Index

Lab 2 : Draw the use case diagram for atm banking.

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

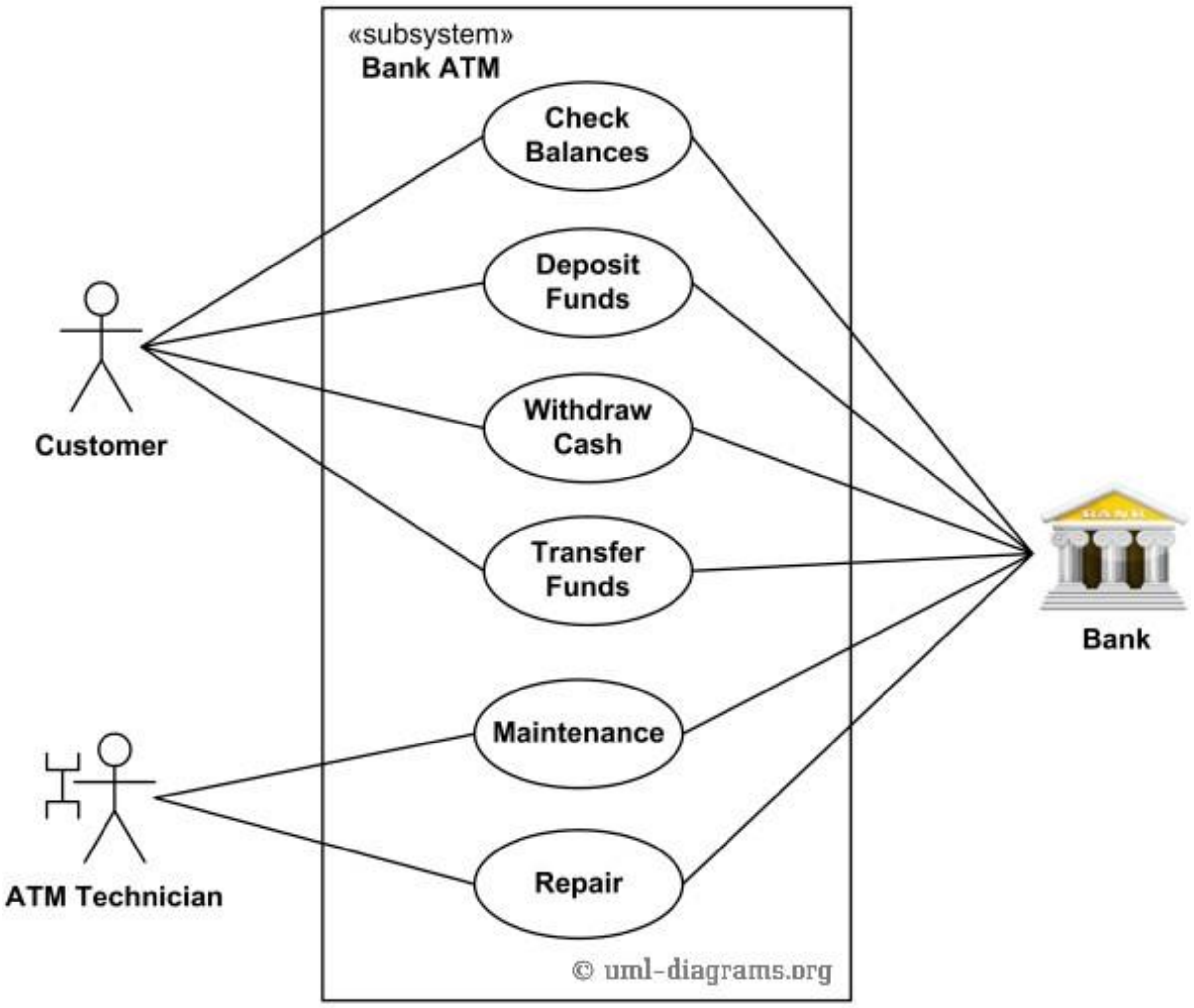
We can say that use cases are nothing but the system functionalities written in an organized manner. The second thing which is relevant to use cases are the actors. Actors can be defined as something that interacts with the system.

Actors can be a human user, some internal applications, or may be some external applications. When we are planning to draw a use case diagram, we should have the following items identified.

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

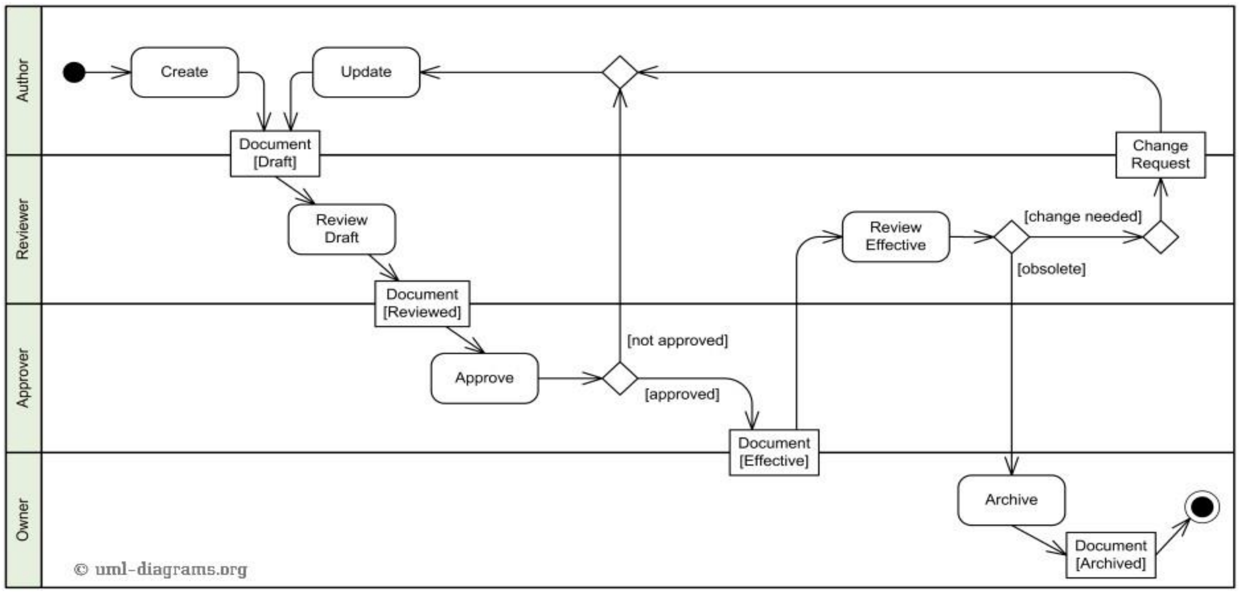
Use case diagrams are drawn to capture the functional requirements of a system. After identifying the above items, we have to use the following guidelines to draw an efficient use case diagram

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


«subsystem»
Bank ATM
Check Balances
Deposit Funds
Withdraw Cash
Transfer Funds
Maintenance
Repair
Customer
ATM Technician
Bank
BANK
© uml-diagrams.org

Lab 3: Draw UML Activity Diagram for Document Management Process activity

UML activity diagram describing a **Document Management Process**. Some kind of formal and properly communicated document management process is usually required in any major corporation especially under a regulatory compliance.
A **document** goes through different **state** or stages - it is created, reviewed, updated, approved, and at some point archived. Different roles participating in this process are **Author**, **Reviewer**, **Approver**, and **Owner**. These roles are represented on the diagram by **partitions** rendered as horizontal "**swimlanes**".



An example of Document Management Process activity.

This activity diagram shows responsibilities of different roles and flow or sequence of document changes. Alternative type of diagram - **state machine diagram** - could also be used in this case to show how document changes its state over time.
Note, that the **Document** object is not the only object node shown on this activity diagram. There is also another object - **Change Request**, an object which is used to pass changes to the document requested by **Reviewer**. State diagram for the Document will only show the document states and transitions, so activity diagram is useful when different roles and several object nodes are involved

Lab 4 : Draw Class Diagram Library Management System.

Classes of Library Management System :

- **Library Management System class –**
  It manages all operations of Library Management System. It is central part of organization for which software is being designed.
- **User Class –**
  It manages all operations of user.
- **Librarian Class –** It manages all operations of Librarian.
- **Book Class –**
  It manages all operations of books. It is basic building block of system.
- **Account Class –**
  It manages all operations of account.
- **Library database Class –**
  It manages all operations of library database.
- **Staff Class –**
  It manages all operations of staff.
- **Student Class –**
  It manages all operations of student.

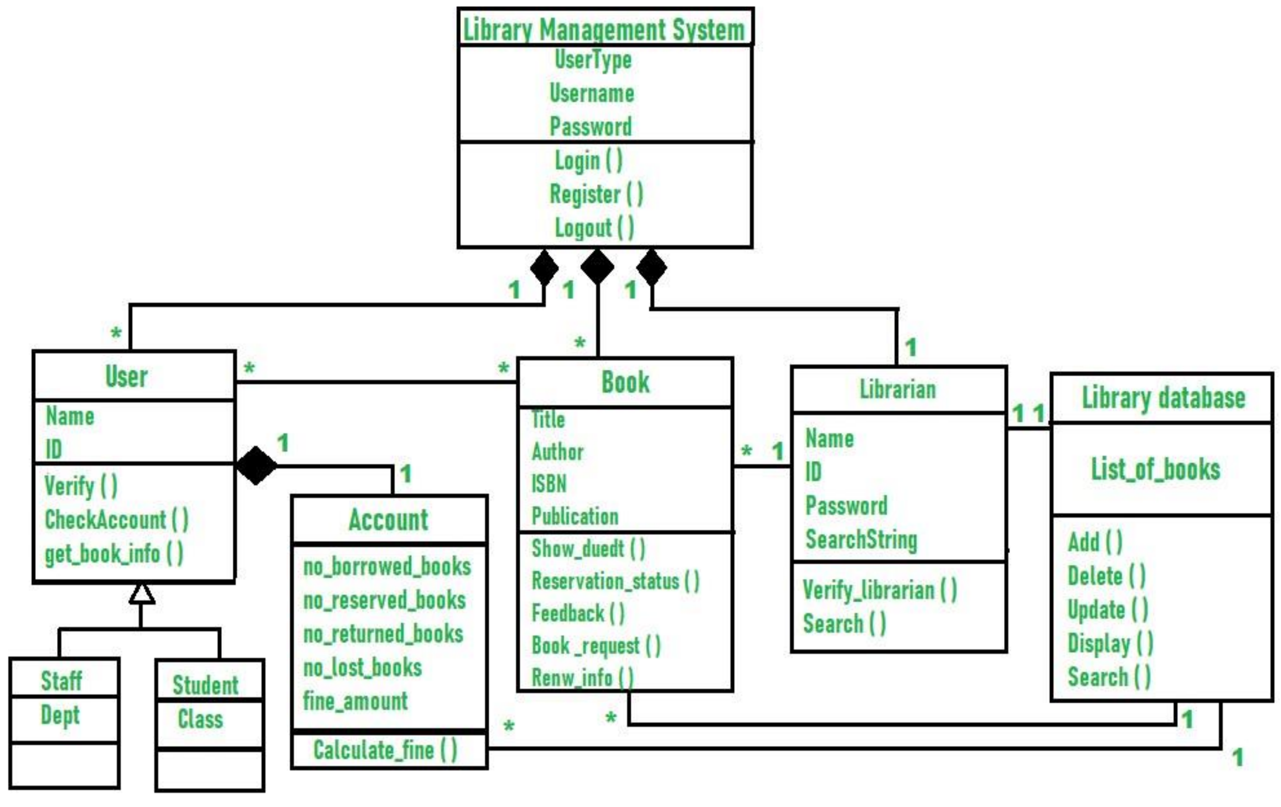
Attributes of Library Management System :

- **Library Management System Attributes –**
  UserType, Username, Password
- **User Attributes –** Name, Id
- **Librarian Attributes –**
  Name, Id, Password, SearchString
- **Book Attributes –**
  Title, Author, ISBN, Publication
- **Account Attributes –** no_borrowed_books, no_reserved_books, no_returned_books, no_lost_books fine_amount
- **Library database Attributes –** List_of_books
- **Staff Class Attributes –** Dept
- **Student Class Attributes –** Class

Methods of Library Management System :

- **Library Management System Methods –**
  Login(), Register(), Logout()
- **User Methods –**
  Verify(), CheckAccount(), get_book_info()

- **Librarian Methods –**
  Verify_librarian(), Search()
- **Book Methods –**
  Show_duedt(), Reservation_status(), Feedback(), Book_request(), Renew_info()
- **Account Methods –** Calculate_fine()
- **Library database Methods –**
  Add(), Delete(), Update(), Display(), Search()



CLASS DIAGRAM FOR LIBRARY MANAGEMENT SYSTEM

Lab 5: Sequence diagram for online bookshop

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.

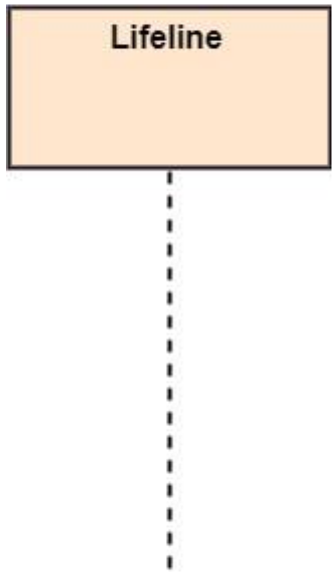
Purpose of a Sequence Diagram

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

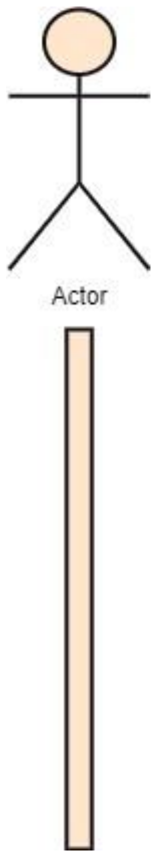
Lifeline

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



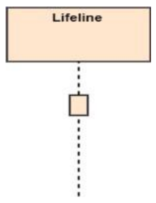
Actor

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.



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.

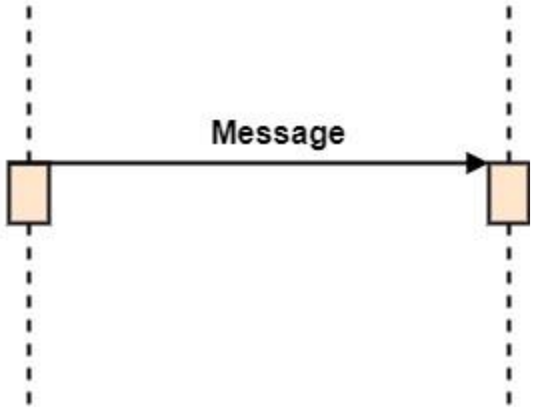


Messages

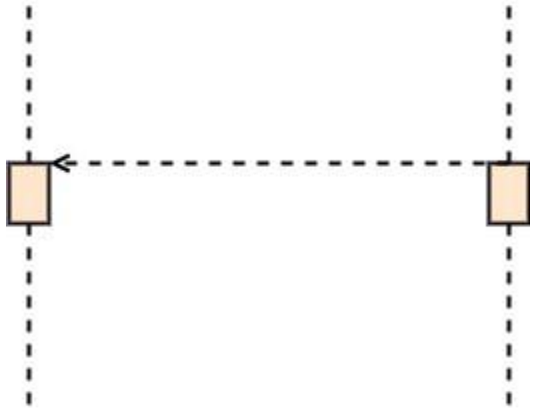
The messages depict the interaction between the objects and are represented by arrows. They are in the sequential order on the lifeline. The core of the sequence diagram is formed by messages and lifelines.

Following are types of messages enlisted below:

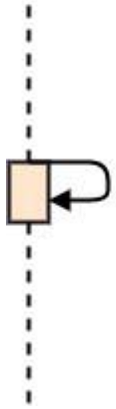
- **Call Message:** It defines a particular communication between the lifelines of an interaction, which represents that the target lifeline has invoked an operation.



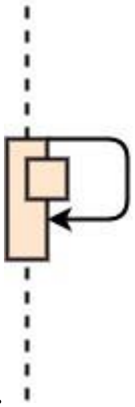
- **Return Message:** It defines a particular communication between the lifelines of interaction that represent the flow of information from the receiver of the corresponding caller message.



- **Self Message:** It describes a communication, particularly between the lifelines of an interaction that represents a message of the same lifeline, has been invoked.



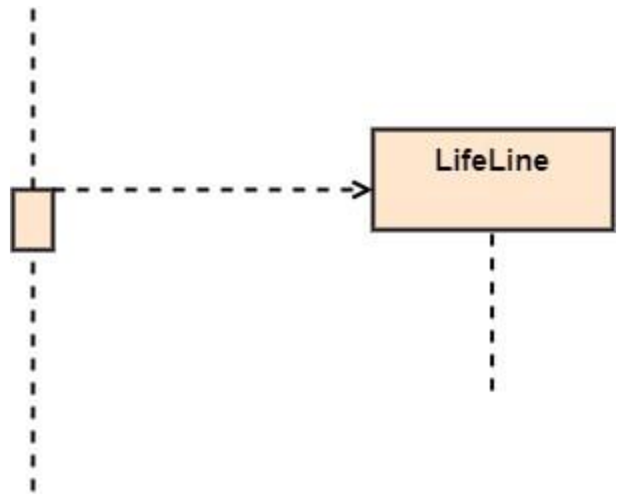
- **Recursive Message:** A self message sent for recursive purpose is called a recursive message. In other words, it can be said that the recursive message is



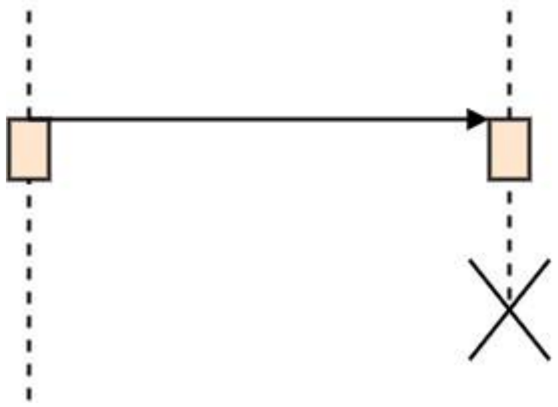
a special case of the self message as it represents the recursive calls.

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

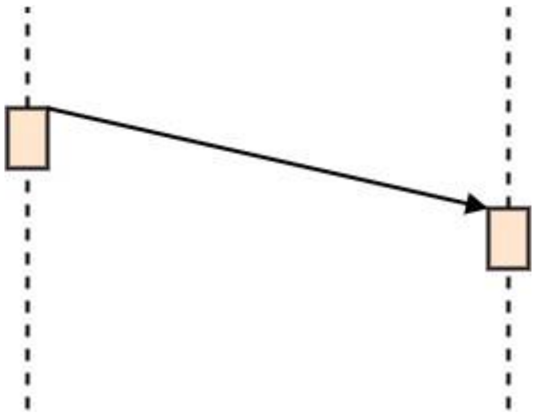
.



- **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.

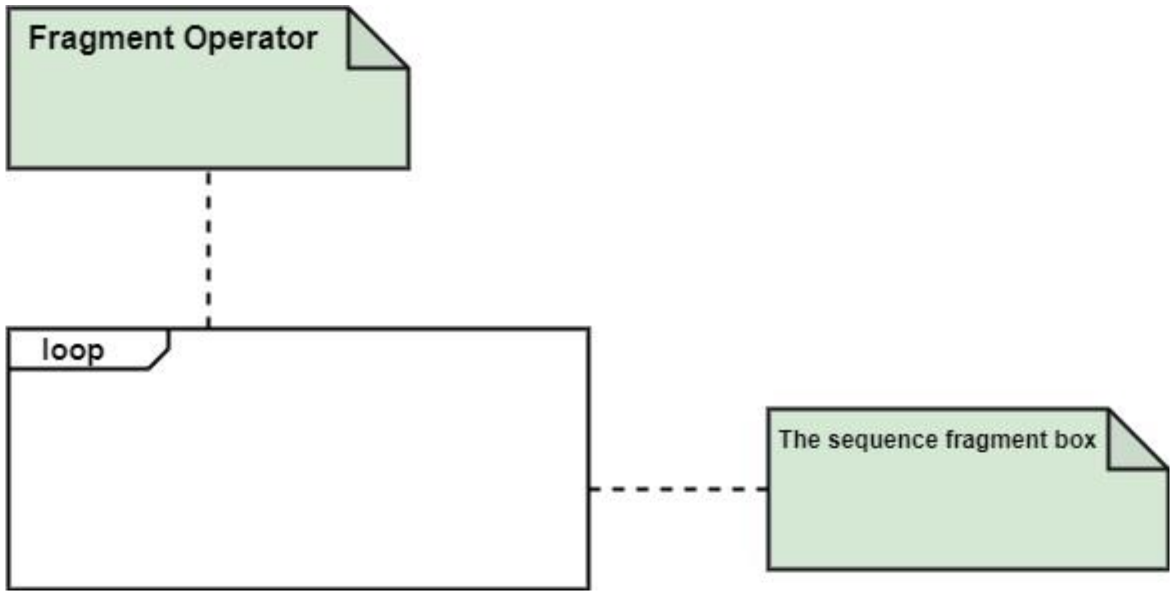


Note

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

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.



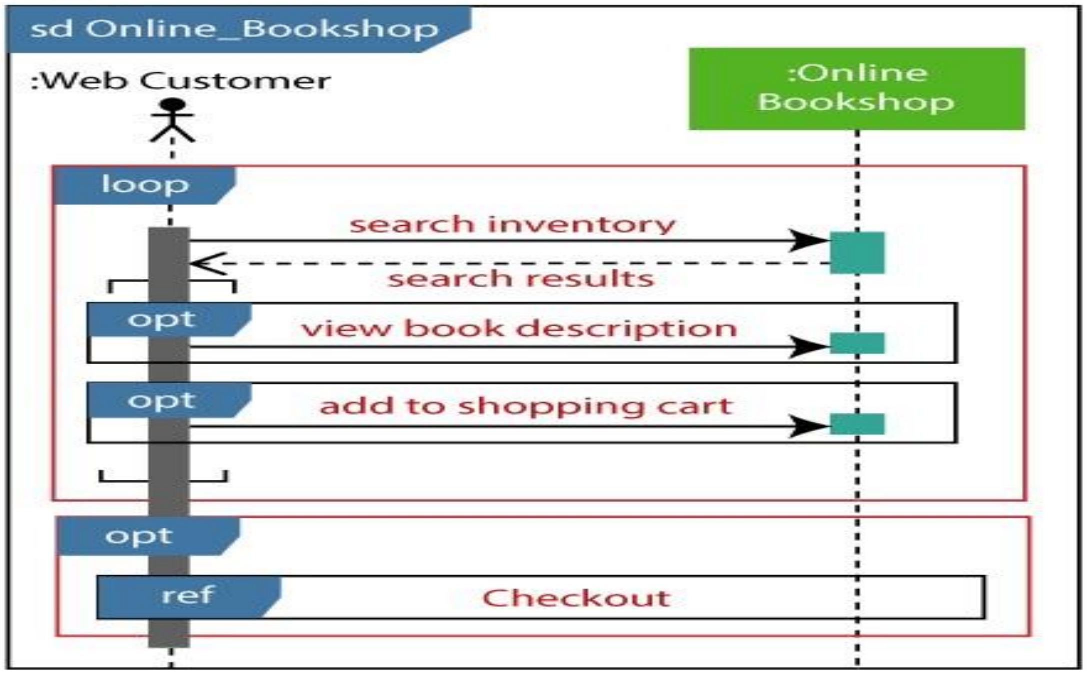
Types of fragments

Following are the types of fragments enlisted below;

Operator	Fragment Type
alt	Alternative multiple fragments: The only fragment for which the condition is true, will execute.
opt	Optional: If the supplied condition is true, only then the fragments will execute. It is similar to alt with only one trace.
par	Parallel: Parallel executes fragments.
loop	Loop: Fragments are run multiple times, and the basis of interaction is shown by the guard.

region	Critical region: Only one thread can execute a fragment at once.
neg	Negative: A worthless communication is shown by the fragment.
ref	Reference: An interaction portrayed in another diagram. In this, a frame is drawn so as to cover the lifelines involved in the communication. The parameter and return value can be explained.
sd	Sequence Diagram: It is used to surround the whole sequence diagram.

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.



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.

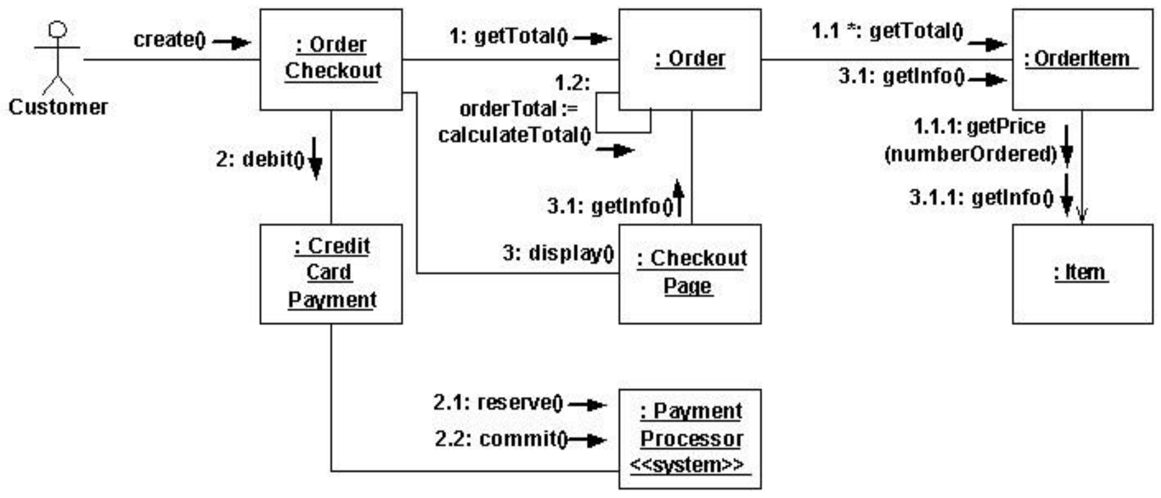
4. It is easy to generate.
5. Implement both forward and reverse engineering.
6. It can easily update as per the new change in the system.

Lab 6 : . Draw the Collaboration Diagram for Banking Communication System

The collaboration diagram is used to show the relationship between the objects in a system. Both the sequence and the collaboration diagrams represent the same information but differently. Instead of showing the flow of messages, it depicts the architecture of the object residing in the system as it is based on object-oriented programming. An object consists of several features. Multiple objects present in the system are connected to each other. The collaboration diagram, which is also known as a communication diagram, is used to portray the object's architecture in the system. Collaboration diagrams are often used to:

- Provide a birds-eye view of a collection of collaborating objects, particularly within a real-time environment.
- Allocate functionality to classes by exploring the behavioral aspects of a system.
- Model the logic of the implementation of a complex operation, particularly one that interacts with a large number of other objects.
- Explore the roles that objects take within a system, as well as the different relationships they are involved with when in those roles.

Draw



Lab 7: Draw State chart diagram for online library

Statechart diagram is one of the five UML diagrams used to model the dynamic nature of a system. They define different states of an object during its lifetime and these states are changed by events. Statechart diagrams are useful to model the reactive systems. Reactive systems can be defined as a system that responds to external or internal events.

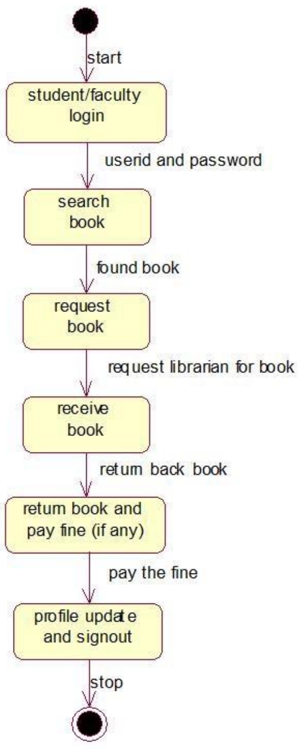
Statechart diagram describes the flow of control from one state to another state. States are defined as a condition in which an object exists and it changes when some event is triggered. The most important purpose of Statechart diagram is to model lifetime of an object from creation to termination.

Statechart diagrams are also used for forward and reverse engineering of a system. However, the main purpose is to model the reactive system.

Following are the main purposes of using Statechart diagrams −

- To model the dynamic aspect of a system.
- To model the life time of a reactive system.
- To describe different states of an object during its life time.

- Define a state machine to model the states of an object.



Lab 9 : Draw the deployment diagram of traffic control system

A deployment diagram is a UML diagram type that shows the execution architecture of a system, including nodes such as hardware or software execution environments, and the middleware connecting them.

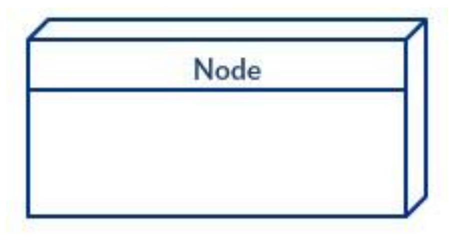
Deployment diagrams are typically used to visualize the physical hardware and software of a system. Using it you can understand how the system will be physically deployed on the hardware.

Deployment diagrams help model the hardware topology of a system compared to other UML diagram types which mostly outline the logical components of a system.

Deployment Diagram Notations

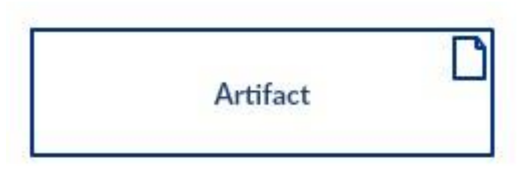
In order to draw a deployment diagram, you need to first become familiar with the following deployment diagram notations and deployment diagram elements.

Nodes



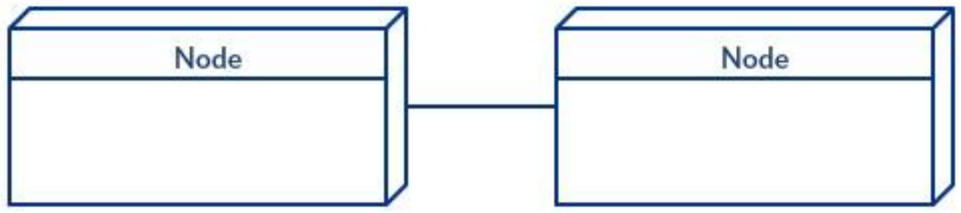
A node, represented as a cube, is a physical entity that executes one or more components, subsystems or executables. A node could be a hardware or software element.

Artifacts



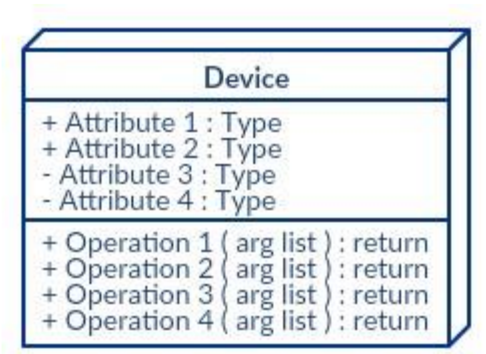
Artifacts are concrete elements that are caused by a development process. Examples of artifacts are libraries, archives, configuration files, executable files etc.

Communication Association



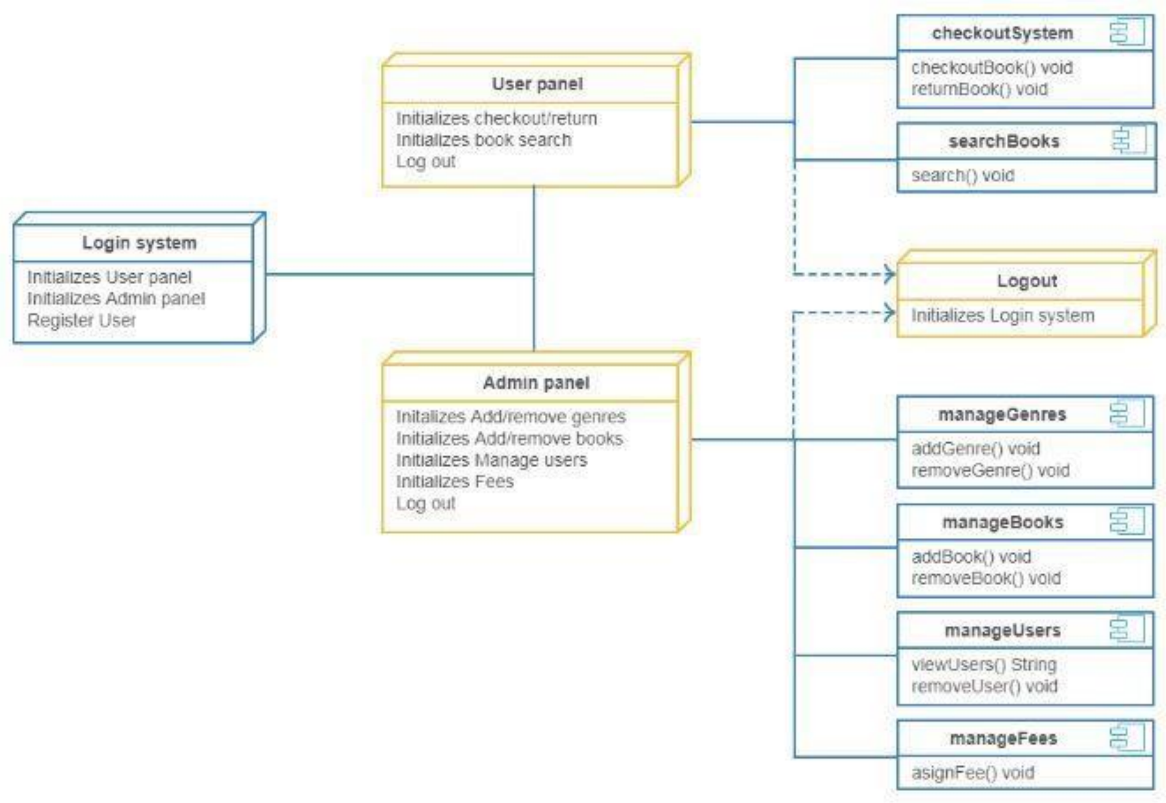
This is represented by a solid line between two nodes. It shows the path of communication between nodes.

Devices


Device
+ Attribute 1 : Type
+ Attribute 2 : Type
- Attribute 3 : Type
- Attribute 4 : Type
+ Operation 1 (arg list) : return
+ Operation 2 (arg list) : return
+ Operation 3 (arg list) : return
+ Operation 4 (arg list) : return

A device is a node that is used to represent a physical computational resource in a system. An example of a device is an application server.

Deployment Specifications


User panel
Initializes checkout/return
Initializes book search
Log out
Login system
Initializes User panel
Initializes Admin panel
Register User
Admin panel
Initalizes Add/remove genres
Initializes Add/remove books
Initializes Manage users
Initializes Fees
Log out
checkoutSystem
checkoutBook() void
returnBook() void
searchBooks
search() void
Logout
Initializes Login system
manageGenres
addGenre() void
removeGenre() void
manageBooks
addBook() void
removeBook() void
manageUsers
viewUsers() String
removeUser() void
manageFees
asignFee() void