Object Oriented System

Object oriented systems represent data as objects. Other objects and users interact with these objects. Both the data and the information related to the executable file required for the data interpretation are comprised of object.

In the object-oriented approach, the focus is on capturing the structure and behavior of information systems into small modules that combines both data and process. The main aim of Object Oriented Design (OOD) is to improve the quality and productivity of system analysis and design by making it more usable.

Object-oriented analysis is a method of analysis that examines requirements from the perspective of the classes and objects found in the vocabulary of the problem domain. Object-oriented design is a method of design encompassing the process of object-oriented decomposition and a notation for depicting logical and physical as well as static and dynamic models of the system under design.

Object Oriented Development Life Cycle

The Object Oriented Methodology of Building Systems takes the objects as the basis. For this, first the system to be developed is observed and analyzed and the requirements are defined as in any other method of system development. Once this is done, the objects in the required system are identified. For example in case of a Banking System, a customer is an object, a chequebook is an object, and even an account is an object. Object oriented development life cycle contains:

1. System Analysis
2. System Design
3. Object Design
4. Implementation

Object Oriented Development Life Cycle

1. System Analysis-As in any other system development model, system analysis is the first phase of development in case of Object Modeling too. In this phase, the developer interacts with the user of the system to find out the user requirements and analyses the system to understand the functioning. Based on this system study, the analyst prepares a model of the desired system. This model is purely based on what the system is required to do. At this stage the implementation details are not taken care of. Only the model of the system is prepared based on the idea that the system is made up of a set of interacting objects. The important elements of the system are emphasized

Object Oriented Development Life Cycle

1. System Design -System Design is the next development stage where the overall architecture of the desired system is decided. The system is organized as a set of sub systems interacting with each other. While designing the system as a set of interacting subsystems, the analyst takes care of specifications as observed in system analysis as well as what is required out of the new system by the end user. As the basic philosophy of Object-Oriented method of system analysis is to perceive the system as a set of interacting objects, a bigger system may also be seen as a set of interacting smaller subsystems that in turn are composed of a set of interacting objects. While designing the system, the stress lies on the objects comprising the system and not on the processes being carried out in the system as in the case of traditional Waterfall Model where the processes form the important part of the system.

Object Oriented Development Life Cycle

1. Object Design-n this phase, the details of the system analysis and system design are implemented. The Objects identified in the system design phase are designed. Here the implementation of these objects is decided as the data structures get defined and also the interrelationships between the objects are defined. Object Oriented Philosophy is very much similar to real world and hence is gaining popularity as the systems here are seen as a set of interacting objects as in the real world. To implement this concept, the process-based structural programming is not used; instead objects are created using data structures. Just as every programming language provides various data types and various variables of that type can be created, similarly, in case of objects certain data types are predefined. For example, we can define a data type called pen and then create and use several objects of this data type.

Object Oriented Development Life Cycle

Implementation-During this phase, the class objects and the interrelationships of these classes are translated and actually coded using the programming language decided upon. The databases are made and the complete system is given a functional shape

Object Oriented Model

While developing systems based on this approach, the analyst makes use of certain models to analyze and depict these objects. The methodology supports and uses three basic Models:

Object Model - This model describes the objects in a system and their interrelationships. This model observes all the objects as static and does not pay any attention to their dynamic nature.

Dynamic Model - This model depicts the dynamic aspects of the system. It portrays the changes occurring in the states of various objects with the events that might occur in the system.

Functional Model - This model basically describes the data transformations of the system. This describes the flow of data and the changes that occur to the data throughout the system.

Basic Characteristics of Object Oriented System

Class-A class is a collection of similar objects. It is a template where certain basic characteristics of a set of objects are defined. The class defines the basic attributes and the operations of the objects of that type. Defining a class does not define any object, but it only creates a template. For objects to be actually created instances of the class are created as per the requirement of the case.

Abstraction-The process of abstraction hides all the low-level details and represents only those features that are essential. It is one of the important characteristics of an object using which the particular object can be differentiated from others. It helps in defining boundaries in a crisp manner with regard to the viewer's perspective. The prime objective of abstraction is to isolate the important aspects from the unimportant ones and suppress the later. Classes are also known as a list of abstract properties as they implement abstraction. It is possible to simplify the complex reality with the help of abstraction. This is done by designing classes that fits with the problem. Work can be also carried out at the most suitable level of inheritance based on the problem.

Basic Characteristics of Object Oriented System

- Inheritance- Inheritance is the process of declaring and defining a new class by extending features of existing class. The extension of features is done in terms of procedures or methods and data. With the help of inheritance the objects belonging to one class are able to get the properties of the objects belonging to another class. With the help of inheritance, code can be reused by just adding the new features to an existing class without carrying out any modification. To do this a new class has to be derived from the existing class.
- Polymorphism-The capability of taking multiple forms is known as polymorphism. Depending on the instance an operation may change its behavior. The behavior is based on the data type that has been used to perform the operation. In order to implement inheritance polymorphism is used extensively. Using polymorphism the programmer is able to treat the members of parent class and derived class in the same way. Polymorphism provides ability to the objects of different data types to respond to methods of same name based on some particular behavior based on a specific type. It becomes possible to abstractly use an operator like +, - or* in various situations.

Basic Characteristics of Object Oriented System

- Inheritance- Inheritance is the process of declaring and defining a new class by extending features of existing class. The extension of features is done in terms of procedures or methods and data. With the help of inheritance the objects belonging to one class are able to get the properties of the objects belonging to another class. With the help of inheritance, code can be reused by just adding the new features to an existing class without carrying out any modification. To do this a new class has to be derived from the existing class.
- Polymorphism-The capability of taking multiple forms is known as polymorphism. Depending on the instance an operation may change its behavior. The behavior is based on the data type that has been used to perform the operation. In order to implement inheritance polymorphism is used extensively. Using polymorphism the programmer is able to treat the members of parent class and derived class in the same way. Polymorphism provides ability to the objects of different data types to respond to methods of same name based on some particular behavior based on a specific type. It becomes possible to abstractly use an operator like +, - or* in various situations.

Basic Characteristics of Object Oriented System

Encapsulation-Encapsulation is defined as the process of storing data and associated functions within a single class. This process is also known as information hiding and is complimentary to abstraction. In order to create the objects both data and code are defined using this process. However, the process hides both code and data. Some of the code and data remain restricted as private member of the object while the others can be accessed directly as interface of the object. As a practice the data members are kept private and they are accessed using the member functions of the class which are declared in the public part. Only the functions declared within the class access the data. With the help of encapsulation it is possible to hide the operational details of a class from the objects

Advantages of object oriented methodology

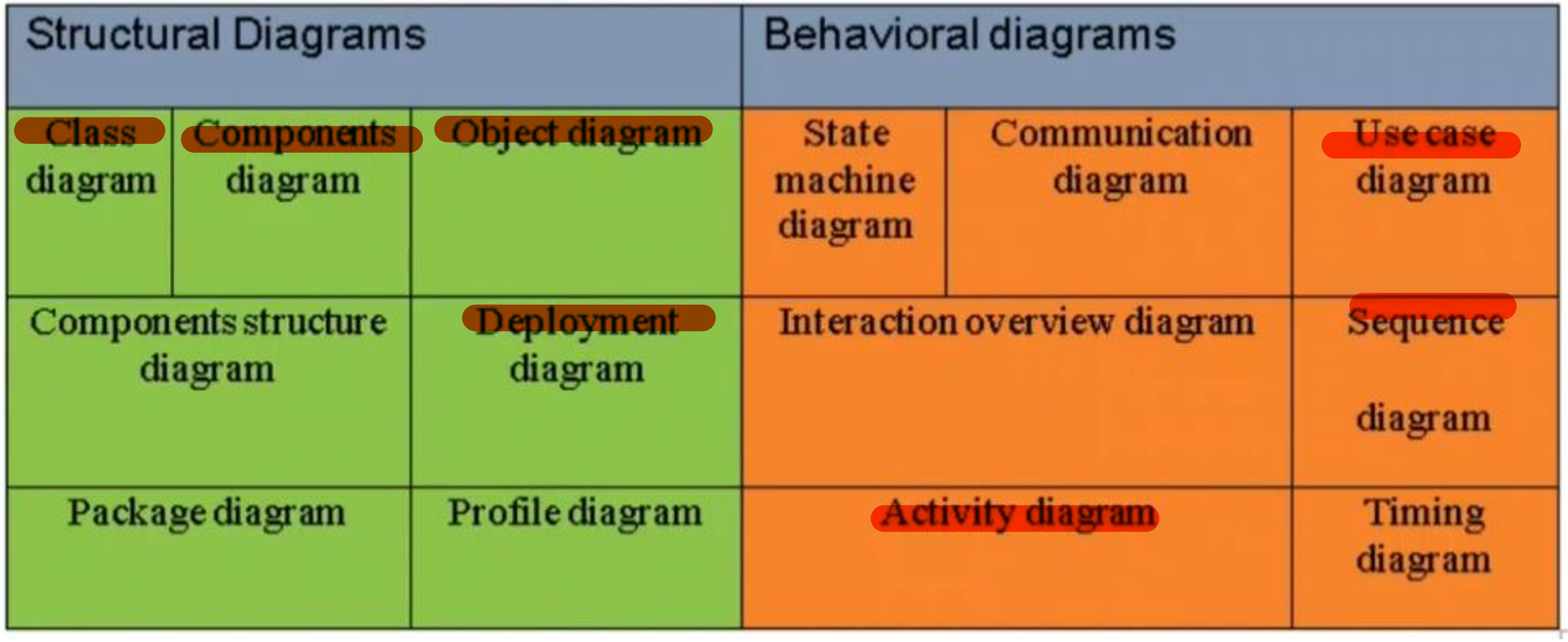
- Object Oriented Methodology closely represents the problem domain. Because of this, it is easier to produce and understand designs.
- The objects in the system are immune to requirement changes. Therefore, allows changes more easily.
- Object Oriented Methodology designs encourage more re-use. New applications can use the existing modules, thereby reduces the development cost and cycle time.
- Object Oriented Methodology approach is more natural. It provides nice structures for thinking and abstracting and leads to modular design

Unified Modeling Language (UML)

The Unified Modeling Language (UML) is a general-purpose modeling language in the field of software engineering, which is designed to provide a standard way to visualize the design of a system. It was created and developed by Grady Booch, Ivar Jacobson and James Rumbaugh. UML is designed to enable users to develop an expressive, ready to use visual modeling language. In addition, it supports high level development concepts such as frameworks, patterns and collaborations. UML diagrams can be divided into two categories. The first type includes six diagram types representing structural information. The second includes the remaining seven representing general types of behavior.

1. Structural Diagram-This type of diagram shows the different objects (things) used in the system.
2. Behavior Diagram-This type of diagram show what should happen in a system. They describe how the objects interact with each other to create a functioning system.

Unified Modeling Language (UML)



Structural Diagrams			Behavioral diagrams		
Class diagram	Components diagram	Object diagram	State machine diagram	Communication diagram	Use case diagram
Components structure diagram		Deployment diagram	Interaction overview diagram		Sequence diagram
Package diagram		Profile diagram	Activity diagram		Timing diagram

Class Diagram

- The class diagram depicts a static view of an application. It represents the types of objects residing in the system and the relationships between them. A class consists of its objects, and also it may inherit from other classes. A class diagram is used to visualize, describe, document various different aspects of the system, and also construct executable software code.
- It shows the attributes, classes, functions, and relationships to give an overview of the software system

components of a Class Diagram

- **Upper Section-**ClassName
- **Middle Section-**Attributes
- **Lower Section-**Methods

Class Notation

- **A class notation represent three parts and we enclose them in a rectangle.**
- **Upper Section(**ClassName) Every class must have a name and the name of the class appears in the top and first part of the rectangle.
- **Middle Section(**Attributes)- The second part of the class is its attributes which is used to represent the property of the class. It is written in the middle portion od rectangle and the data type of attributes is written after colon.
- **Lower Section(**Methods)-Operation are shown in the third partition. It shows the actual function performed on the class. The + operator is written before class operation.

Library
-Library reg_no :int -Library name: string -Address: string
+Add book +Remove book

Object Diagram

- 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.

Object Diagram

Notation of an object diagram

Object name: Class
+ object attributes

Example of object diagram

Apple: Fruit
+color: red +quantity: 1kg

Mango: Fruit
+color: yellow +quantity: 2kg

How to draw Object Diagram

1.All the objects present in the system should be examined before start drawing the object diagram.

2.Before creating the object diagram, the relation between the objects must be acknowledged.

3.The association relationship among the entities must be cleared already.

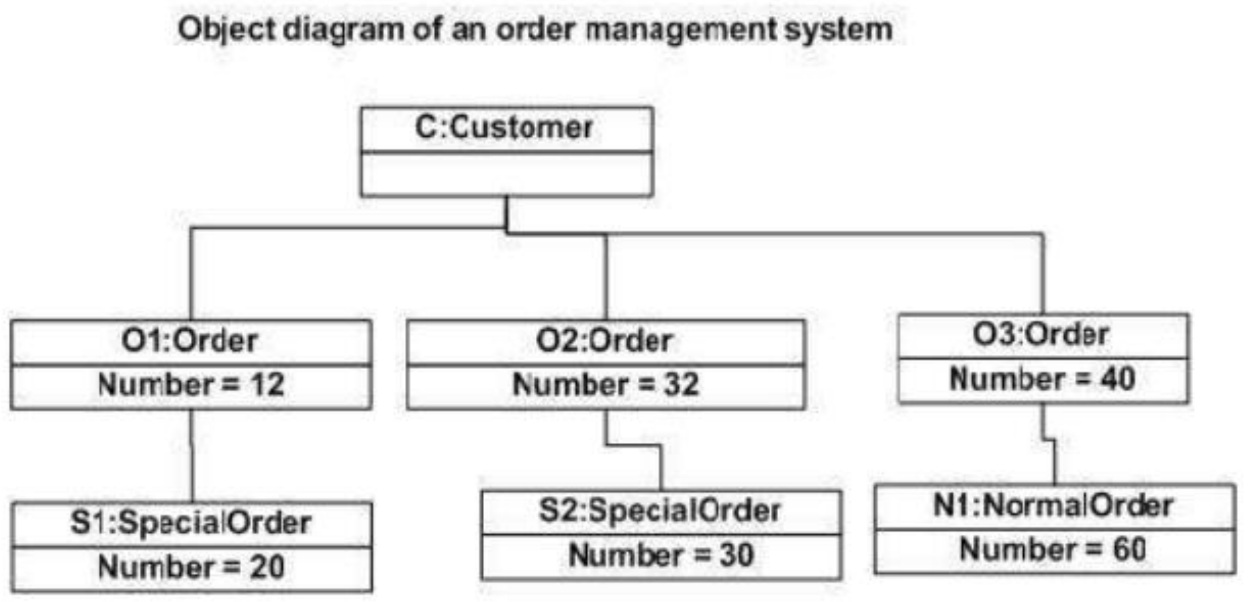
4.To represent the functionality of an object, a proper meaningful name should be assigned.

5.The objects are to be examined to understand its functionality.

Class diagram vs. Object diagram

Class Diagram	Object Diagram
It depicts the static view of a system.	It portrays the real-time behavior of a system.
Dynamic changes are not included in the class diagram.	Dynamic changes are captured in the object diagram.
The data values and attributes of an instance are not involved here.	It incorporates data values and attributes of an entity.
The object behavior is manipulated in the class diagram.	Objects are the instances of a class.

Object diagram



Deployment diagram

The deployment diagram visualizes the physical hardware on which the software will be deployed. It portrays the static deployment view of a system. It involves the nodes and their relationships.

It ascertains how software is deployed on the hardware. It maps the software architecture created in design to the physical system architecture, where the software will be executed as a node. Since it involves many nodes, the relationship is shown by utilizing communication paths.

Purpose of Deployment Diagram

The main purpose of the deployment diagram is to represent how software is installed on the hardware component. It depicts in what manner a software interacts with hardware to perform its execution.
Both the deployment diagram and the component diagram are closely interrelated to each other as they focus on software and hardware components. The component diagram represents the components of a system, whereas the deployment diagram describes how they are actually deployed on the hardware.
The deployment diagram does not focus on the logical components of the system, but it put its attention on the hardware topology.
Following are the purposes of deployment diagram enlisted below:

1.To envision the hardware topology of the system.

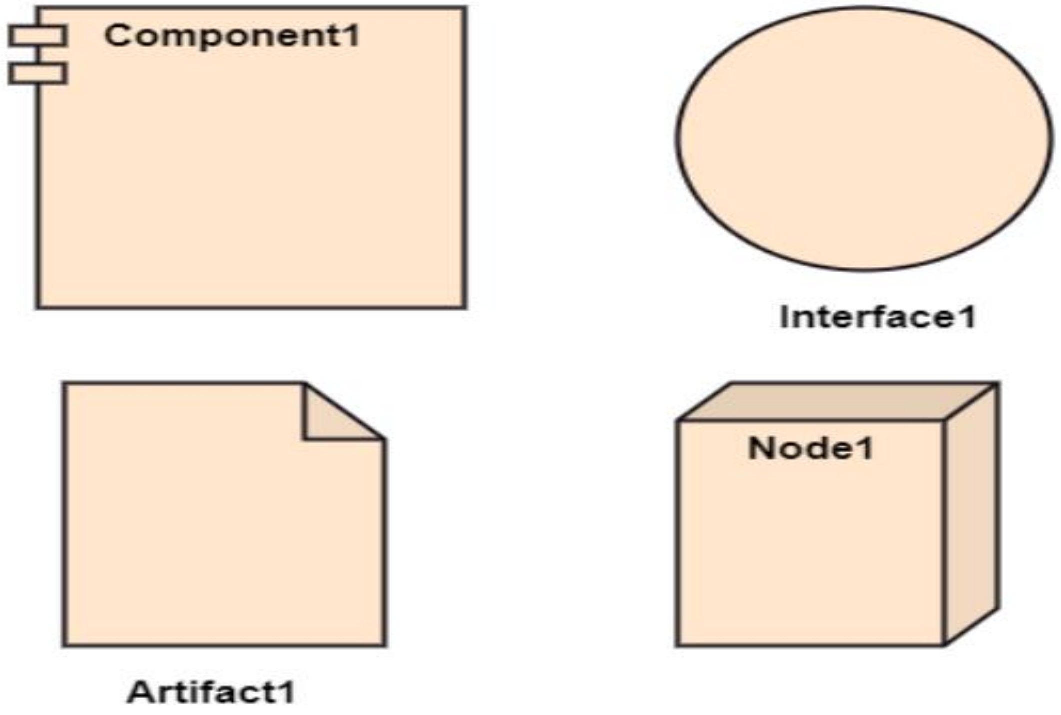
2.To represent the hardware components on which the software components are installed.

3.To describe the processing of nodes at the runtime.

Symbol and notation of Deployment diagram

The deployment diagram consist of the following notations:

1.A component
2.An artifact
3.An interface
4.A node



Symbol and notation of Deployment diagram

The deployment diagram consist of the following notations:

1.A component-A rectangle with two tabs that indicates a software element.

2.An artifact- A product developed by the software, symbolized by a rectangle with the name and the word “artifact” enclosed by double arrows. Artifacts represent concrete elements in the physical world that are the result of a development process. Examples of artifacts are executable files, libraries, archives, database schemas, configuration files, etc.

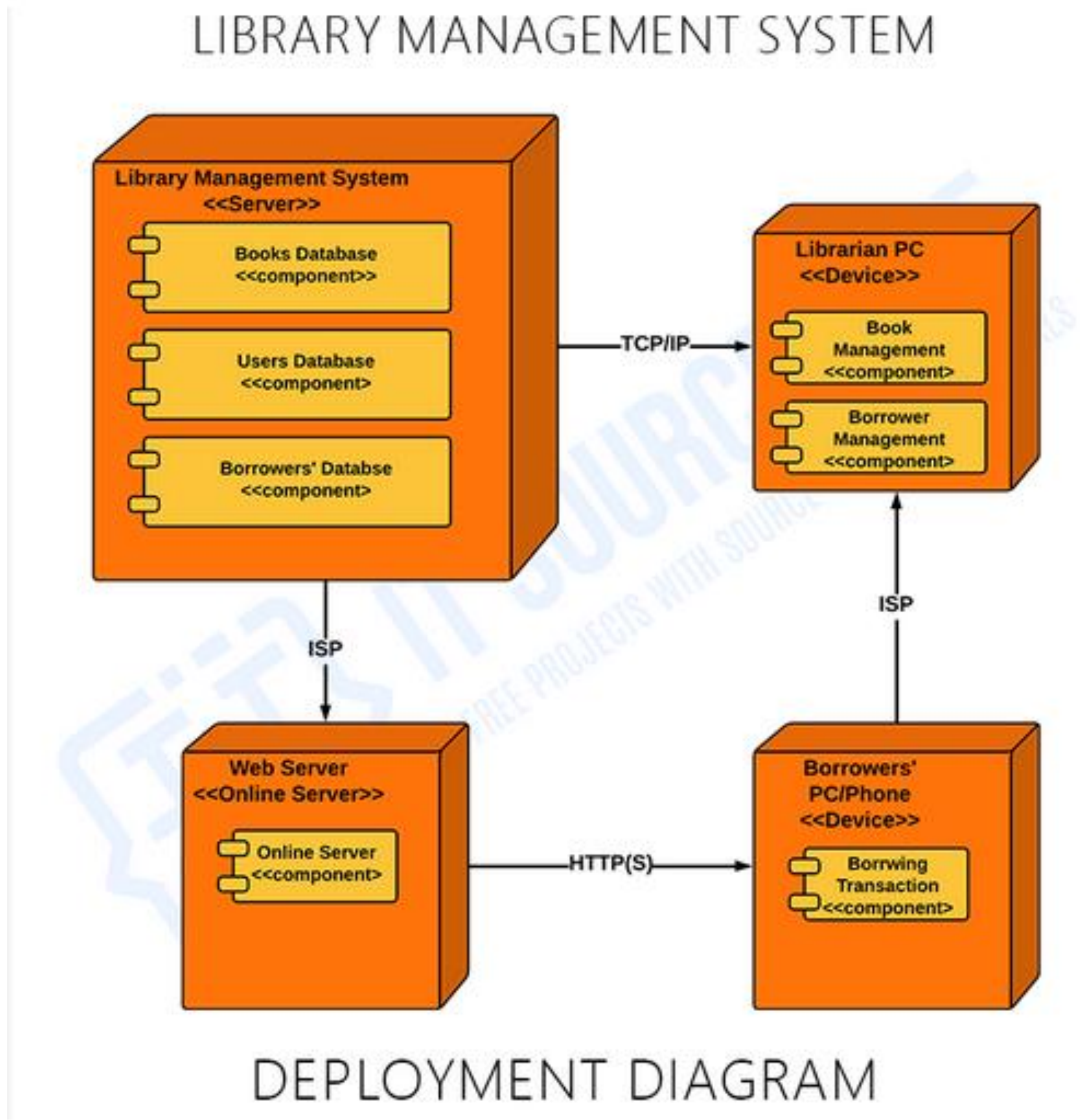
Symbol and notation of Deployment diagram

3.An interface-A circle that indicates a contractual relationship. Those objects that realize the interface must complete some sort of obligation.

4.A node-Node is a computational resource upon which artifacts are deployed for execution. A node is a physical thing that can execute one or more artifacts. A node may vary in its size depending upon the size of the project. A node has two types as follows:

- Device-It is a node that represents a physical machine capable of performing computations. A device can be a router or a server PC.
- Execution Environment-It is a node that represents an environment in which software is going to execute. For example, Java applications are executed in java virtual machine (JVM)

Example of Deployment diagram


LIBRARY MANAGEMENT SYSTEM
Library Management System <<Server>>
Books Database <<component>>
Users Database <<component>
Borrowers' Databse <<component>
TCP/IP
Librarian PC <<Device>>
Book Management <<component>
Borrower Management <<component>
ISP
ISP
Web Server <<Online Server>>
Online Server <<component>
HTTP(S)
Borrowers' PC/Phone <<Device>>
Borrwing Transaction <<component>
DEPLOYMENT DIAGRAM

Component diagram

Component diagram are different in terms of nature and behavior. Component diagrams are used to model the physical aspects of a system. Now, the question is, what are these physical aspects? Physical aspects are the elements such as executables, libraries, files, documents etc. which reside in a node.

Component diagrams are used to visualize the organization and relationships among components in a system .These diagrams are also used to make executable systems.

Purpose of Component diagram

Component diagram is a special kind of diagram in UML. The purpose is also different from all other diagrams discussed so far. It does not describe the functionality of the system but it describes the components used to make those functionalities. Thus from that point of view, component diagrams are used to visualize the physical components in a system. These components are libraries, packages, files, etc. Component diagrams can also be described as a static implementation view of a system. Static implementation represents the organization of the components at a particular moment. A single component diagram cannot represent the entire system but a collection of diagrams is used to represent the whole.

The purpose of the component diagram can be summarized as −

- Visualize the components of a system.
- Construct executables by using forward and reverse engineering.
- Describe the organization and relationships of the components.

Before drawing a component diagram, the following artifacts are to be identified clearly−

- Files used in the system.
- Libraries and other artifacts relevant to the application.
- Relationships among the artifacts

Component Diagram Symbols

Component diagram is a special kind of diagram in UML. The purpose is also different from all other diagrams discussed so far. It does not describe the functionality of the system but it describes the components used to make those functionalities. Thus from that point of view, component diagrams are used to visualize the physical components in a system. These components are libraries, packages, files, etc. Component diagrams can also be described as a static implementation view of a system. Static implementation represents the organization of the components at a particular moment. A single component diagram cannot represent the entire system but a collection of diagrams is used to represent the whole.

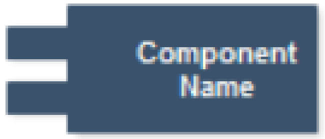
The purpose of the component diagram can be summarized as −

- ▫ Visualize the components of a system.
- ▫ Construct executables by using forward and reverse engineering.
- ▫ Describe the organization and relationships of the components.

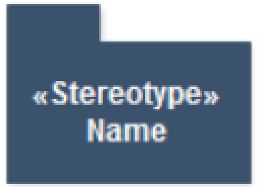
Before drawing a component diagram, the following artifacts are to be identified clearly−

- ▫ Files used in the system.
- ▫ Libraries and other artifacts relevant to the application.
- ▫ Relationships among the artifacts

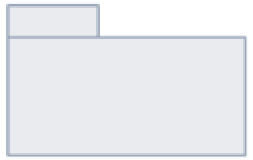
Component Diagram Symbols



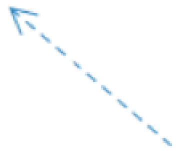
Component represents a modular part of a system. A component defines its behavior in terms of provided and required interfaces.



Package is used to group elements, and to provide a namespace for the grouped elements

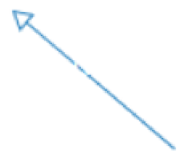


Package container is used to define UML elements such as classes, use cases, and components.

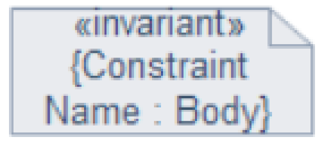


Dependency relationship is a relationship in which one element, the client, uses or depends on another element, the supplier.

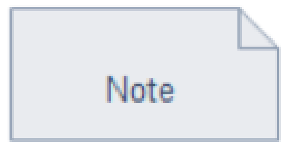
Component Diagram Symbols



Generalization is a relationship in which one model element (the child) is based on another model element (the parent).

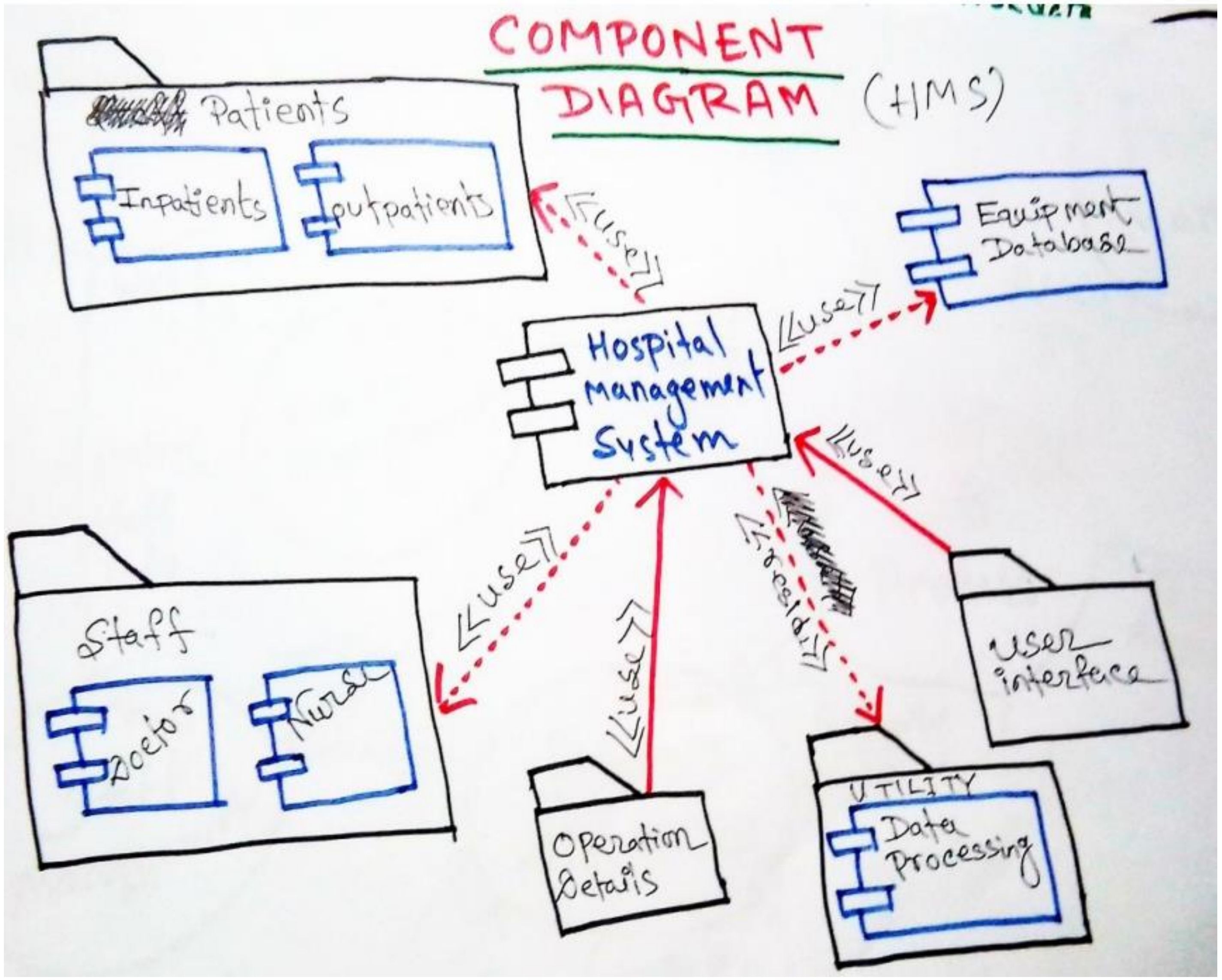


Constraint is an extension mechanism that enables you to refine the semantics of a UML model element.

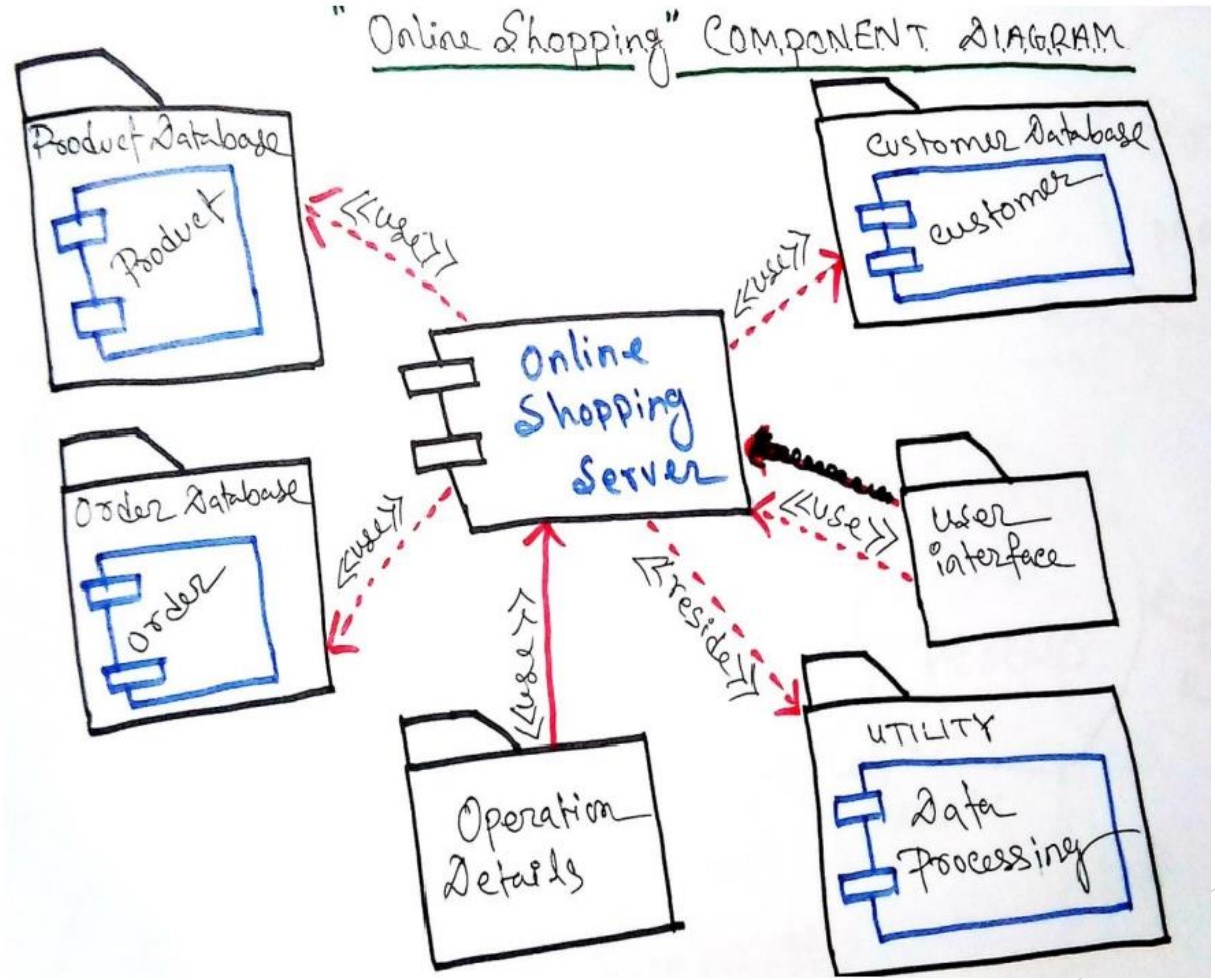


Note contains comments or textual information

Component Diagram of HMS



Component Diagram of Online Shopping



Behavioral Modelling

Behavioral model describes the interaction in the system. It represents the interaction among the structural diagrams. Behavioral modeling shows the dynamic nature of the system. They consist of the following:-

- State machine diagram
- Communication diagram
- Use Case diagram
- Interaction Overview diagram
- Sequence diagram
- Activity diagram
- Timing diagram

Use Case Diagram

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

Purpose of Use Case Diagrams

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

Following are the purposes of a use case diagram given below:

1. It gathers the system's needs.
2. It depicts the external view of the system.
3. It recognizes the internal as well as external factors that influence the system.
4. It represents the interaction between the actors.

How to draw a Use Case diagram?

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

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

Use Case Diagram objects

Use case diagrams consist of 4 objects.

•Actor

•Use case

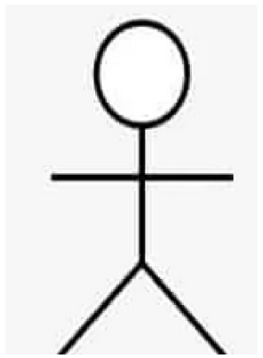
•System

•Package

Use Case Diagram objects

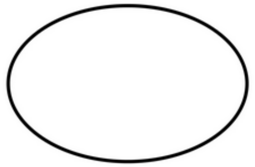
Use case diagrams consist of 4 objects.

•Actor-Actor in a use case diagram is **any entity that performs a role** in one given system. This could be a person, organization or an external system and usually drawn like skeleton shown below.

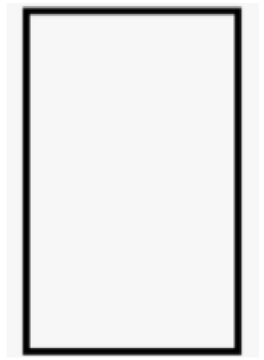


Use Case Diagram objects

- **Use case**-A use case **represents a function or an action within the system**. It’s drawn as an oval and named with the function.

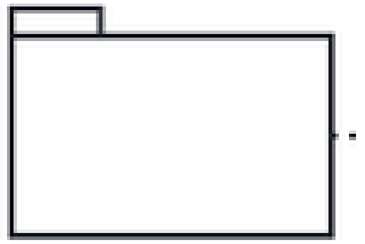


- **System-**The system is used to **define the scope of the use case** and drawn as a rectangle. This an optional element but useful when you’re visualizing large systems. For example, you can create all the use cases and then use the system object to define the scope covered by your project. Or you can even use it to show the different areas covered in different releases.



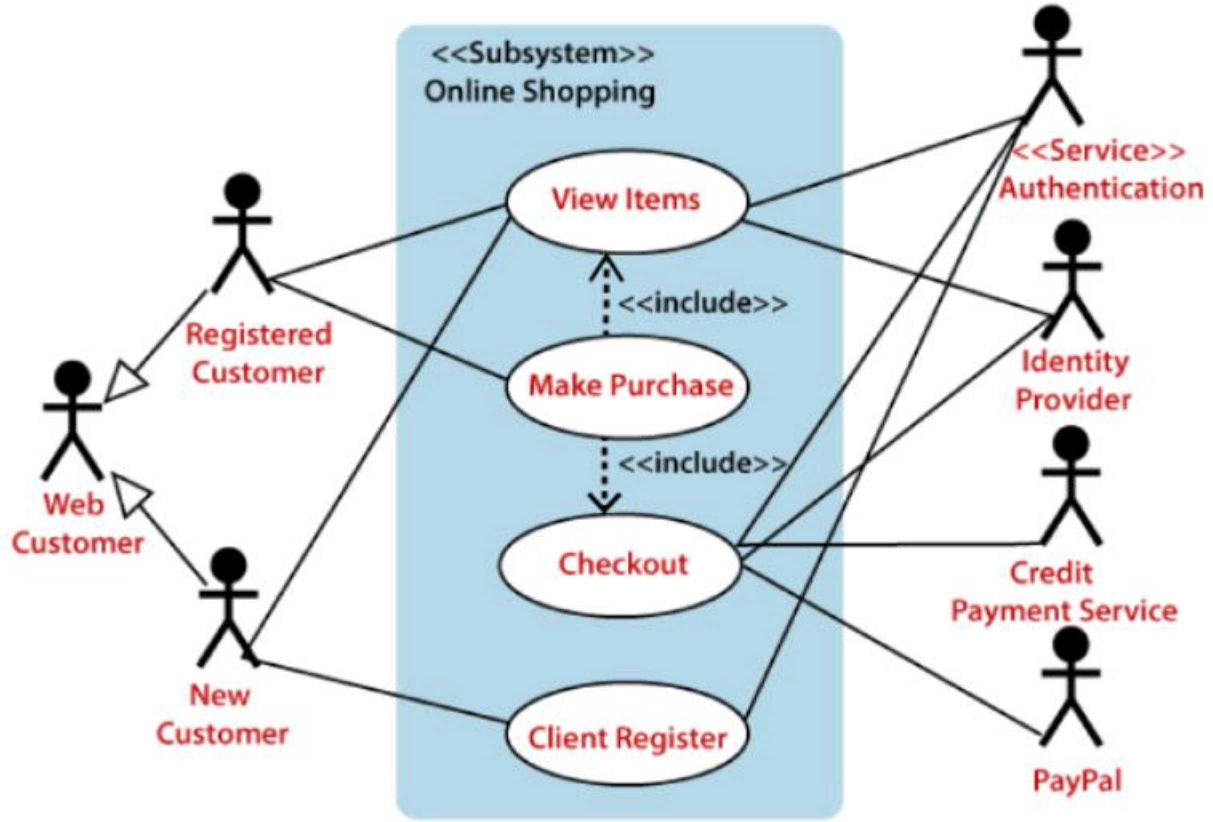
Use Case Diagram objects

•**Package-**The package is another optional element that is extremely useful in complex diagrams. Similar to class diagrams, packages are **used to group together use cases**. They are drawn like the image shown below.



Example of a Use Case Diagram

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



Example of a Use Case Diagram Conti….

The **View Items** is further extended by several use cases such as; Search Items, Browse Items, View Recommended Items, Add to Shopping Cart, Add to Wish list. All of these extended use cases provide some functions to customers, which allows them to search for an item. The View Items is further extended by several use cases such as; Search Items, Browse Items, View Recommended Items, Add to Shopping Cart, Add to Wish list. All of these extended use cases provide some functions to customers, which allows them to search for an item

Activity Diagram

An activity diagram visually presents a series of actions or flow of control in a system similar to a flowchart or a data flow diagram. Activity diagrams are often used in business process modeling. They can also describe the steps in a use case diagram. Activities modeled can be sequential and concurrent. In both cases an activity diagram will have a beginning (an initial state) and an end (a final state).

Notation of an Activity diagram

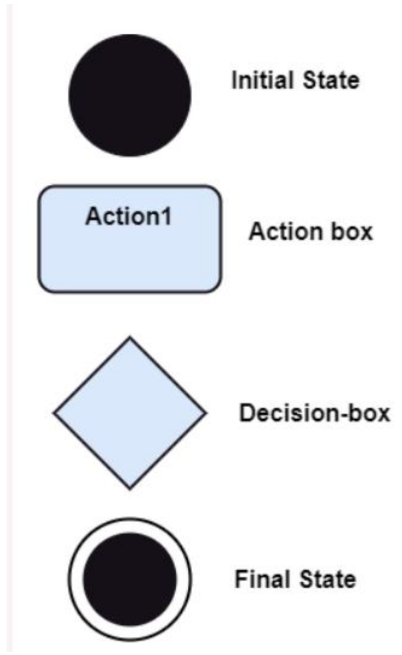
Activity diagram constitutes following notations:

Initial State: It depicts the initial stage or beginning of the set of actions.

Final State: It is the stage where all the control flows and object flows end.

Decision Box: It makes sure that the control flow or object flow will follow only one path.

Action Box: It represents the set of actions that are to be performed.



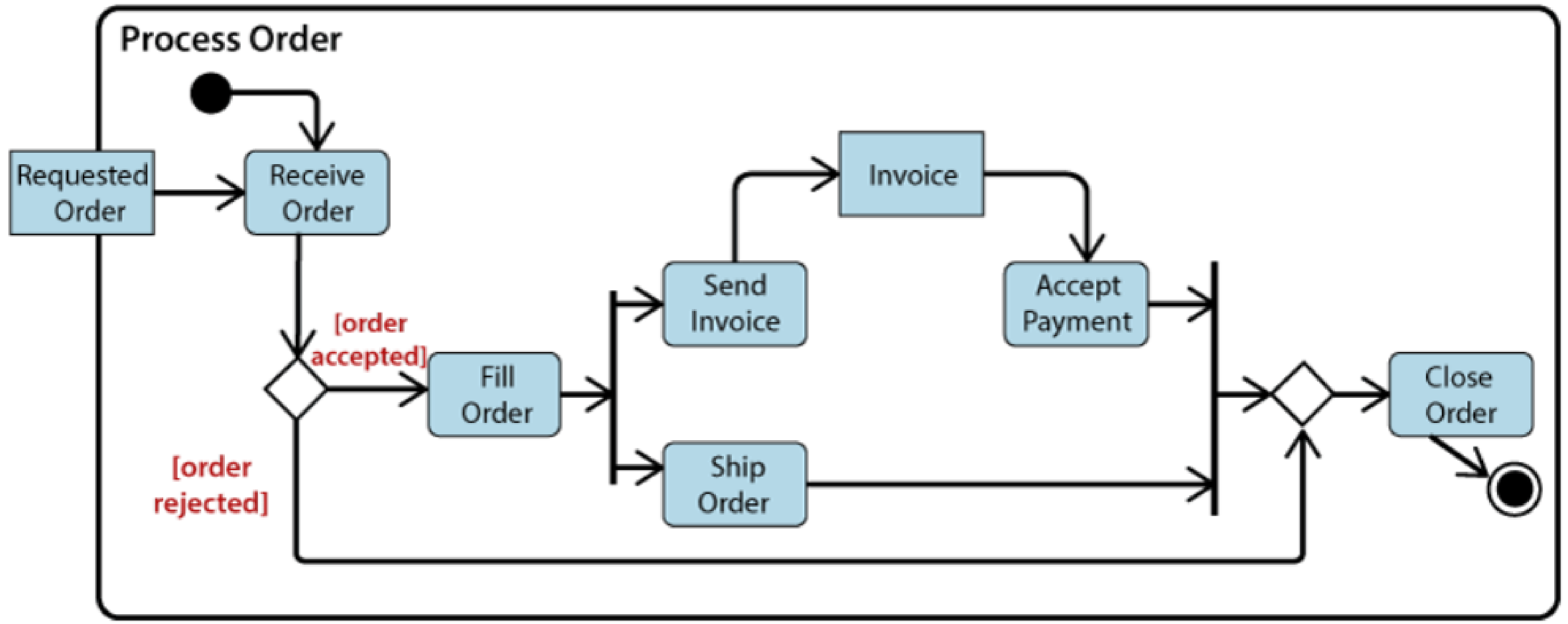
How to draw an Activity Diagram?

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

1. A meaningful name should be given to each and every activity.
2. Identify all of the constraints.
3. Acknowledge the activity associations.

Example of an Activity Diagram

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



Sequence Diagram

Sequence diagram is the most commonly used **interaction** diagram. **Interaction diagram -** An interaction diagram is used to show the **interactive behavior** of a system. Since visualizing the interactions in a system can be a cumbersome task, we use different types of interaction diagrams to capture various features and aspects of interaction in a system. **Sequence Diagrams -** A sequence diagram simply depicts interaction between objects in a sequential order i.e. the order in which these interactions take place. We can also use the terms event diagrams or event scenarios to refer to a sequence diagram. Sequence diagrams describe how and in what order the objects in a system function. These diagrams are widely used by businessmen and software developers to document and understand requirements for new and existing systems.

Sequence Diagram

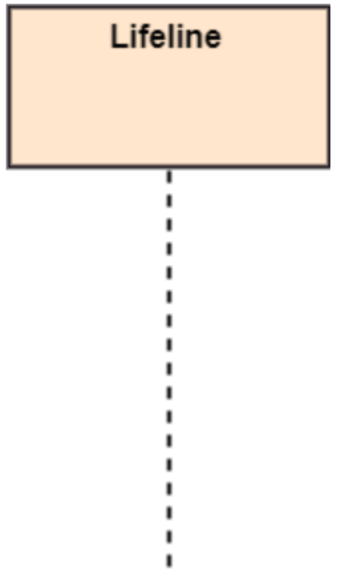
- The sequence diagram represents the flow of messages in the system and is also termed as an event diagram. It helps in envisioning several dynamic scenarios. It portrays the communication between any two lifelines as a time-ordered sequence of events, such that these lifelines took part at the run time. In UML, the lifeline is represented by a vertical bar, whereas the message flow is represented by a vertical dotted line that extends across the bottom of the page. It incorporates the iterations as well as branching.

Purpose of a Sequence Diagram

1. To model high-level interaction among active objects within a system.
2. To model interaction among objects inside a collaboration realizing a use case.
3. It either models generic interactions or some certain instances of interaction.

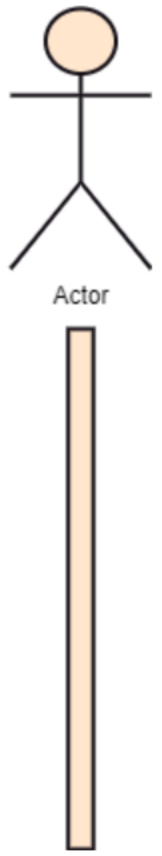
Notations of a Sequence Diagram

- **Lifeline**
  - An individual participant in the sequence diagram is represented by a lifeline. It is positioned at the top of the diagram.



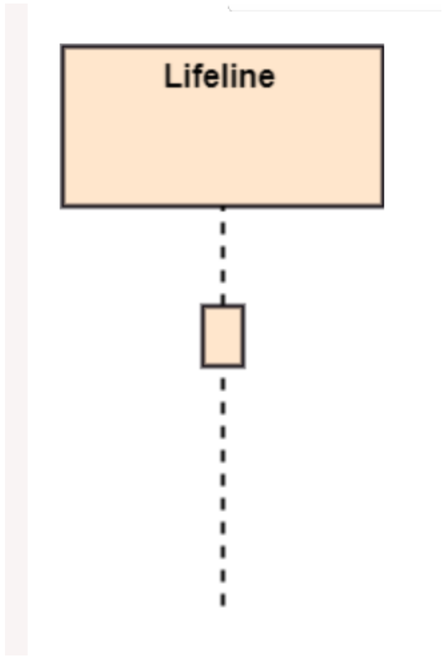
Notations of a Sequence Diagram

- **Actor**
  - A role played by an entity that interacts with the subject is called as an actor. It is out of the scope of the system. It represents the role, which involves human users and external hardware or subjects. An actor may or may not represent a physical entity, but it purely depicts the role of an entity. Several distinct roles can be played by an actor or vice versa.



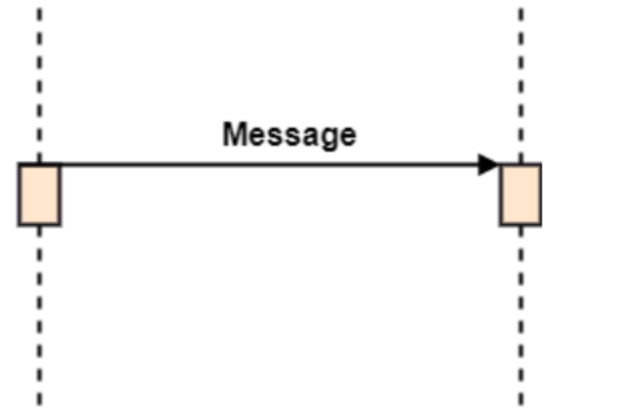
Notations of a Sequence Diagram

- **Activation**
  - It is represented by a thin rectangle on the lifeline. It describes that time period in which an operation is performed by an element, such that the top and the bottom of the rectangle is associated with the initiation and the completion time, each respectively.



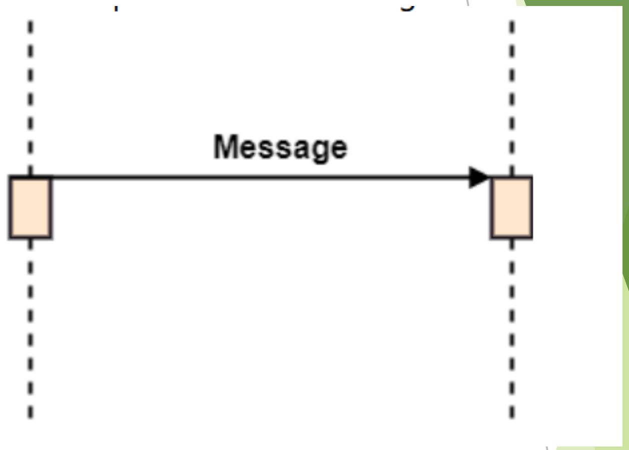
Notations of a Sequence Diagram

- **Messages**
- The messages depict the interaction between the objects and are represented by arrows. They are in the sequential order on the lifeline. The core of the sequence diagram is formed by messages and lifelines.
- Following are types of messages enlisted below:
- **Call Message:** It defines a particular communication between the lifelines of an interaction, which represents that the target lifeline has invoked an operation.

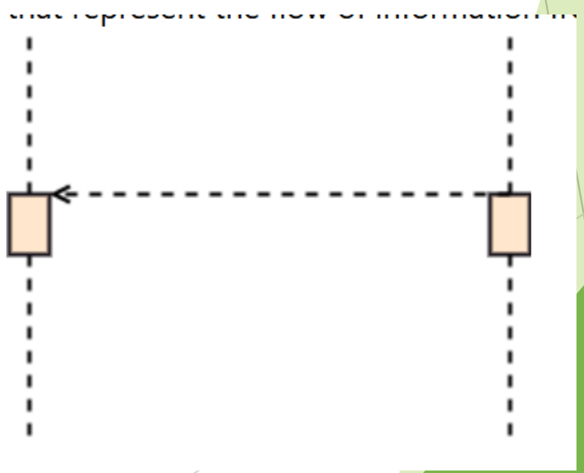


Notations of a Sequence Diagram

- **Call Message:** It defines a particular communication between the lifelines of an interaction, which represents that the target lifeline has invoked an operation.

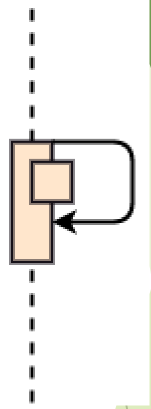


Return Message: It defines a particular communication between the lifelines of interaction that represent the flow of information from the receiver of the corresponding caller message.

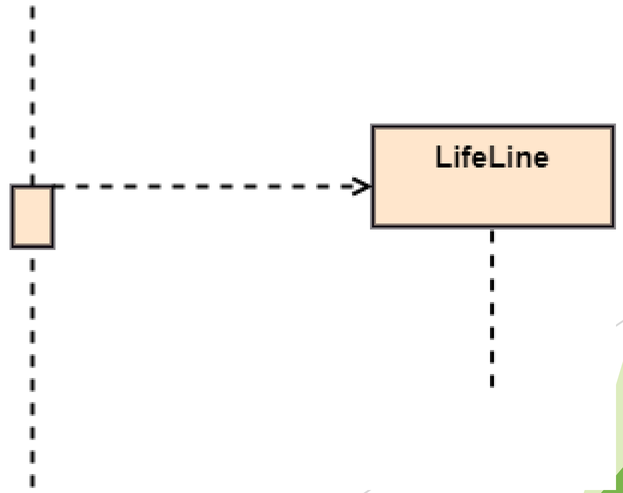


Notations of a Sequence Diagram

- **Recursive Message:** A self message sent for recursive purpose is called a recursive message. In other words, it can be said that the recursive message is a special case of the self message as it represents the recursive calls.

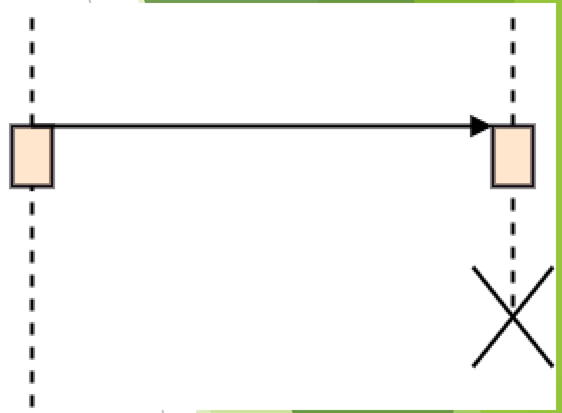


Create Message: It describes a communication, particularly between the lifelines of an interaction describing that the target (lifeline) has been instantiated.

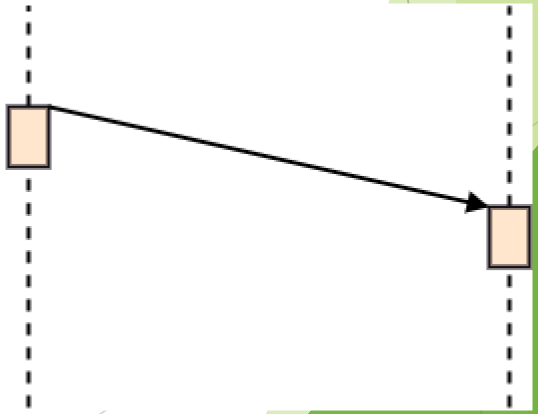


Notations of a Sequence Diagram

- **Destroy Message:** It describes a communication, particularly between the lifelines of an interaction that depicts a request to destroy the lifecycle of the target.

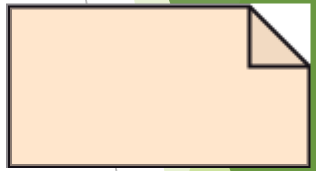


Duration Message: It describes a communication particularly between the lifelines of an interaction, which portrays the time passage of the message while modeling a system.



Notations of a Sequence Diagram

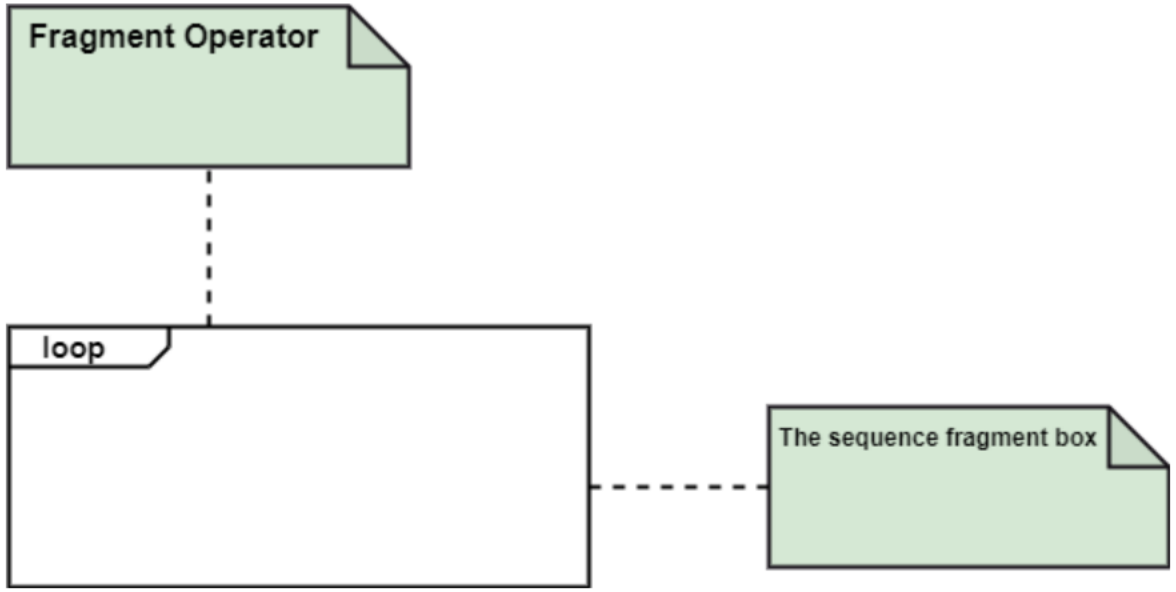
- **Note**
  - A note is the capability of attaching several remarks to the element. It basically carries useful information for the modelers.



Notations of a Sequence Diagram

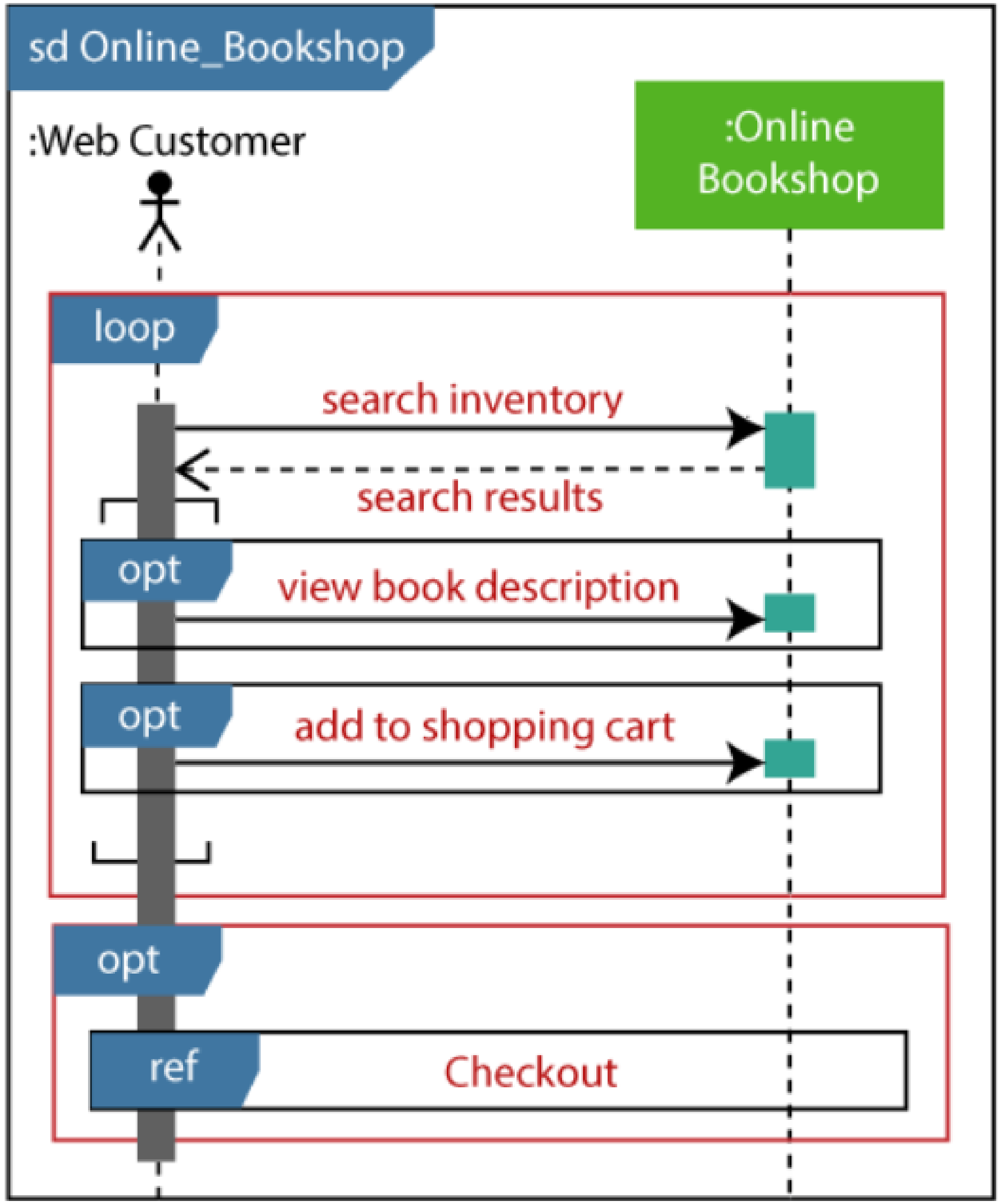
- **Sequence Fragments**

1. Sequence fragments have been introduced by UML 2.0, which makes it quite easy for the creation and maintenance of an accurate sequence diagram.
2. It is represented by a box called a combined fragment, encloses a part of interaction inside a sequence diagram.
3. The type of fragment is shown by a fragment operator.

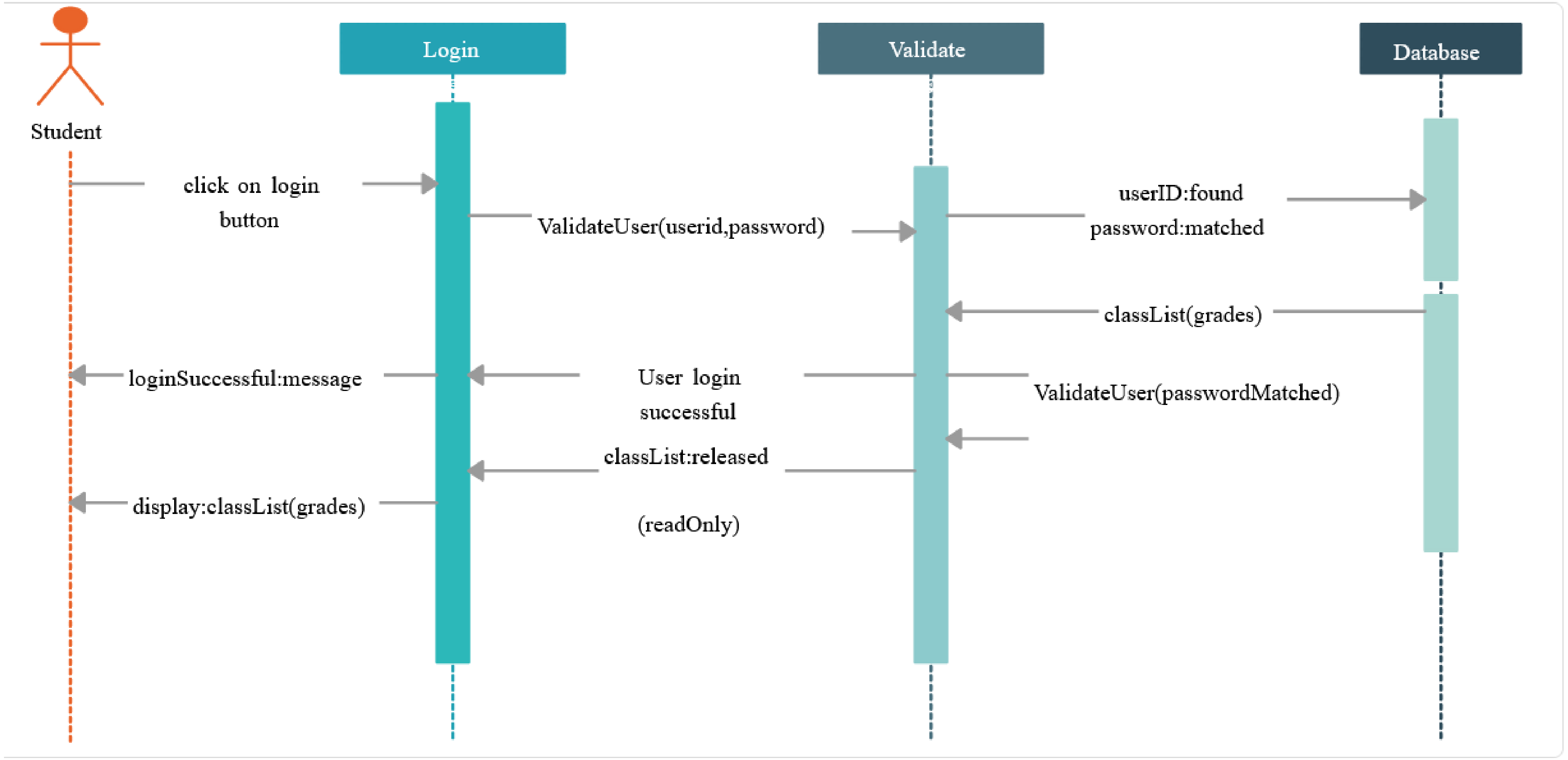


Example of a Sequence Diagram

- An example of a high-level sequence diagram for online bookshop is given below.
- Any online customer can search for a book catalog, view a description of a particular book, add a book to its shopping cart, and do checkout.



Student Login Sequence Diagram Example



Sequence Diagram

Benefits of a Sequence Diagram

1. It explores the real-time application.
2. It depicts the message flow between the different objects.
3. It has easy maintenance.
4. It is easy to generate.
5. Implement both forward and reverse engineering.
6. It can easily update as per the new change in the system.

The drawback of a Sequence Diagram

1. In the case of too many lifelines, the sequence diagram can get more complex.
2. The incorrect result may be produced, if the order of the flow of messages changes.
3. Since each sequence needs distinct notations for its representation, it may make the diagram more complex.
4. The type of sequence is decided by the type of message.

UML State Machine Diagram

- The state machine diagram is also called the Statechart or State Transition diagram, which shows the order of states underwent by an object within the system. It captures the software system's behavior. It models the behavior of a class, a subsystem, a package, and a complete system.
- It tends out to be an efficient way of modeling the interactions and collaborations in the external entities and the system. It models event-based systems to handle the state of an object. It also defines several distinct states of a component within the system. Each object/component has a specific state.

UML State Machine Diagram

- Following are the types of a state machine diagram that are given below:

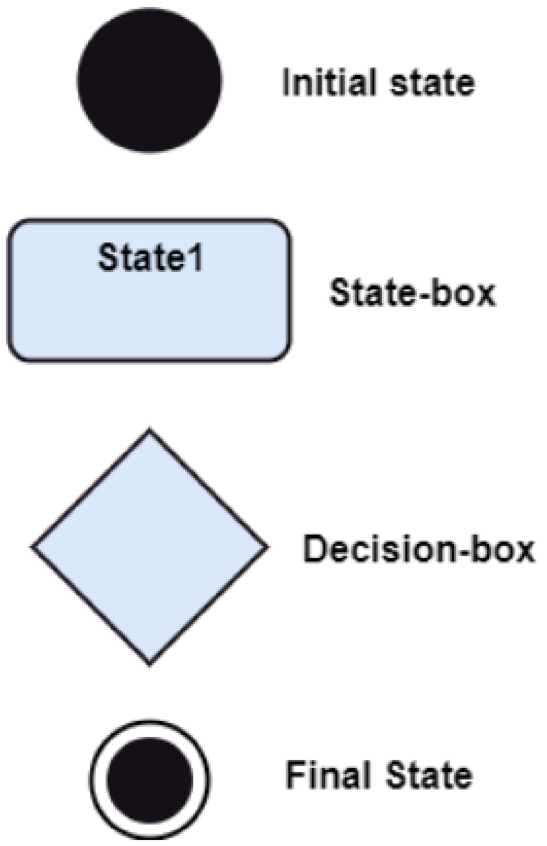
1. **Behavioral state machine**
   The behavioral state machine diagram records the behavior of an object within the system. It depicts an implementation of a particular entity. It models the behavior of the system.
2. **Protocol state machine**
   It captures the behavior of the protocol. The protocol state machine depicts the change in the state of the protocol and parallel changes within the system. But it does not portray the implementation of a particular component.

Why State Machine Diagram?

- Since it records the dynamic view of a system, it portrays the behavior of a software application. During a lifespan, an object underwent several states, such that the lifespan exist until the program is executing. Each state depicts some useful information about the object.
- t blueprints an interactive system that response back to either the internal events or the external ones. The execution flow from one state to another is represented by a state machine diagram. It visualizes an object state from its creation to its termination.
- The main purpose is to depict each state of an individual object. It represents an interactive system and the entities inside the system. It records the dynamic behavior of the system.

Notation of a State Machine Diagram

1. **Initial state:** It defines the initial state (beginning) of a system, and it is represented by a black filled circle.
2. **Final state:** It represents the final state (end) of a system. It is denoted by a filled circle present within a circle.
3. **Decision box:** It is of diamond shape that represents the decisions to be made on the basis of an evaluated guard.
4. **Transition:** A change of control from one state to another due to the occurrence of some event is termed as a transition. It is represented by an arrow labeled with an event due to which the change has ensued.
5. **State box:** It depicts the conditions or circumstances of a particular object of a class at a specific point of time. A rectangle with round corners is used to represent the state box.



Types of State

- The UML consist of three states:

1. **Simple state:** It does not constitute any substructure.
2. **Composite state:** It consists of nested states (substates), such that it does not contain more than one initial state and one final state. It can be nested to any level.
3. **Submachine state:** The submachine state is semantically identical to the composite state, but it can be reused.

How to Draw a State Machine Diagram

- The state machine diagram is used to portray various states underwent by an object. The change in one state to another is due to the occurrence of some event. All of the possible states of a particular component must be identified before drawing a state machine diagram.
- The primary focus of the state machine diagram is to depict the states of a system. These states are essential while drawing a state transition diagram. The objects, states, and events due to which the state transition occurs must be acknowledged before the implementation of a state machine diagram.
- Following are the steps that are to be incorporated while drawing a state machine diagram:

1. A unique and understandable name should be assigned to the state transition that describes the behavior of the system.
2. Out of multiple objects, only the essential objects are implemented.
3. A proper name should be given to the events and the transitions.

When to use a State Machine Diagram?

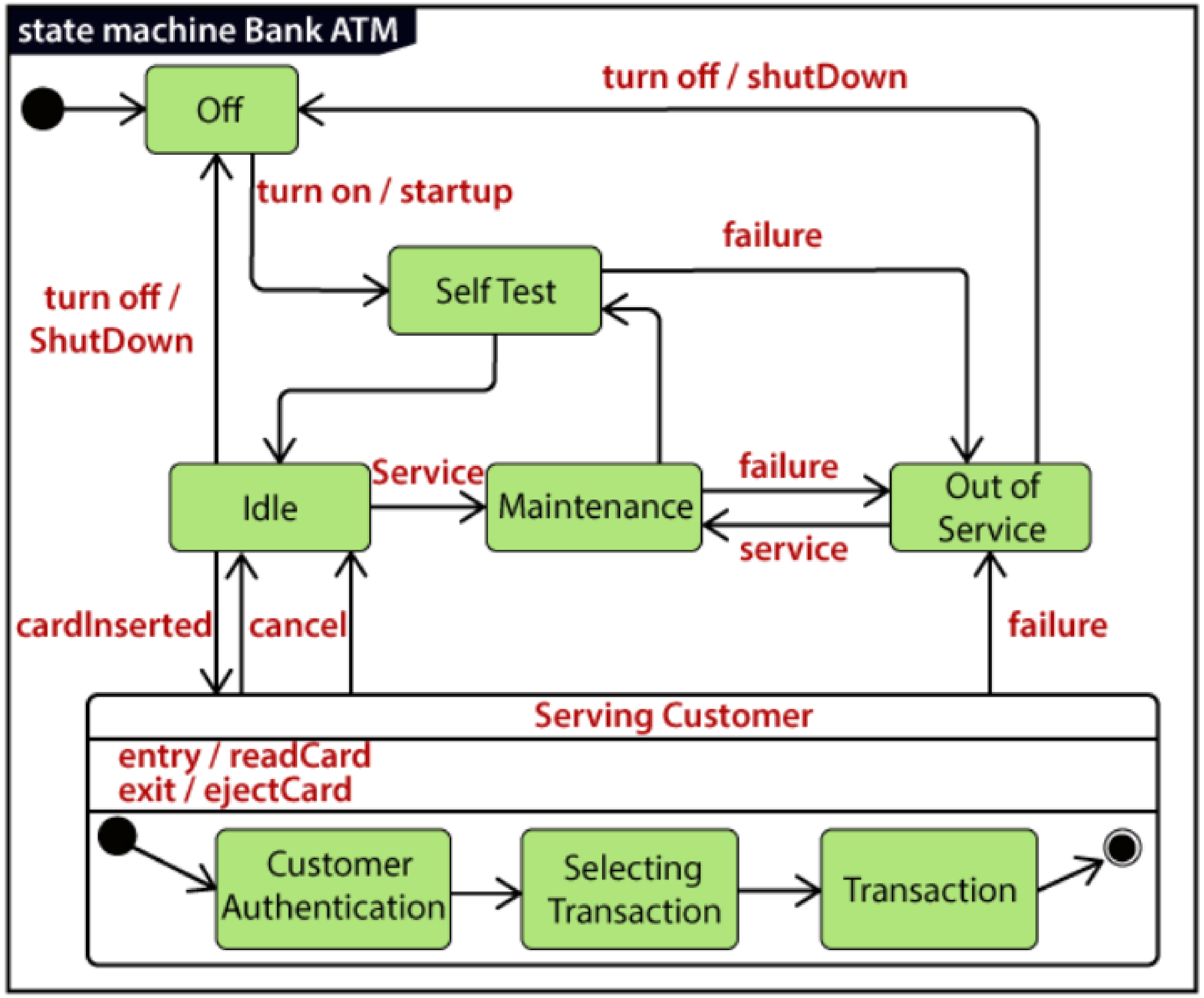
- The state machine diagram implements the real-world models as well as the object-oriented systems. It records the dynamic behavior of the system, which is used to differentiate between the dynamic and static behavior of a system.
- It portrays the changes underwent by an object from the start to the end. It basically envisions how triggering an event can cause a change within the system.
- State machine diagram is used for:

1. For modeling the object states of a system.
2. For modeling the reactive system as it consists of reactive objects.
3. For pinpointing the events responsible for state transitions.
4. For implementing forward and reverse engineering.

Example of a State Machine Diagram

- An example of a top-level state machine diagram showing Bank Automated Teller Machine (ATM) is given below.
- Initially, the ATM is turned off. After the power supply is turned on, the ATM starts performing the startup action and enters into the **Self Test** state. If the test fails, the ATM will enter into the **Out Of Service** state, or it will undergo **a triggerless transition** to the **Idle** state. This is the state where the customer waits for the interaction.
- Whenever the customer inserts the bank or credit card in the ATM's card reader, the ATM state changes from **Idle** to **Serving Customer**, the entry action **readCard** is performed after entering into **Serving Customer** state. Since the customer can cancel the transaction at any instant, so the transition from **Serving Customer** state back to the **Idle** state could be triggered by **cancel** event

- here the **Serving Customer** is a composite state with sequential substates that are **Customer Authentication, Selecting Transaction,** and **Transaction**.
- **Customer Authentication** and **Transaction** are the composite states itself is displayed by a hidden decomposition indication icon. After the transaction is finished, the **Serving Customer** encompasses a triggerless transition back to the **Idle** state. On leaving the state, it undergoes the exit action **ejectCard** that discharges the customer card.


state machine Bank ATM
Off
turn off / shutDown
turn on / startup
failure
Self Test
turn off / ShutDown
Idle
Service
Maintenance
failure
Out of Service
service
cardInserted
cancel
failure
Serving Customer
entry / readCard
exit / ejectCard
Customer Authentication
Selecting Transaction
Transaction