentities where the components are deployed.

Deployment diagrams are used for visualizing the deployment view of a system. This is generally used by the deployment team.



Note − If the above descriptions and usages are observed carefully then it is very clear that all the diagrams have some relationship with one another. Component diagrams are dependent upon the classes, interfaces, etc. which are part of class/object diagram. Again, the deployment diagram is dependent upon the components, which are used to make component diagrams.

Behavioral Diagrams

Any system can have two aspects, static and dynamic. So, a model is considered as completewhen both the aspects are fully covered.

Behavioral diagrams basically capture the dynamic aspect of a system. Dynamic aspect canbe further described as the changing/moving parts of a system.

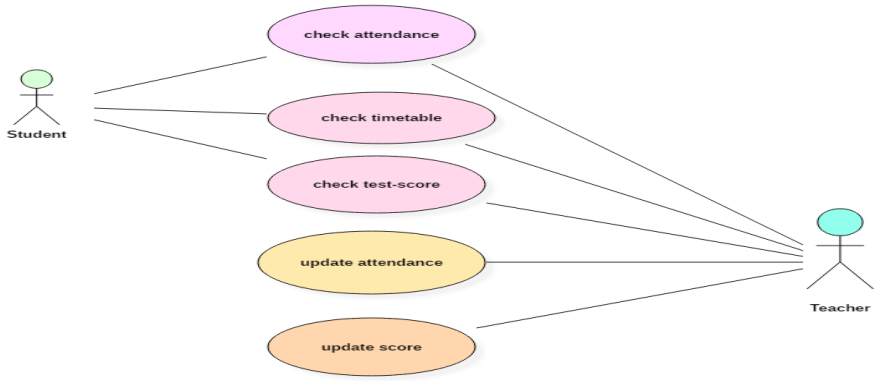
UML has the following five types of behavioral diagrams −

- Use case diagram
- Sequence diagram
- Collaboration diagram
- State chart diagram
- Activity diagram

Use Case Diagram

Use case diagrams are a set of use cases, actors, and their relationships. They represent the use case view of a system.

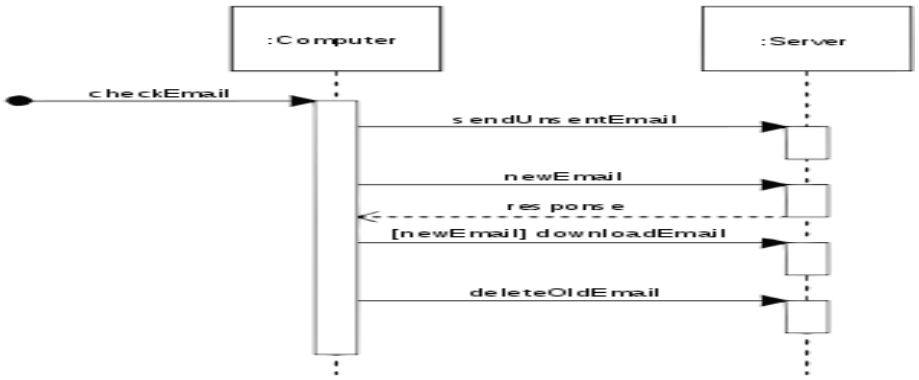
A use case represents a particular functionality of a system. Hence, use case diagram is used to describe the relationships among the functionalities and their internal/external controllers. These controllers are known as **actors**.



Sequence Diagram

A sequence diagram is an interaction diagram. From the name, it is clear that the diagram deals with some sequences, which are the sequence of messages flowing from one object to another.

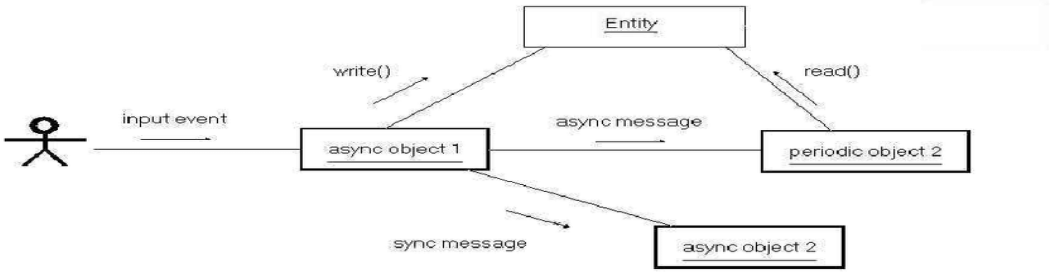
Interaction among the components of a system is very important from implementation and execution perspective. Sequence diagram is used to visualize the sequence of calls in a system to perform a specific functionality.



Collaboration Diagram

Collaboration diagram is another form of interaction diagram. It represents the structural organization of a system and the messages sent/received. Structural organization consists of objects and links.

The purpose of collaboration diagram is similar to sequence diagram. However, the specific purpose of collaboration diagram is to visualize the organization of objects and their interaction.

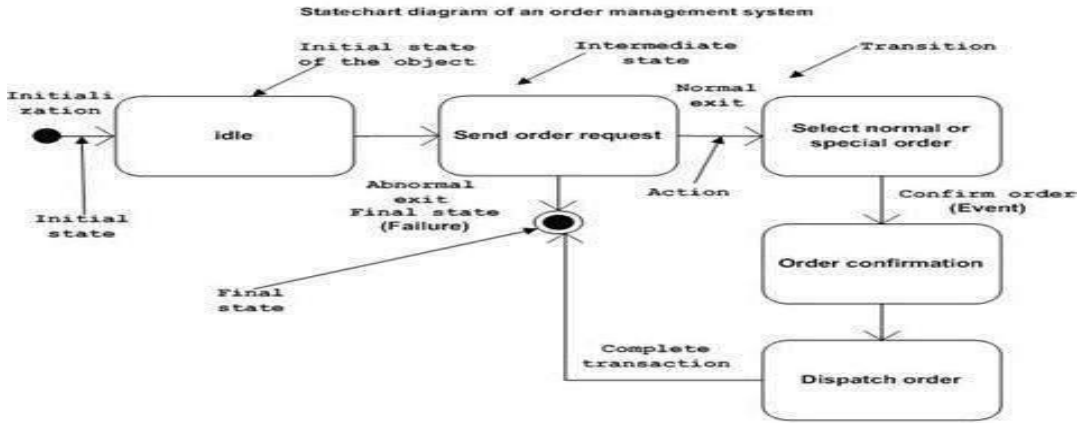


State chart Diagram

Any real-time system is expected to be reacted by some kind of internal/external events.These events are responsible for state change of the system.

State chart diagram is used to represent the event driven state change of a system. It basically describes the state change of a class, interface, etc.

State chart diagram is used to visualize the reaction of a system by internal/external factors.

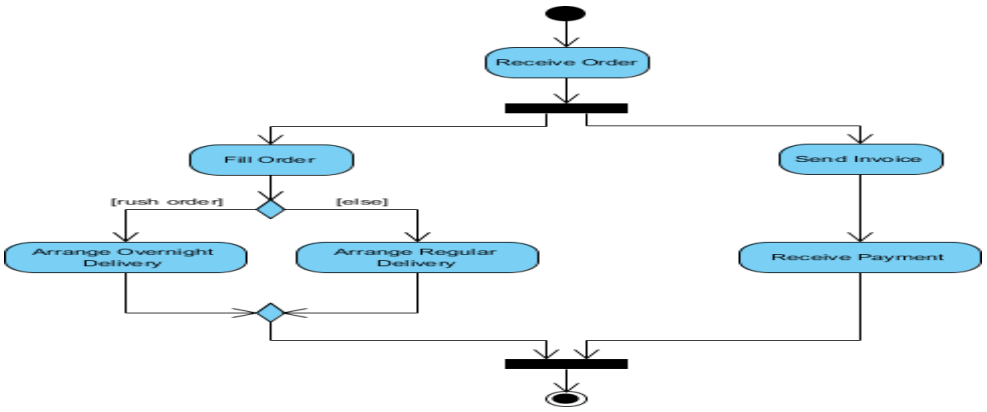


Activity Diagram

Activity diagram describes the flow of control in a system. It consists of activities and links. The flow can be sequential, concurrent, or branched.

Activities are nothing but the functions of a system. Numbers of activity diagrams are prepared to capture the entire flow in a system.

Activity diagrams are used to visualize the flow of controls in a system. This is prepared to have an idea of how the system will work when executed.



Note − Dynamic nature of a system is very difficult to capture. UML has provided features to capture the dynamics of a system from different angles. Sequence diagrams and collaboration diagrams are isomorphic, hence they can be converted from one another without losing any information. This is also true for State chart and activity diagram.

Class Diagram

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.

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 modeling of objectoriented systems because they are the only UML diagrams, which can be mapped directly with object-oriented languages.

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

Purpose of Class Diagrams

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

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

The purpose of the class diagram can be summarized as −

- Analysis and design of the static view of an application.

- Base for component and deployment diagrams.
- Forward and reverse engineering.

How to Draw a Class Diagram?

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

Class diagrams have a 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. A collection of class diagrams represent the whole system.

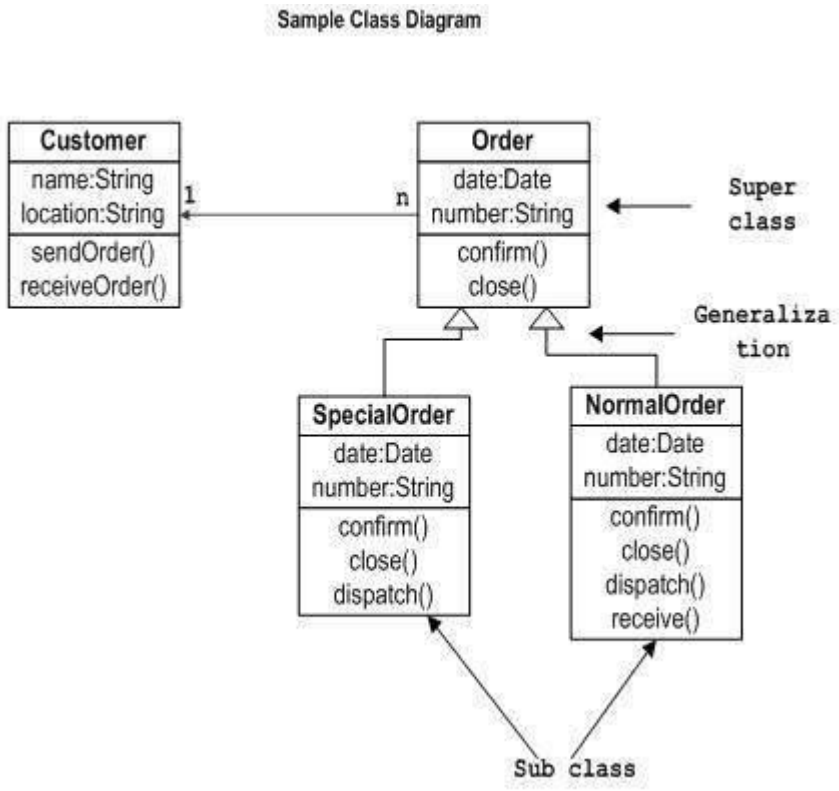
The following points should be remembered while drawing a class diagram −

- 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, as unnecessary properties will make the diagram complicated.
- Use notes whenever required to describe some aspect of the diagram. 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 reworked as many times as possible to make it correct.

The following diagram is an example of an Order System of an application. It describes a particular aspect of the entire application.

- First of all, Order and Customer are identified as the two elements of the system. They have a one-to-many relationship because a customer can have multiple orders.
- 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 ().

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 the 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 such as Java, C++, etc. From practical experience, class diagram is generally used for construction purpose.

In a nutshell it can be said, 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.

Object Diagrams

Object diagrams are derived from class diagrams so object diagrams are dependent upon class diagrams.

Object diagrams represent an instance of a class diagram. The basic concepts are similar for class diagrams and object diagrams. Object diagrams also represent the static view of a system but this static view is a snapshot of the system at a particular moment.

Object diagrams are used to render a set of objects and their relationships as an instance.

Purpose of Object Diagrams

The purpose of a diagram should be understood clearly to implement it practically. The purposes of object diagrams are similar to class diagrams.

The difference is that a class diagram represents an abstract model consisting of classes and their relationships. However, an object diagram represents an instance at a particular moment, which is concrete in nature.

It means the object diagram is closer to the actual system behavior. The purpose is to capture the static view of a system at a particular moment.

The purpose of the object diagram can be summarized as −

- Forward and reverse engineering.
- Object relationships of a system
- Static view of an interaction.
- Understand object behaviour and their relationship from practical perspective

How to Draw an Object Diagram?

We have already discussed that an object diagram is an instance of a class diagram. It implies that an object diagram consists of instances of things used in a class diagram.

So both diagrams are made of same basic elements but in different form. In class diagram elements are in abstract form to represent the blue print and in object diagram the elements are in concrete form to represent the real world object.

To capture a particular system, numbers of class diagrams are limited. However, if we consider object diagrams then we can have unlimited number of instances,

which are unique in nature. Only those instances are considered, which have an impact on the system.

From the above discussion, it is clear that a single object diagram cannot capture all the necessary instances or rather cannot specify all the objects of a system. Hence, the solution is −

- First, analyze the system and decide which instances have important data and association.
- Second, consider only those instances, which will cover the functionality.
- Third, make some optimization as the number of instances are unlimited.

Before drawing an object diagram, the following things should be remembered and understood clearly −

- Object diagrams consist of objects.
- The link in object diagram is used to connect objects.
- Objects and links are the two elements used to construct an object diagram.

After this, the following things are to be decided before starting the construction of the diagram −

- The object diagram should have a meaningful name to indicate its purpose.
- The most important elements are to be identified.
- The association among objects should be clarified.
- Values of different elements need to be captured to include in the object diagram.
- Add proper notes at points where more clarity is required.

The following diagram is an example of an object diagram. It represents the Order management system which we have discussed in the chapter Class Diagram. The following diagram is an instance of the system at a particular time of purchase. It has the following objects.

- Customer
- Order
- SpecialOrder
- NormalOrder

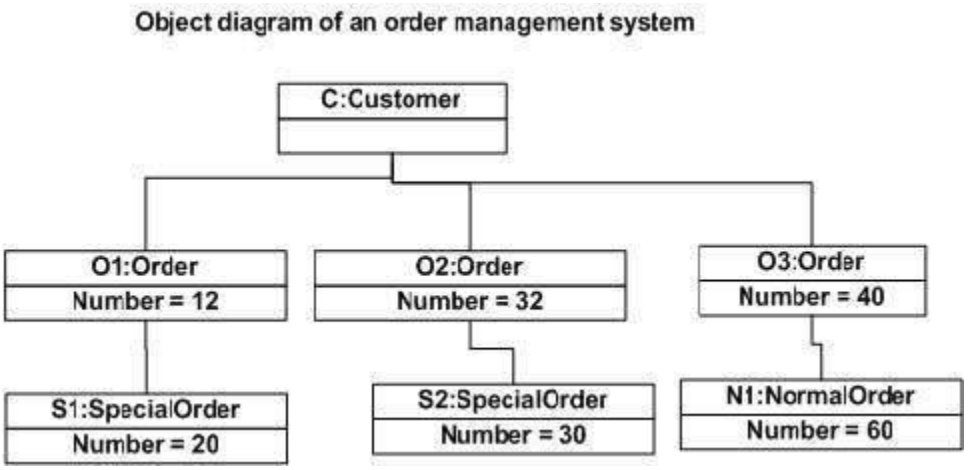
Now the customer object (C) is associated with three order objects (O1, O2, and O3). These order objects are associated with special order and normal order objects (S1, S2, and N1). The customer has the following three orders with different numbers (12, 32 and 40) for the particular time considered.

The customer can increase the number of orders in future and in that scenario the object diagram will reflect that. If order, special order, and normal order objects are observed then you will find that they have some values.

For orders, the values are 12, 32, and 40 which implies that the objects have these values for a particular moment (here the particular time when the purchase is made is considered as the moment) when the instance is captured

The same is true for special order and normal order objects which have number of orders as 20, 30, and 60. If a different time of purchase is considered, then these values will change accordingly.

The following object diagram has been drawn considering all the points mentioned above



Where to Use Object Diagrams?

Object diagrams can be imagined as the snapshot of a running system at a particular moment. Let us consider an example of a running train

Now, if you take a snap of the running train then you will find a static picture of it having the following −

- A particular state which is running.
- A particular number of passengers. which will change if the snap is taken in a different time

Here, we can imagine the snap of the running train is an object having the above values. And this is true for any real-life simple or complex system.

In a nutshell, it can be said that object diagrams are used for −

- Making the prototype of a system.
- Reverse engineering.
- Modeling complex data structures.

- Understanding the system from practical perspective.

Class to Object Diagram Example - Order System