# UNIT III DYNAMIC AND IMPLEMENTATION UML DIAGRAMS

**Dynamic Diagrams** - UML interaction diagrams - System Sequence diagram - collaboration diagram - When to use communication Diagrams - state machine diagram and Modelling - When to use state diagrams - Activity diagram - when to use activity diagrams

**Implementation Diagrams** - UML package diagram - When to use package diagrams - Component & Deployment Diagrams - when to use Component & Deployment diagram.

## UML Interaction Diagrams

object Interaction

### Overview

* The diagram is used to describe some type of interactions among the different elements in the model. This interaction is a part of dynamic behaviour of the system.
* This interactive behaviour is represented in UML by two diagram known as sequence diagram and Collaboration diagram.
* Sequence Diagram emphasizes on time sequence of messages.
* Collaboration Diagram emphasizes on the structural organization of the objects that send & receive messages.



### Purpose

* To Visualize the interactive behaviour of the system.
* Now Visualizing interaction is a difficult task.
* So, the solution is to use different types of models to capture the different aspects of the interaction i.e, why sequence & collaboration diagrams are used to capture dynamic nature but from a different angle

1. To capture dynamic behaviour of a System
2. To describe the message flow in the system
3. To describe Structural organization of the objects
4. To describe interaction among objects.

* The purpose of interaction diagrams is to capture the dynamic aspect of a system.

* To capture the dynamic aspects we need to understand what a dynamic aspect is & how it is visualized.

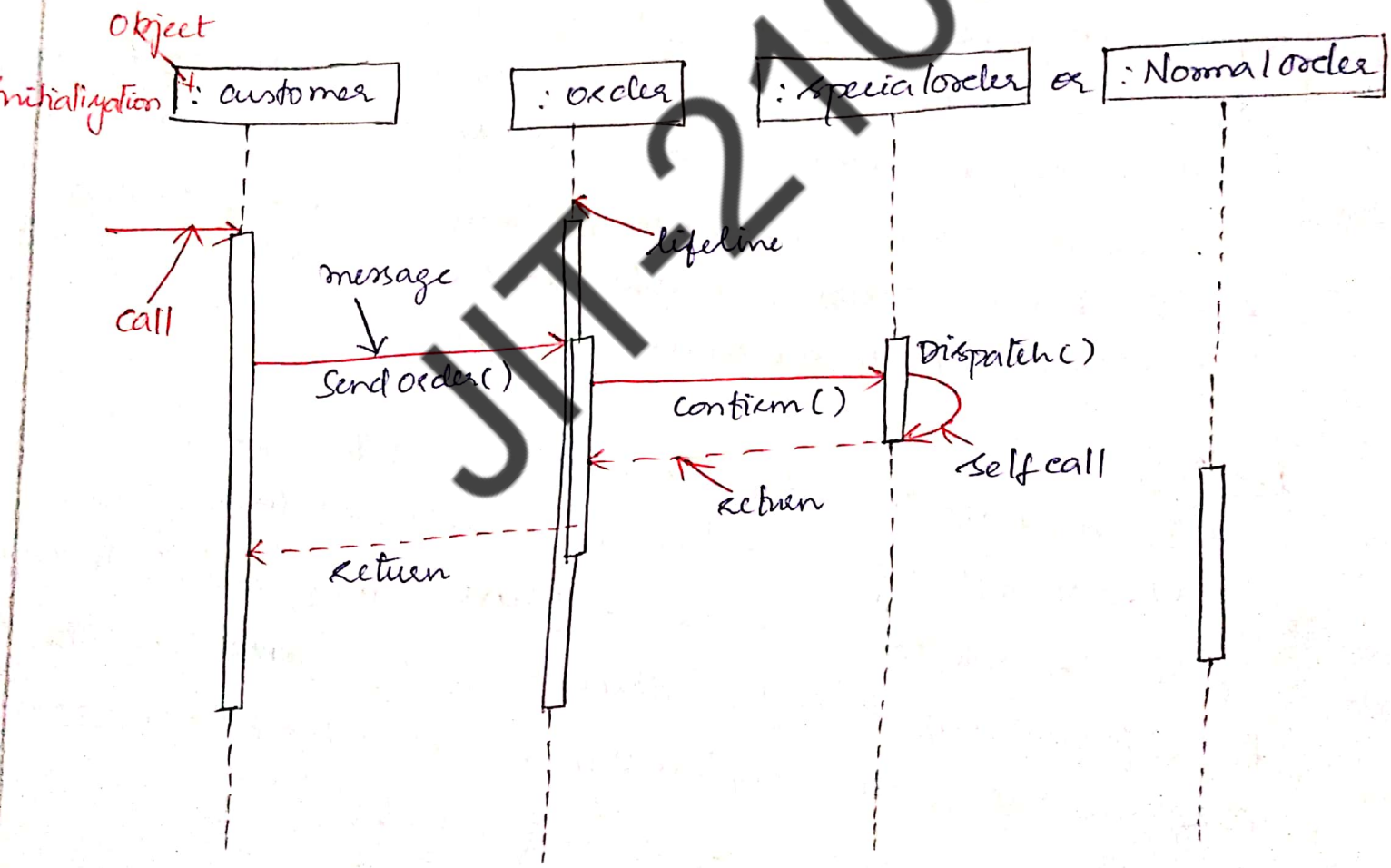
* Dynamic aspect can be defined as the snapshot of the running system at a particular moment.

* We have two types of interaction diagrams in UML.

* One is sequence diagram & other is a collaboration diagram.

## The Sequence Diagram

* The sequence diagram is having 4 objects (Customer, Order, Special Order & Normal Order)




Sequence diagram of an order management system

## System Sequence Diagram

Sequence / user 1 2 3

It is a pictorial representation of one particular scenario of use case, the events that external actors generate, their order & inter-system events

### UML perspective of Sequence Diagram

* Use cases describe how external actors interact with the s/w system.
* During this interaction an actor generate system events to a system requesting for some system operation to handle the event
* UML sequence Diagrams are thereby a notation to depict actor interactions and the operations initialed by them.

What are the three events that affects the s/w system?

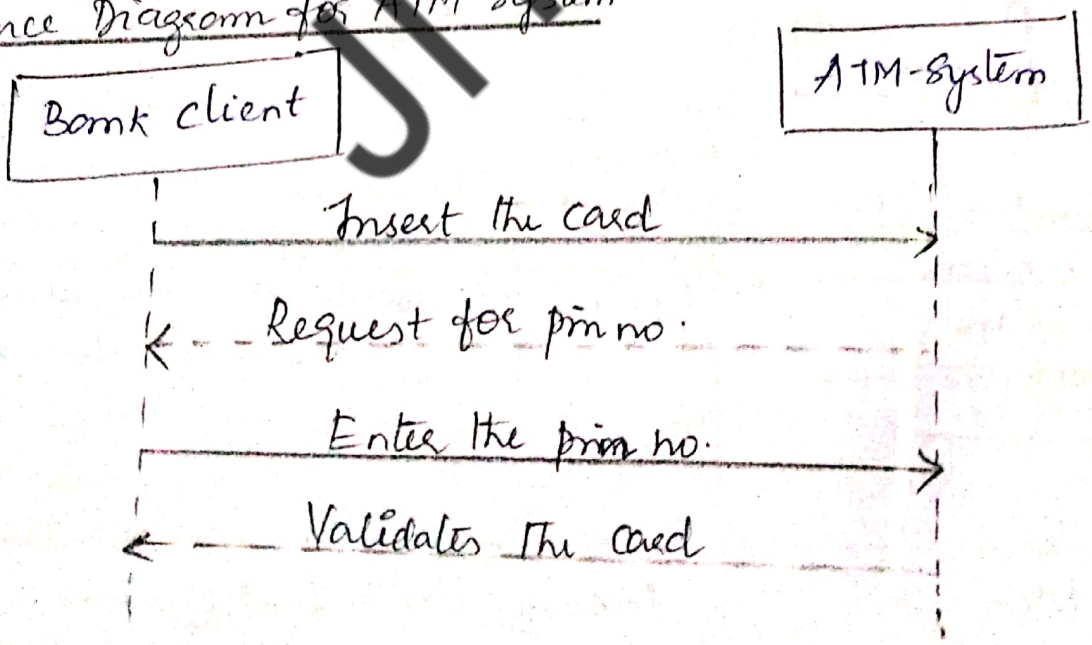
i) External Events from actors (humans or computers)

ii) Times Events

iii) Faults or Exception.

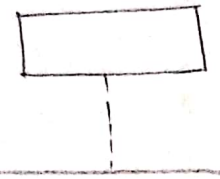
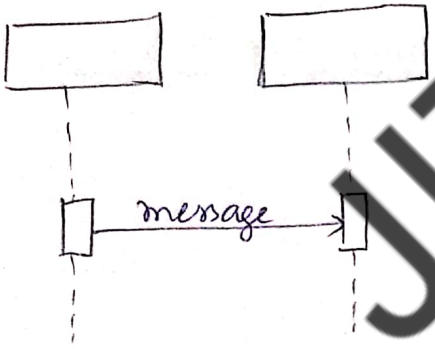
### Applying UML: Sequence Diagrams

### Sequence Diagram for ATM system

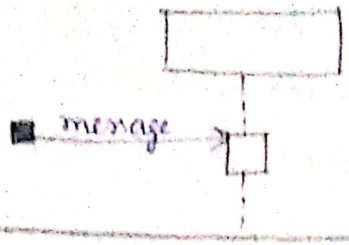



Simple sequence Diagram for ATM system card Validity

## Basic Notations Used in Sequence Diagram

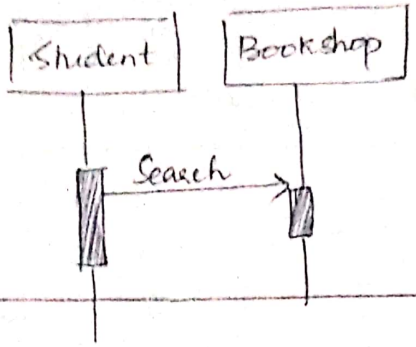
| S.No | Notation | Description |
|---|---|---|
| 1. | Lifeline  | Lifeline represents an individual participant in the interaction |
| 2. | Message   | Message is a kind of Comm. btn lifelines of an interaction |

3. Found Message



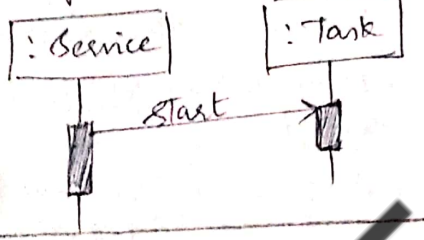

A found msg is a msg where the receiving event occurrence is known

4. Synchronous Message



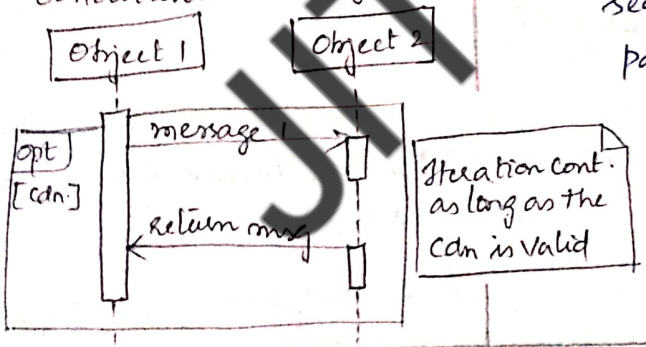

It typically represents operation call send msg & suspend execution while waiting for response

5. Asynchronous Message



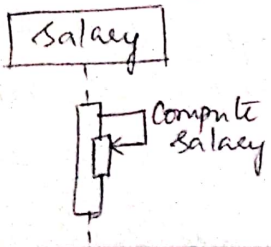

Asynchronous message send message & proceed immediately without waiting for return value

6. Conditional Message



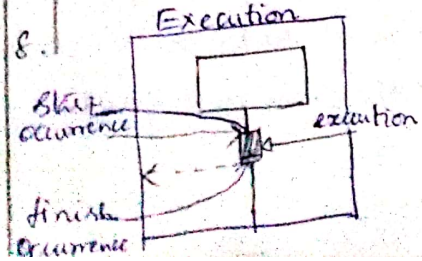

Conditional messages are used in sequence diagram Whenever particular sequence occurs based on cdn.

7. self message




Self message is a message by the Object to itself

8.




Execution (activation) is interaction fragment which represents a period in which participant's life time Executing, Sending, waiting duration of an exe is represented by two execution occurrences - Start & finish occurrence
Execution is represented as a thin grey or white rectangle on the lifeline

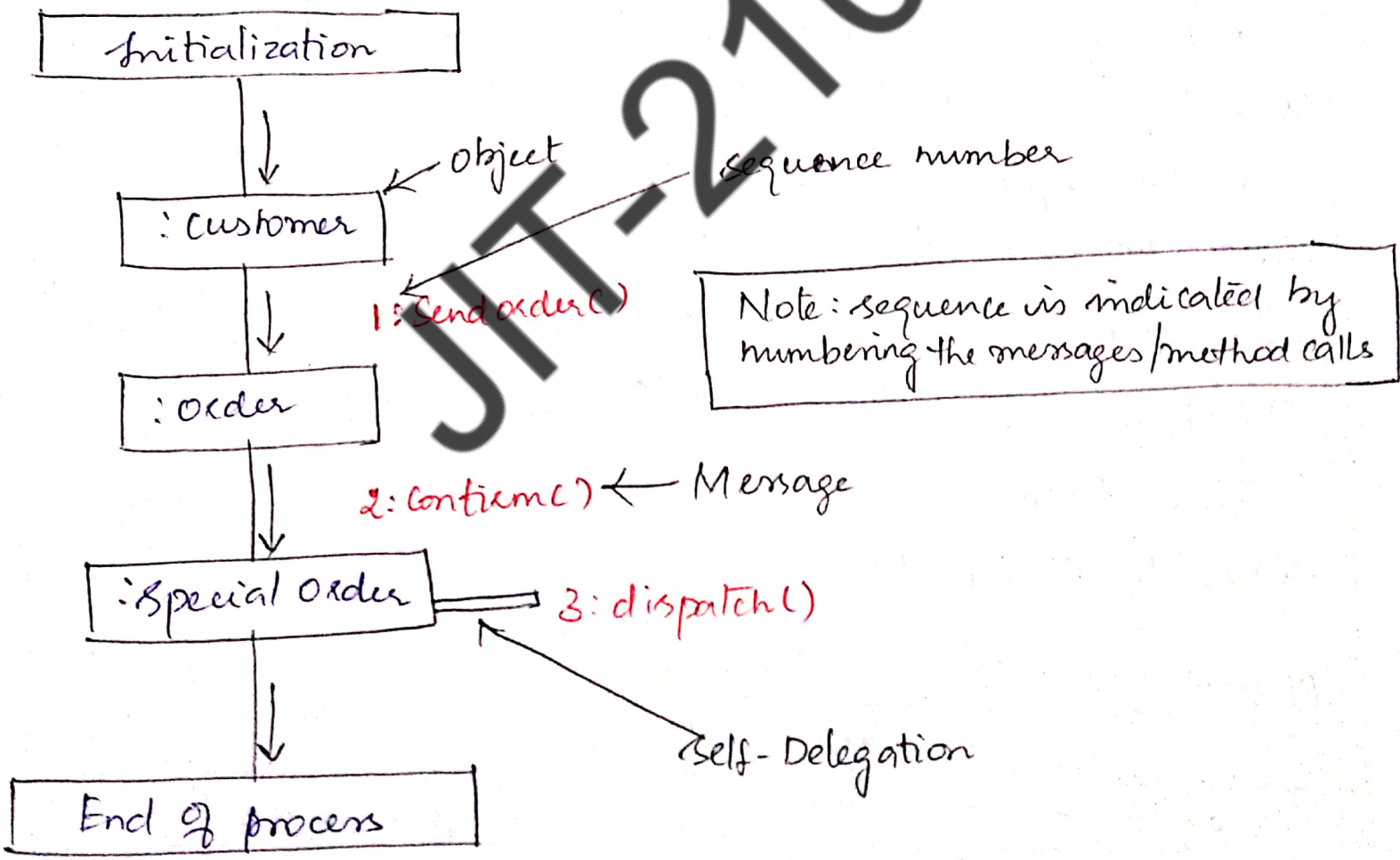
Uses:

1. To model flow of control by time sequence.
2. To model flow of control by structural organisations.
3. For forward engineering.
4. For reverse engineering.

## The Collaboration Diagram

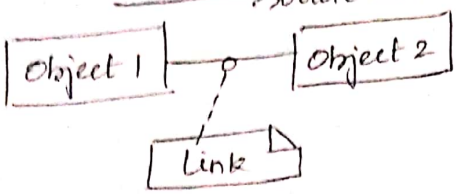
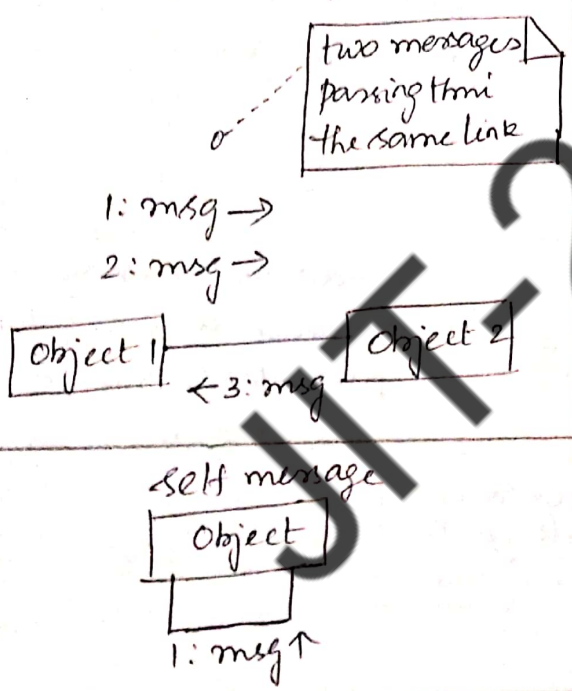
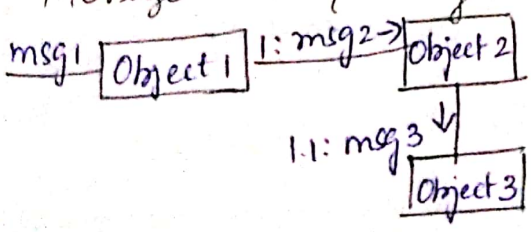
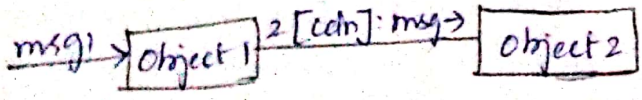
How many message pass between objects

* It shows the object organization as shown below
* Here in collaboration diagram the method call sequence is indicated by some numbering technique as shown below.
* The no. indicates how the methods are called one after another.



## Communication Diagram / collaboration

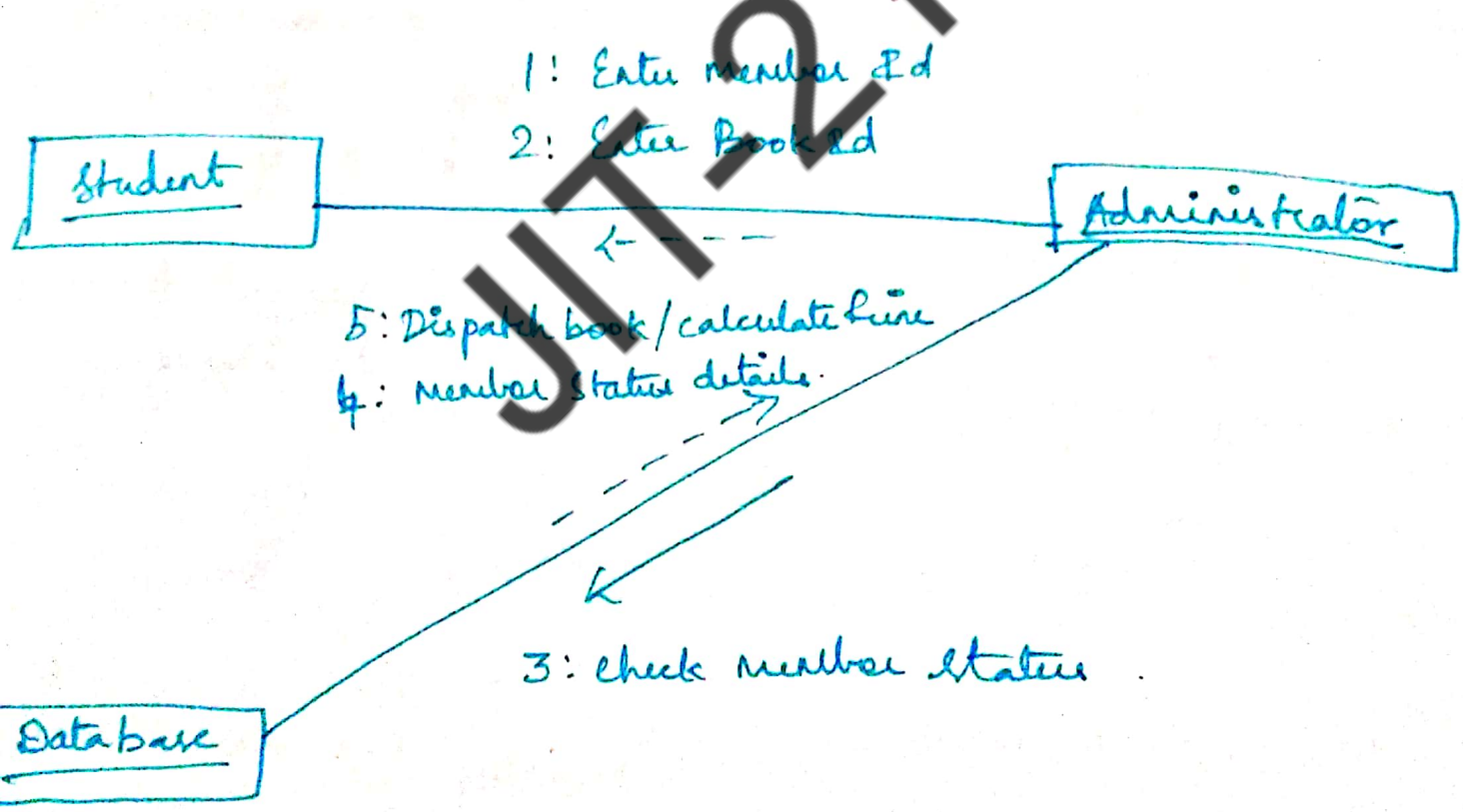
It is used to model the dynamic behavior of the system. When compared to sequence Diagram, the Communication Diagram is more focused on showing the collaboration of objects rather than the time sequence

| S.No | Notation | Description |
|---|---|---|
| 1. | Link<br>solid(line)<br>  | A link is a connection path bfn two objects. |
| 2 | Message<br>  | * Communication bfn objects takes place thru' messages.<br>* Multiple messages flow along the same link<br>* A sequence no. is added to show the sequential order of msg. |
| 3. | self message<br> | Message passed from an object to itself |
| 4. | Message No. sequencing<br>  | No's. included along with the messages indicate the order of the msg. in an interaction |
| 5. | Conditional Message<br>  | cdn. msg. indicates the cdn. for the msg. to be sent (executed) at the given nesting depth. |

Uses:

1. Communication diagram are used to shows the messages that flows from one object to another within the systems.
2. Used to track the source of the message from where it has been sent
3. Used to provide relationships and interactions among software objects.

Communication diagram – Book bank Management system

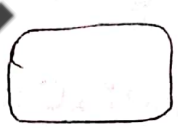


## State diagram

### Definition

* A state diagram is a diagram used to describe the behaviour of systems
* State diagram require that the system described is composed of a finite number of states
* The behaviour is analyzed and represented in series of events that could occur in one or more possible states
* Each diagram represents objects of a single class

### Basic Notation in UML State diagram

| S:NO | Element | Notation | Symbol | description |
|---|---|---|---|---|
| 1 | Initial State | Solid circle | ● | This represents the starting point of the flow. Also called pseudo state |
| 2. | State | Rounded rectangle |  | This represents the state of objects at an instant of time |
| 3. | Transition | Arrow labeled with their events | Event / Action | An arrow indicates the object of transition from one state to another. |
| 4. | Final State | Bull's eye | ⊙ | This indicates the end of the diagram |



### Events, States and Transitions

Events:

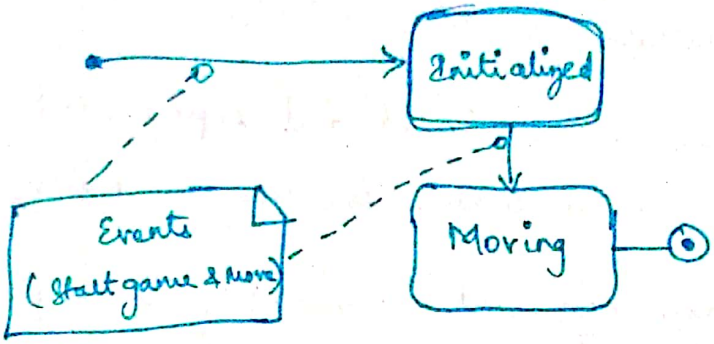
* Events are the notable occurance at a particular point in time
* Events are input to state machine.

Example:

For playing a game,

⇒ Start game is an event to reach initial state

⇒ Move is an event to reach moving state




State:

* A state is the condition of an object at a moment in time i.e, time between events.
* A state is defined as a stage in the evolution or behaviour of an object

Example:

States: Initializing
Moving.

i.e, A game is said to be in Initialized state. and when the player moves the mouse, it goes to moving state.

Transition:

* A transition is a relationship between two states which indicates that when an event occurs.
* The object moves from the previous state to the next state

Example: When the event move occurs, transition takes place from initial state to moving state

How to apply State Machine Diagram?

⇒ State independent objects.

⇒ State dependent objects.

State Independent objects:

* An object is said to be independent object, if an object responds the same way to an event

Example:

* If an Object receives a message and the corresponding method does the same thing.

⇒ Objects are State independent, if their behaviour does not depend on their particular current state.

⇒ These objects are state less or modeless

Note:

Consider state machines for state dependent objects with complex behaviour, rather than state independent objects

State dependent objects:

* An Object is said to be State dependent, if the object react differently to event depending on their state or mod

Example:

Telephone is state dependent.

Note:

* The business information systems have few complex state dependent classes. It is helpful to apply state machine modeling.

* In contrast process control, device control, protocol handle and telecommunication domains have many state dependen objects

## Modeling State dependent Objects

### Two ways to apply State machines

1. To model the behaviour of a reactive object in response to event
2. To model legal sequences of operations, protocol or language specifications
   → A formal grammer for a context free language is a kind of state machine.

Following are the list of common State-dependent objects complex reactive objects.

→ physical devices controlled by software.
   eg: phone, car, microwave oven, air conditioner.
   They have complex and rich reactions to events, and the reactions depends upon their current mode.

→ Transactions and related business objects.
   eg: Transactions → Ticket reservation, payroll calculations
   Business objects → Reservation, payment, Salary

→ Role mutators:
   * Objects that change their role.
   * A person changing roles from being a student to a staff. Each state is represented by a state



### Protocol and legal sequences:

1. communication protocols:

→ Tcp and other new protocols are easily understood with a state machine diagram.

2. UI page / window flow or Navigation

⇒ A state machine is a great tool to model UI.

⇒ Sequence between the web pages are easily understood

3 UI flow controllers or Sessions

⇒ This is fully focused on server side object that controls page flow, ongoing sessions.

example: Web applications that remembers the state with a web client and controls the transitions to new web pages.

4. Use case system operations

⇒ Treating the usecase as an object, A system machine can model system operations.

example: System operation of borrow book use case are

⇒ check status.

⇒ search.

5. Individual UI window event handling.

⇒ This is focused on the event and legal sequences from one window or form

example:

Edit paste ⇒ this action is valid only if there is anything in the clipboard to paste.

More UML State machine diagram Notations:

1. Transition Actions:

⇒ A transition action can cause an action to fire.

⇒ It is performed when performing certain transitions

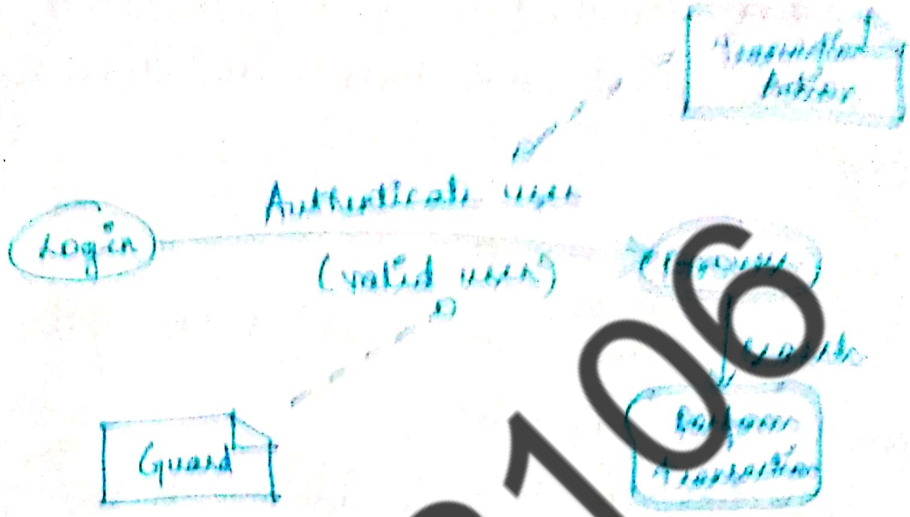
⇒ Transitions represents invocation of a method of a class in terms of software implementation.

2. Guard / conditional guard

A Guard condition is a boolean condition that is evaluated when a transition initiates

→ A transition occurs when the Guard condition is evaluated to be true

Example: Transition Action and Guard

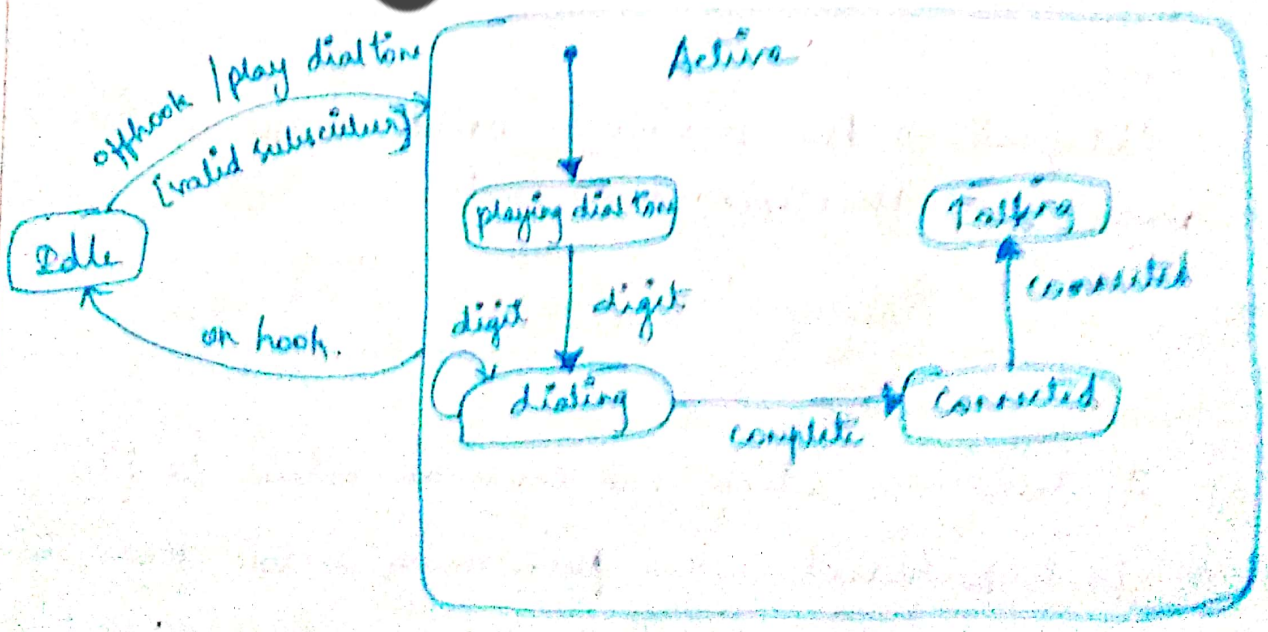



3. Nested states

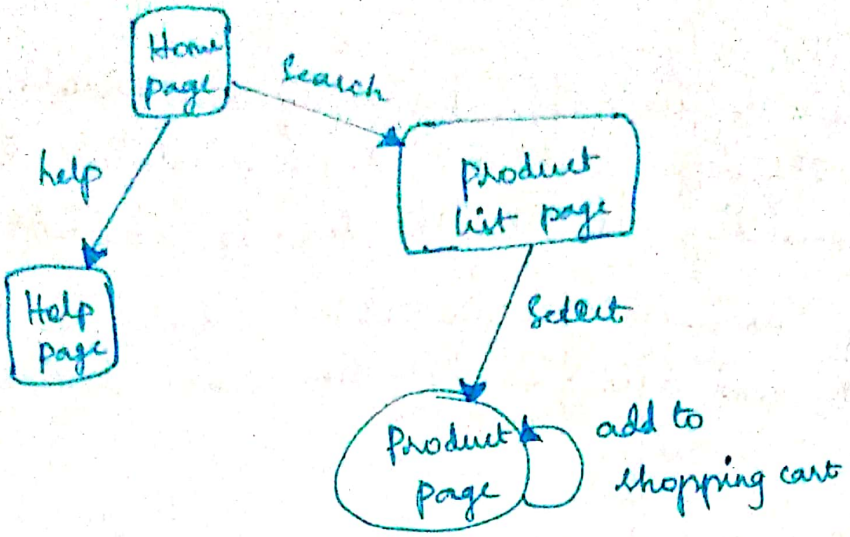
→ States that contain other states are known as Nested states

→ It inherits the transitions of its superstate

Example: Nested states

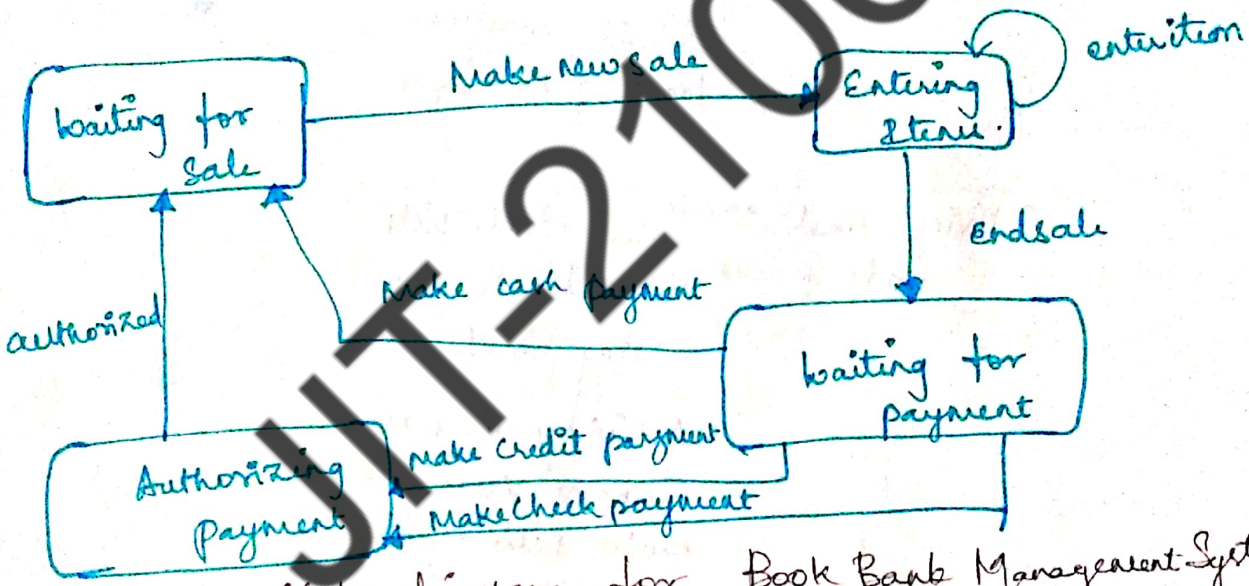


Example: UI Navigation Modeling with State machines



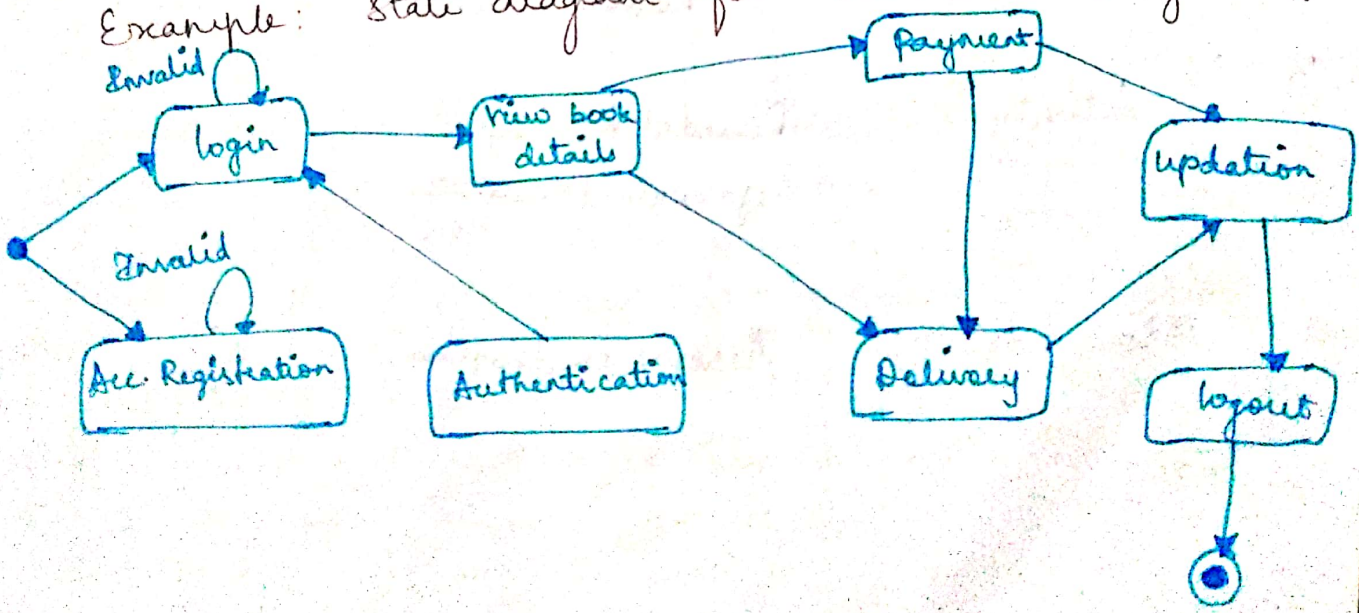

Next gen use case State Machine diagram

State machine for process Sale use case.





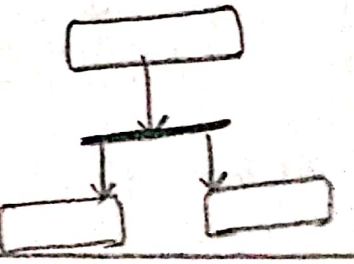
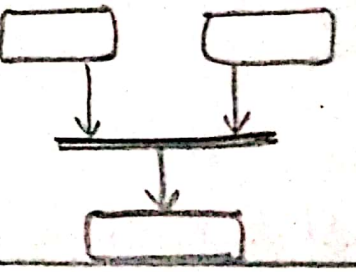
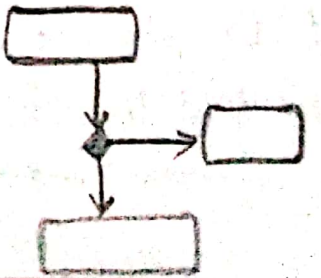
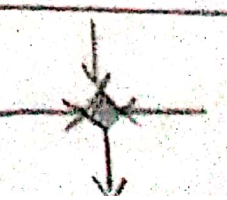

Example: State diagram for Book Bank Management System

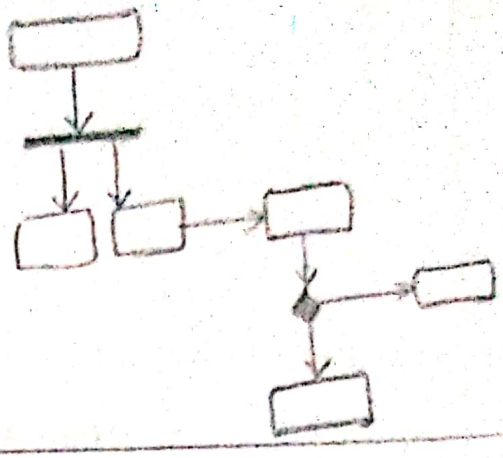
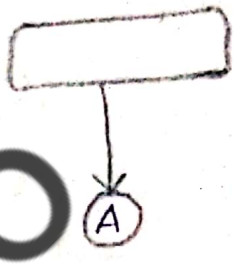


# ACTIVITY DIAGRAM

An Activity Diagram is to provide a view of flows and what is going on inside a use case or among several classes.

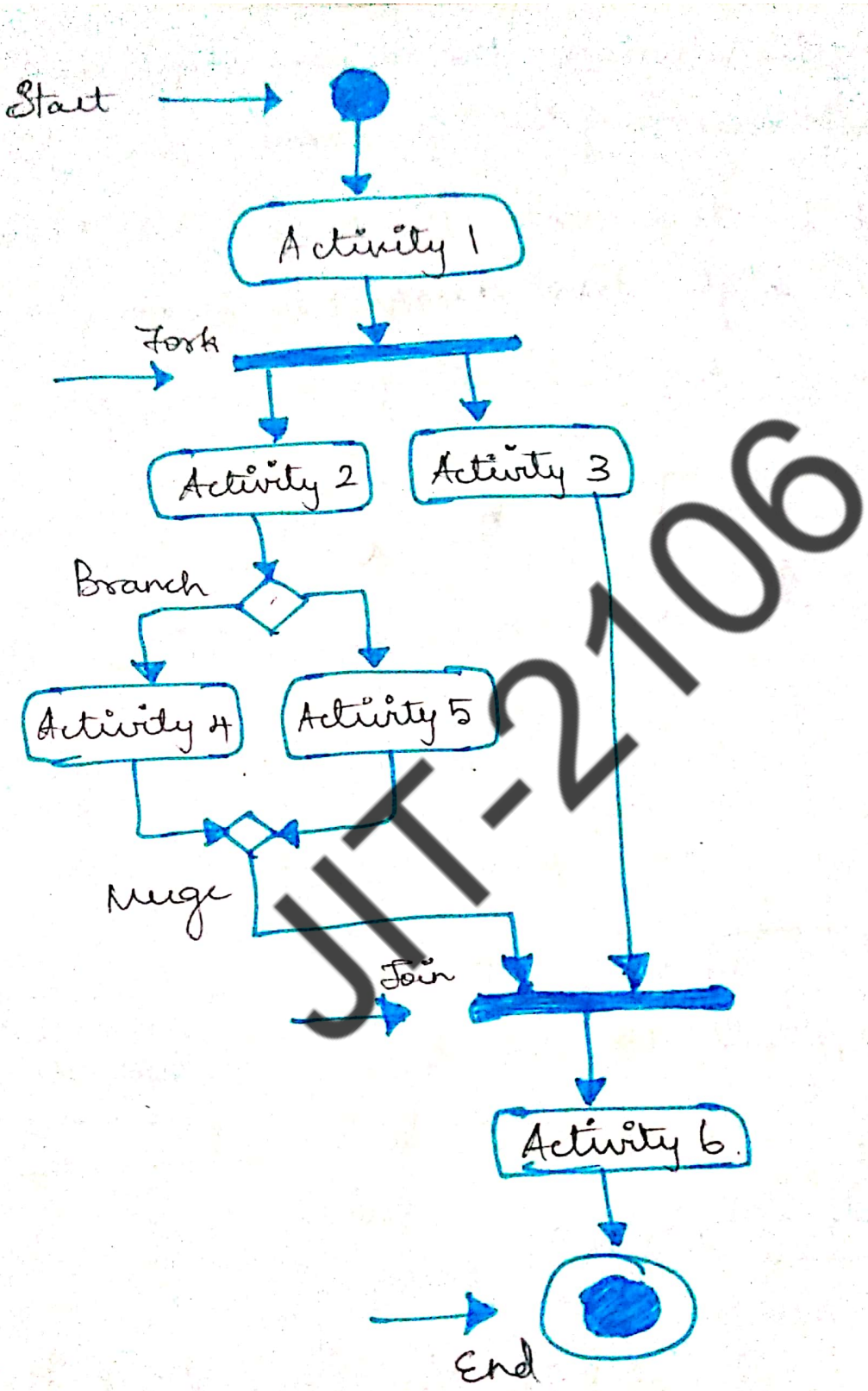
## Notations Used in Activity Diagram

| S.No | Notation | Description | Symbol |
|---|---|---|---|
| 1. | Initial Node | Filled in circle is the starting pt. of the diagram |  |
| 2. | Activity final node | Filled circle with a border is the ending pt. |  |
| 3. | Activity | Roumded rectangles represent activity |  |
| 4. | Flow/edge | Arrows on the diagram |  |
| 5. | Fork | A black bar with one flow going into it & several leaving it. The beginning of parallel activity |  |
| 6. | Join | A black bar with several flows entering into it and one leaving it. |  |
| 7. | Condition | Text on a flow, defining a guard which must evaluate to true in order to traverse the node |  |
| 8. | Decision | A diamond with one flow entering & several leaving |  |
| 9. | Merge | A diamond with several flows entering & one leaving |  |

| | | |
|---|---|---|
| 10. partition | shows different parties involved in the process. Diagram can be organized into several vertical partition called swimlanes indicating who/what is performing activities |  |
| 11. Sub-activity indicates | The symbol in the bottom corner of an activity. It indicates that the activity is described by a more finely detailed activity diagram |   |
| 12. Note | A rectangle with a flip at the top right most corner. |  |

## How to apply Activity Diagrams

* It show the flow of activities through the system
* Diagrams are read from top to bottom & have branches & forks to describe conditions & parallel activities
* A fork is used when multiple activities are occuring at the same time
* Branch describes what activities will take place based on set of cdns
* Branches at some pt. are followed by a merge to indicate the end of the cdn behaviour started by that br.
* After the merge all of the parallel activities must be combined by a join before transitioning into the final activity state
* Activity diagrams are applied to visualize business workflows & processes & Use cases



## Business process Modeling:

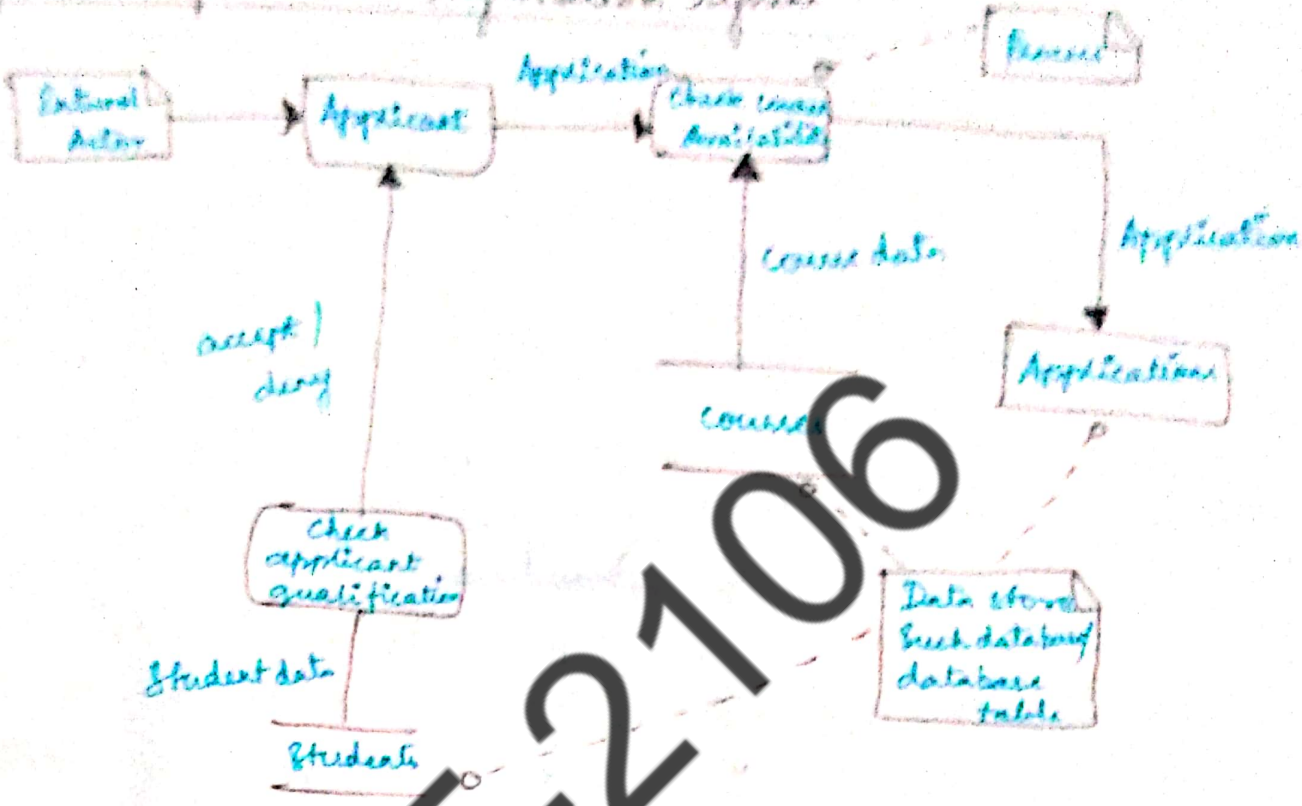
* Bpm is the activity of representing processes of an enterprise so that process may be analyzed and improved in future
* Activity diagram are used to understand the current complex business processes by visualizing it.
* The partitions are useful to see the multiple parties and parallel actions involved in the business

## Data flow modeling

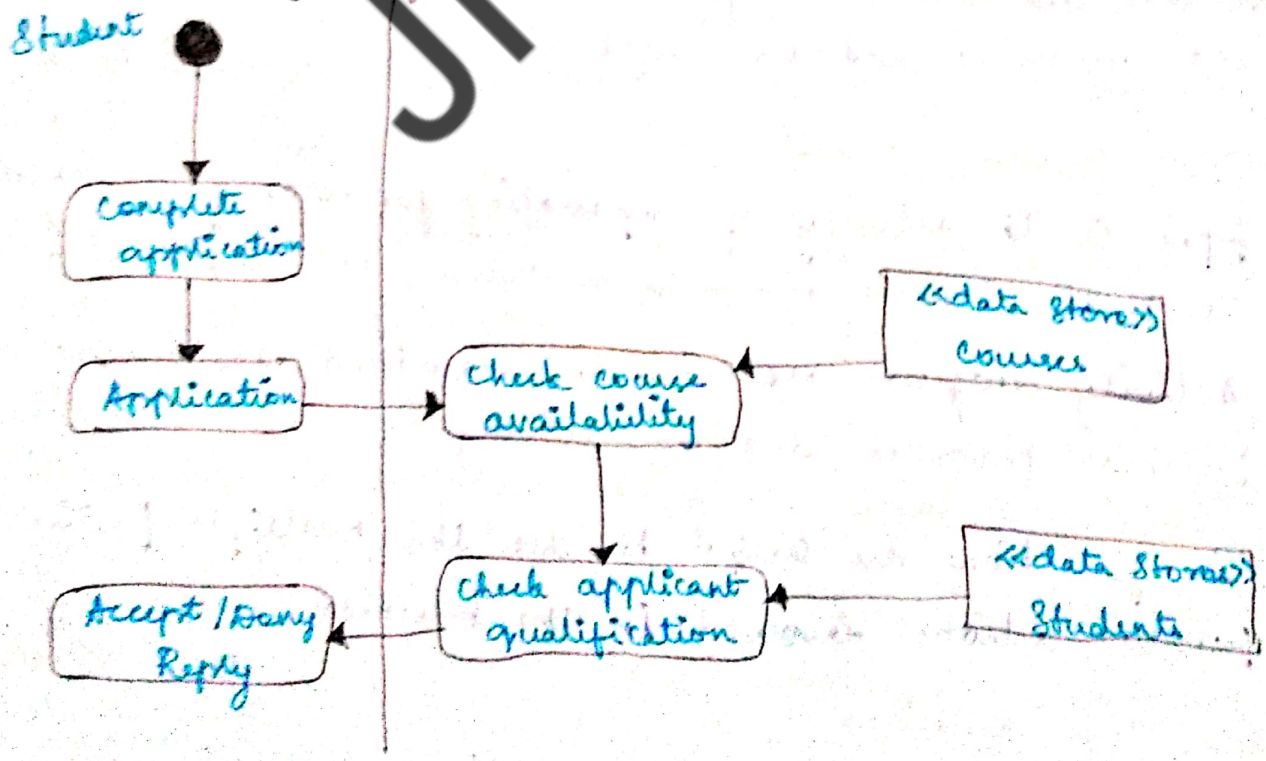
→ A Data flow Modeling (DFD) is a graphical representation of the flow of data through the information system

* DFD is a way to visualize the major steps and data involved in software system process.
* DFD's are used to document the major data flows or to explore a new high level design in terms of data flows

eg: DFD for Course registration system




Applying Activity diagram notation to show Data flow model



## Concurrent programming and parallel algorithm modeling

* Parallel algorithms used in concurrent programming problems involves multiple partitions

Eg: These algorithms are used in 3D simulations like
- ⇒ Oil reservoir modeling
- ⇒ Materials stress analysis
- ⇒ Weather modeling

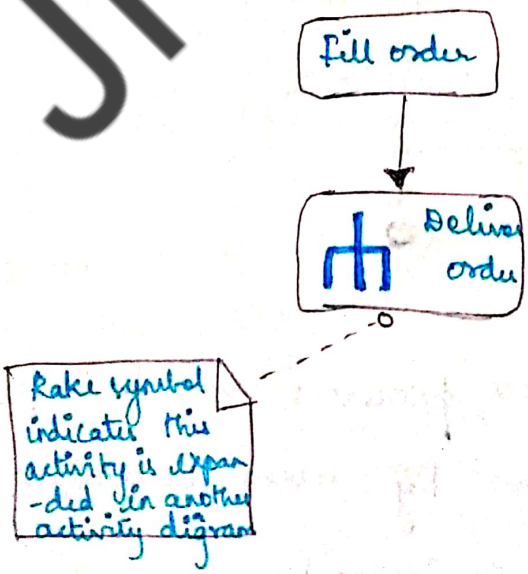
* UML activity Diagram partitions can be used to represent different operating system threads or process

## More UML activity diagram.

### Notations

1. Rake symbol to show an activity is expanded in another activity diagram.


Eg:



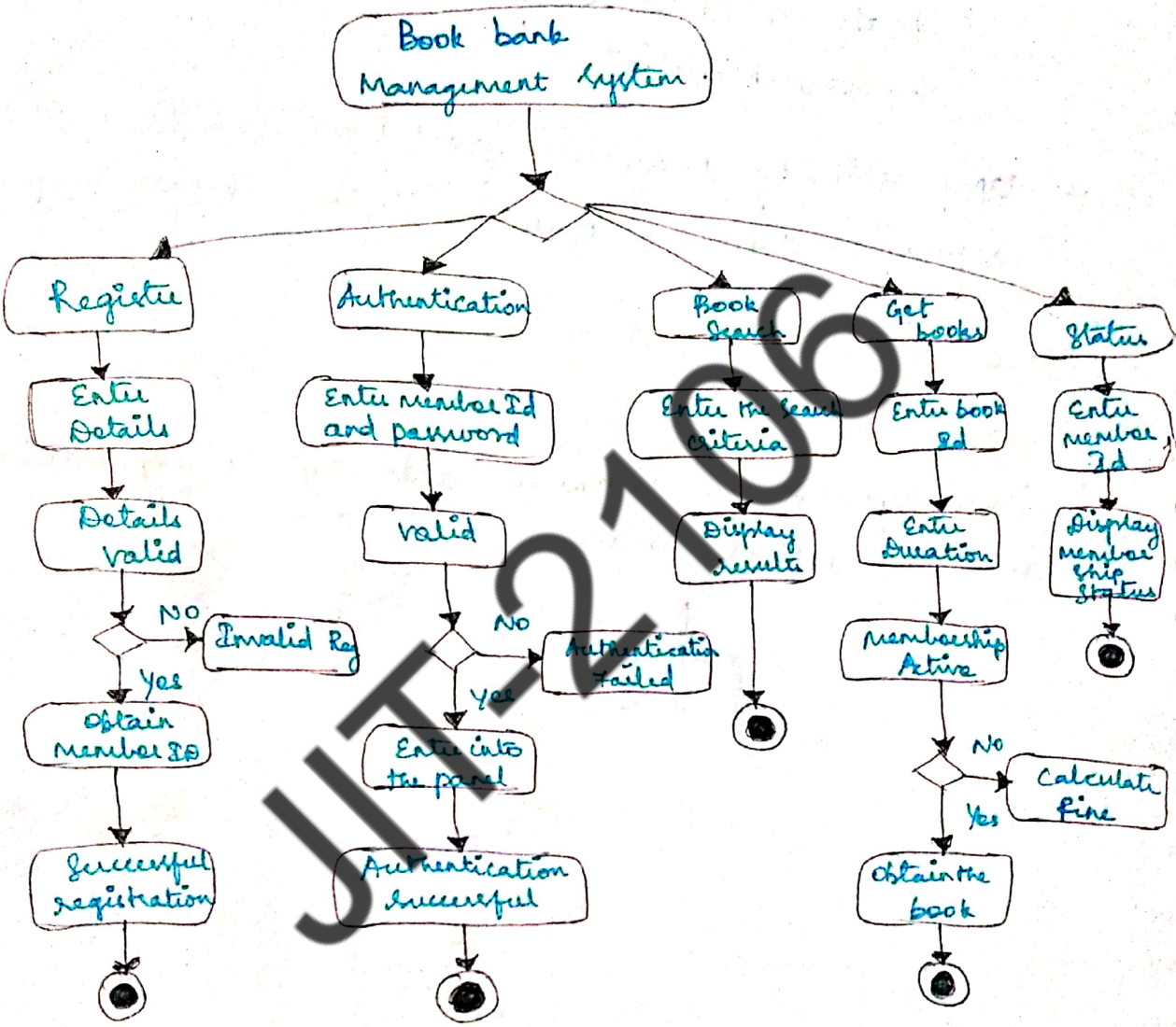

## Guidelines for activity Modeling:

* Use cases are used for simple processes which involves many parties for partition of complex processes

* In order to model a business process, take notation and Sub activity diagram can be used.

* The level of abstraction of action nodes is roughly equal to within a diagram.

Eg: Activity Diagram for Book bank Management System






USES:

1. Visualize business process and workflows.
2. Model work flow by using activities
3. model business requirements
4. High level understanding of system functionalities
5. Investigate business requirements

## Package Diagram

* Package diagrams organize the elements of a system into related groups to minimize dependencies among them.

* UML package diagrams are used to illustrate the logical architecture of a system. The layers, subsystems, packages, etc.

## Package

* package is a namespace used to group together elements that are semantically related and might change together.

* It is general purpose mechanism to organize elements into groups to provide better structure for system model.

## Basic Notation in package diagram:

i) **package:**

To illustrate packages, use a tab folder. Write the name of the package on the tab or inside the folder. Like classes the attributes of a package can be listed.


ii) **visibility:**

* visibility signifies who can access the information contained within the package.

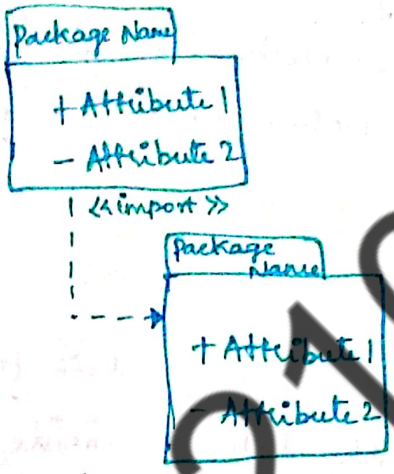
* Private - The attribute or the operation is not accessible to anything outside the package.

* public - Allows the attribute or the operation to be viewed by other packages.

* Protected - Attribute or operation is visible to packages that inherits it.

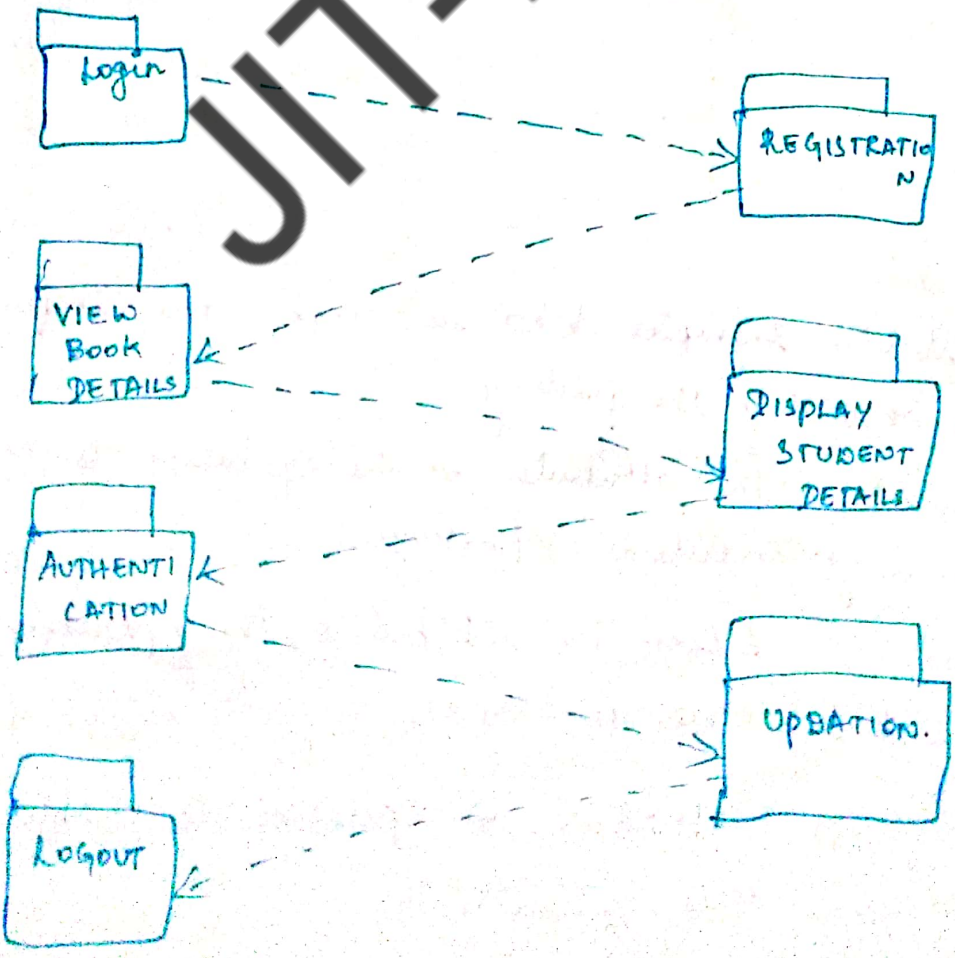

## Dependency:

* It is a relationship in which changes to one package will affect another packages.
* Importing is a type of dependency that grants one package access to the contents of another package.






## Package Diagram for Book bank Management system:



Uses:

* package diagrams can be used to illustrate the functionality of a software system
* This diagram can be used to illustrate the layered architecture of a software system
* The dependencies between the packages can be adorned with labels to indicate the communication mechanism between the layers.

Component Diagram:

* Component diagrams are used to model physical aspects of a system.
* Component diagrams are used to visualize the organisation and relationships and components in a system
* These are also used to make executable systems
* physical aspects are the elements like executables, libraries, files, documents which reside in a Node.

Component:

* A component represents a modular part of system that encapsulates its contents
* It describes the behaviour in terms of provided and required interfaces.
* It is a high level object which implements an Interface. For the services to be supplied by the component in environment like business, machine, container like EJB Enterprise Java beans, web container like servlet and Jsp containers

Purpose:

1. To describe the components used to make functionalities
2. Act as a static implementation view of a system

A single component diagram cannot represent the entire system But a collection of diagrams are used to represent as whole.

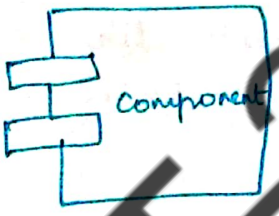
To summarize,

1. Visualize the component of a system
2. Forward and reverse engineering.
3. Describe the organisations and relationship of the components.

Basic Notations

i) Component:

A component is a physical building blocks of the system It is represented as a rectangle as take

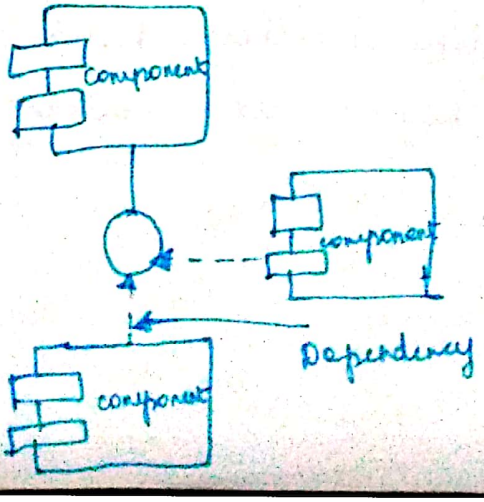



ii) Interface

An Interface describes a group of operations used or created by components

iii) Dependencies

Dependencies are drawn among components using dashed arrows.



## How to draw component Diagram?

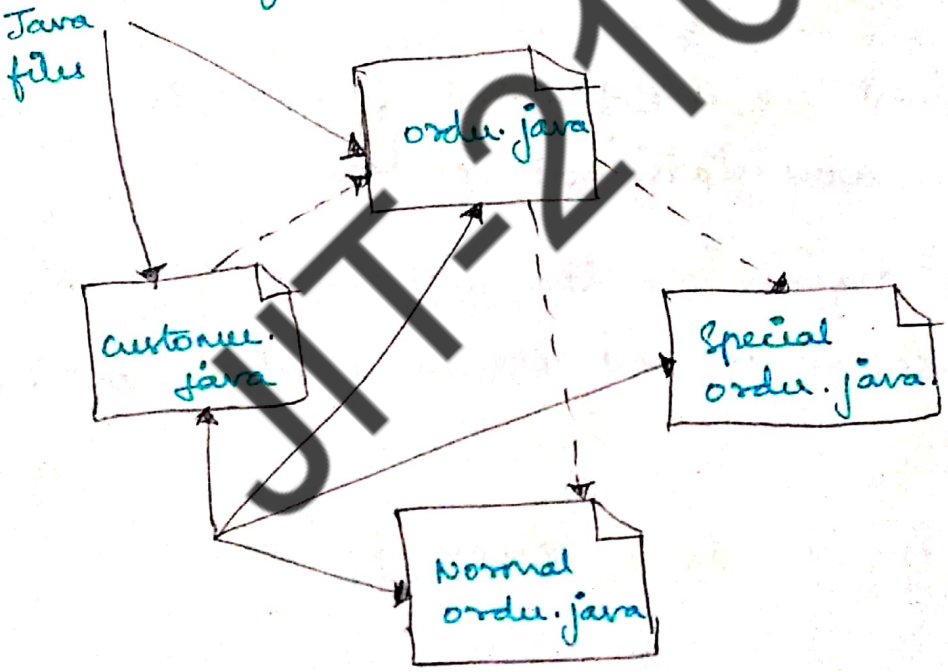
Before drawing a component diagram, the following artifacts are identified.

⇒ Files used in the system
⇒ Libraries and other artifacts relevant to the application
⇒ Relationship among the artifacts.

After Identifying artifacts

⇒ use a meaningful name to identify the components.
⇒ prepare a visualized layout before producing using tools
⇒ Notes to prepare clarifying important points

Component diagram for an order management system.

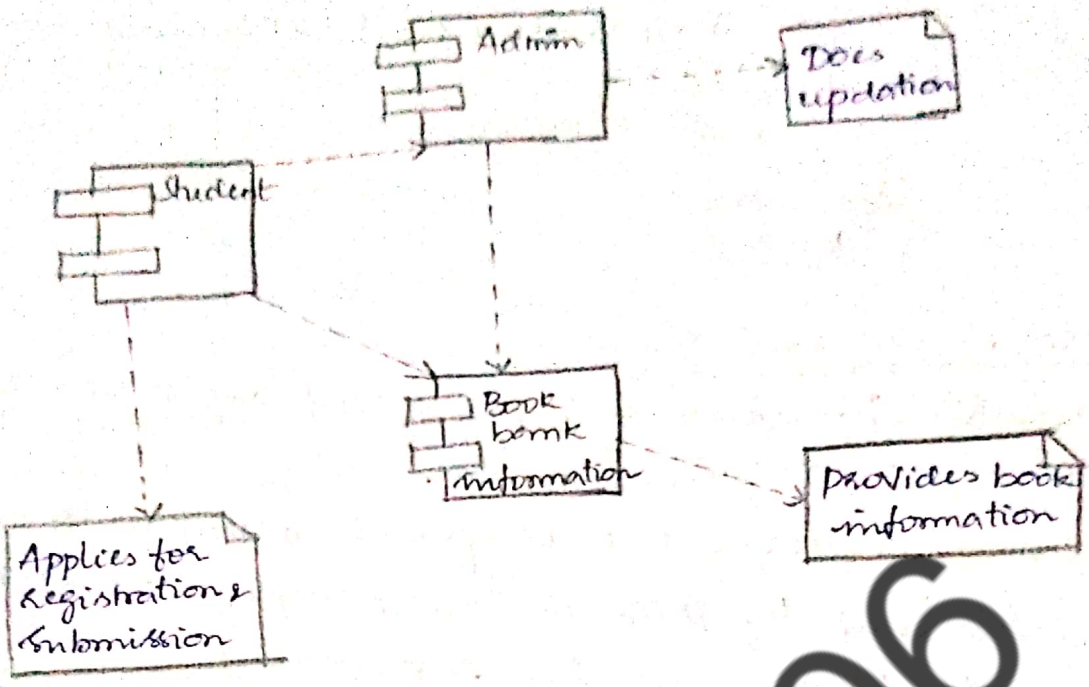





Uses:

1. Model the components of a system.
2. Model database schema.
3. Model executable of an application
4. Model system's source code.

Eg: Component Diagram for a Book Bank Management System




## Deployment Diagram

It is defined as assignment of concrete s/w artifacts (executable files) to computational nodes (processing services).

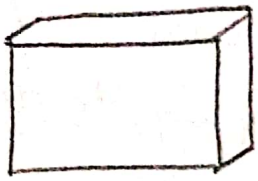
Basic Notations in Deployment diagram

### Node

Nodes appear as boxes, & the artifacts allocated to each node appear as rectangle within the boxes.

A single Node in a deployment diagram may conceptually represent multiple physical nodes, such as a cluster of db servers

Node is shown as perspective, 3-dimensional View of a cube



There are 2 types of Nodes

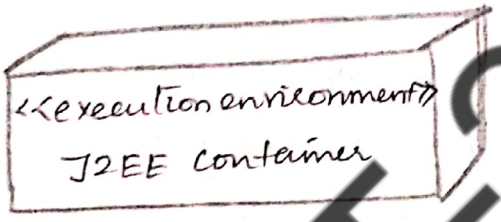
1. Device Node
2. Execution Environment Node

Device Node are physical computing resources with processing memory & services to execute s/w, such as typical computers or mobile phones


Executable Environment Node

* Execution Environment impl. a std. set of services that components require at execution time.

* For each deployment of component, aspects of these services may be determined by properties in a deployment specification for a particular kind of execution environment




* Nodes are interconnected with comm. paths

* Comm. paths can be defined btn nodes such as appln. server & database server to define the possible comm. paths btn the nodes

* Specific n/w topologies can then be defined through links between node instances

Purpose of deployment diagram can be described as:

* Visualize h/w topology of a system
* Describe the h/w components used to deploy s/w components
* Describe runtime processing node

## How to draw Deployment Diagrams

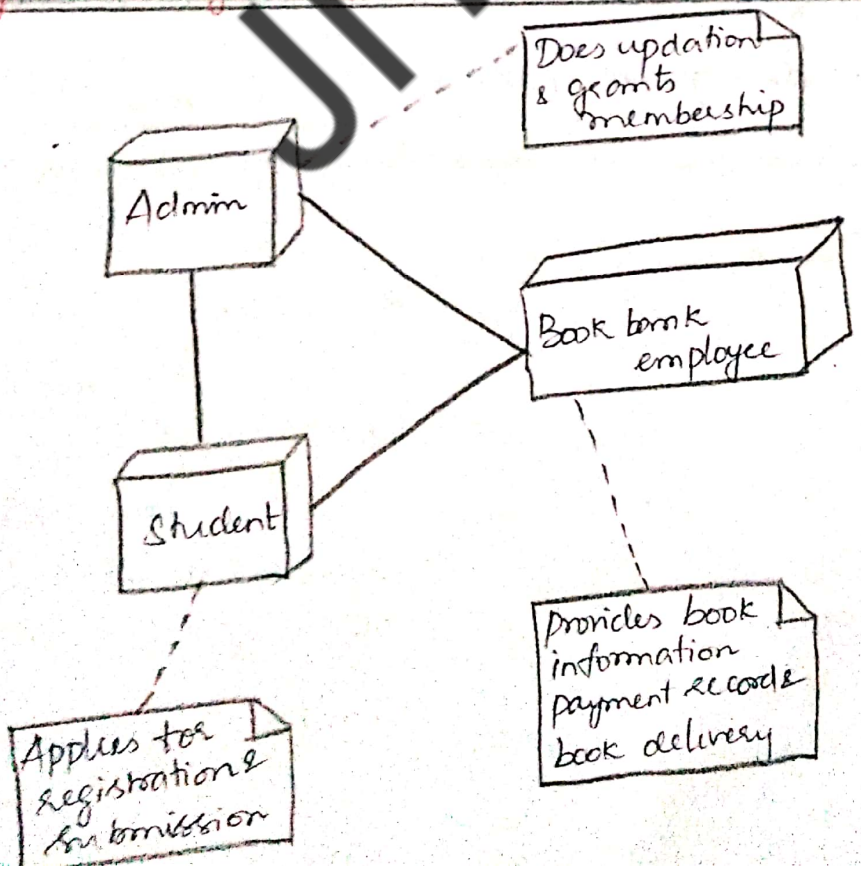
* A deployment diagram consists of nodes. Nodes are nothing but physical h/w used to deploy the applns.
  - Performance
  - Scalability
  - Maintainability
  - Portability

* Before drawing a deployment diagram the following artifacts should be identified
  - Nodes
  - Relationships among Nodes

## Usage of deployment diagrams

- To model the H/w topology of a system
- To model embedded system
- To model h/w details for a c/s system
- To model h/w details of a distributed appln.
- Fwd. & reverse engineering

## Deployment Diagram for a Book Bank Management system

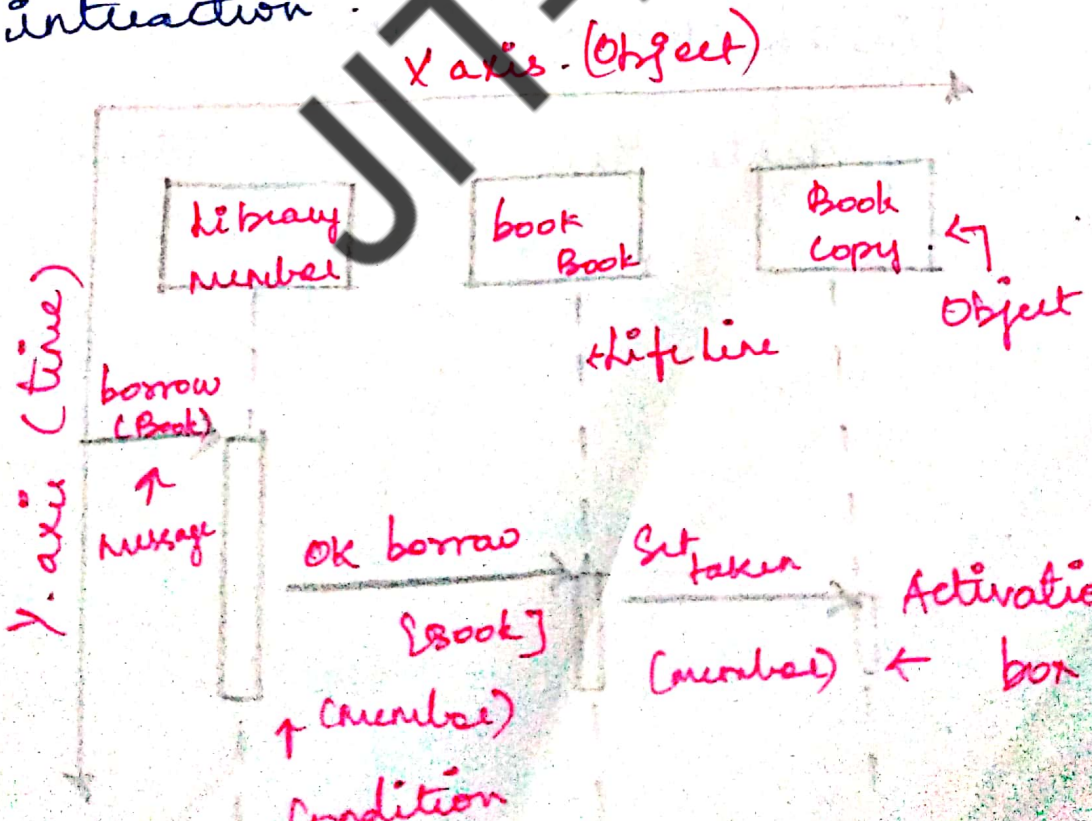


## When to use Component & Deployment Diagrams

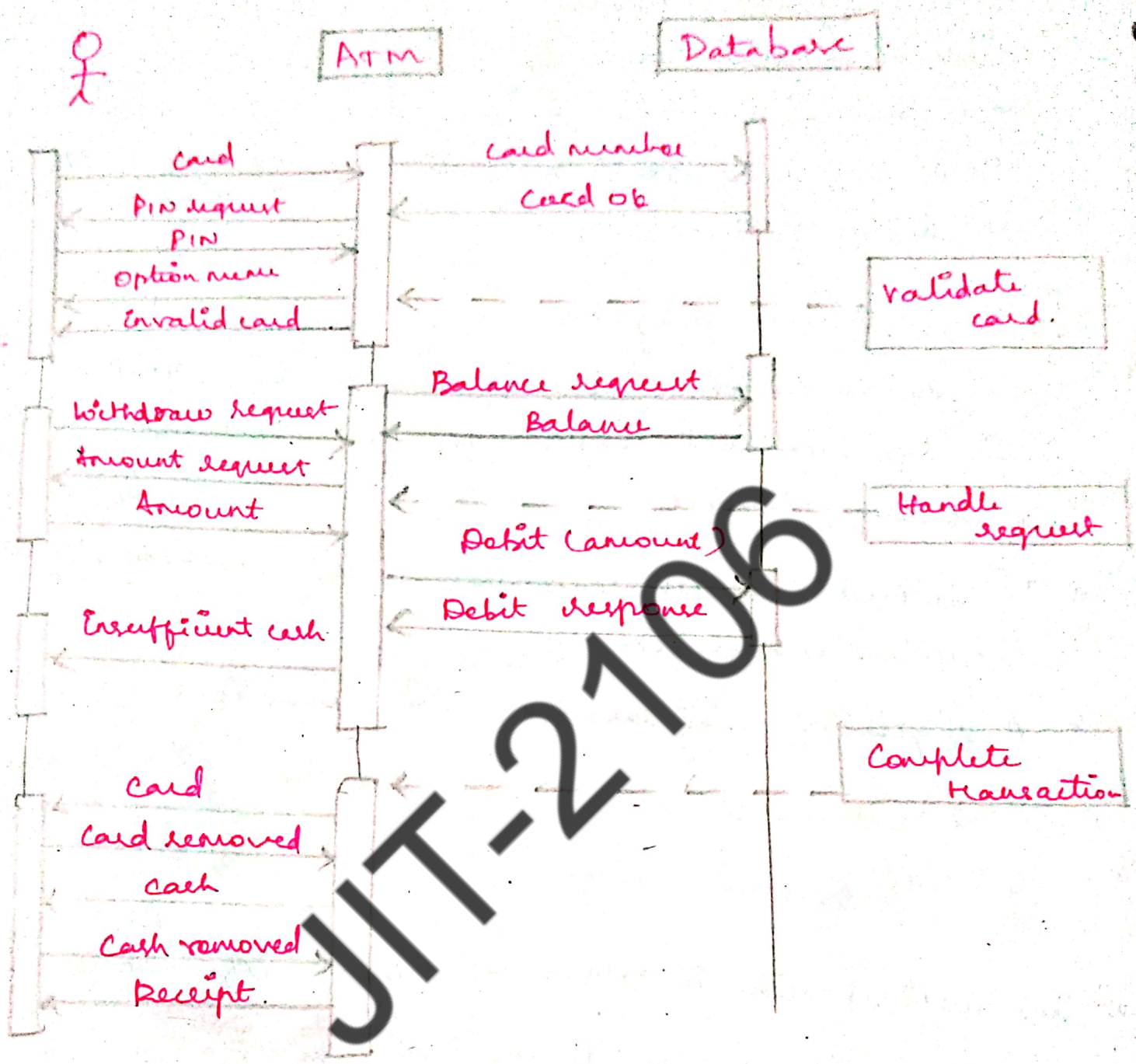
* Component modeling is a specialized type of structural modeling concerned with modeling the implementation of a system.
* Usually apply component modeling during design activities to determine how implementation activities will focus build the system; ie, to determine the elements of the system on which implementation activities will focus.
* Component modeling typically starts after the design of the system is fairly complete, as determined by your system development process
* In contrast to modeling the components of a system, a deployment model shows you the external resources that those components require
* You typically apply deployment modeling during design activities to determine how deployment activities will make the system available to its users ie, to determine the elements of the system on which deployment activities will focus.
* Like component modeling, deployment modeling usually starts after the design of the system is fairly complete, as determined by your system development process

## System Sequence diagram:

* It shows the objects participating in the interaction by their life line and the messages they exchange, arranged in a time sequence.
* It can model simple sequential flow, branching, iteration, recursion and concurrency.
* A sequence diagram has two dimensions
  - ⇒ Vertical dimension → represents time.
  - ⇒ Horizontal dimension → represents different ob
* The vertical line is called as object life line.
* The life line represents the Objects existence duri the interaction.



Sequence diagram - For Atm

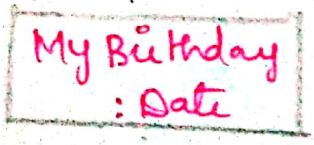





Object and naming:

* syntax : [instance Name][: Class Name]

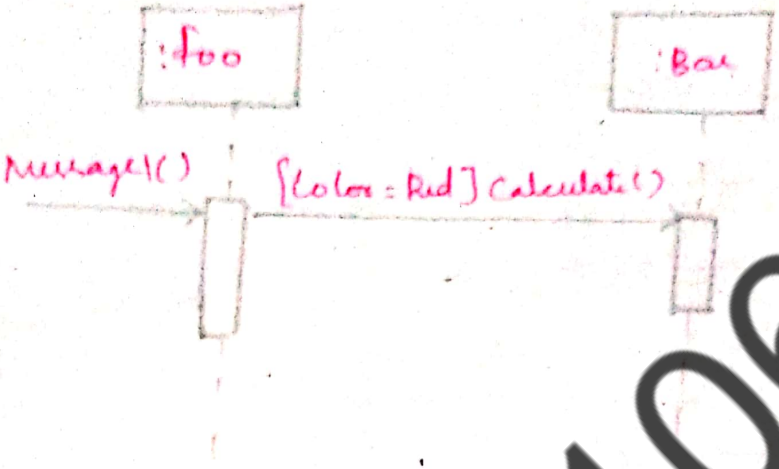
* The life line represents the objects life during the interactions.



Message:

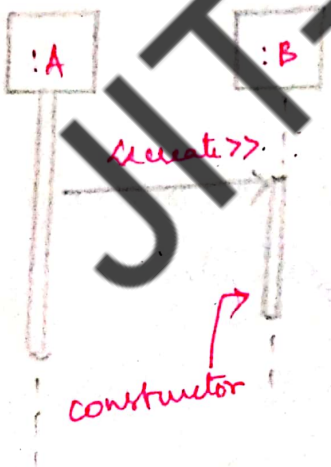
* Each message between objects is represented with a message expression on an arrowed line between two objects. The time organization from top to bottom.

Conditional messages:




Object creation:

An object may create another object via <<create>> messa

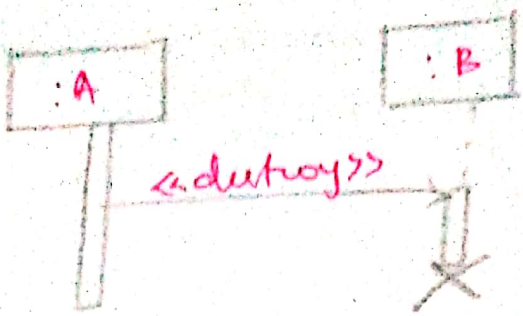



Object destruction:

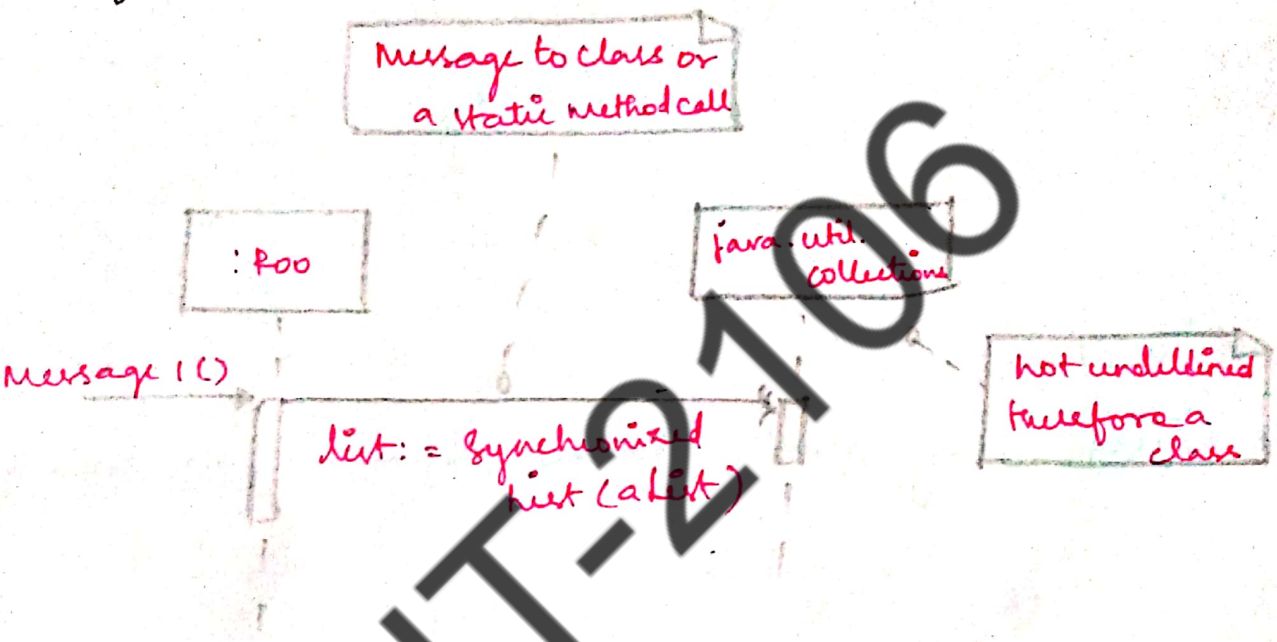
⇒ An object may destroy another object via a <<destroy>> message

⇒ An object may destroy itself.

⇒ Avoid modeling object destruction unless memory management is critical.



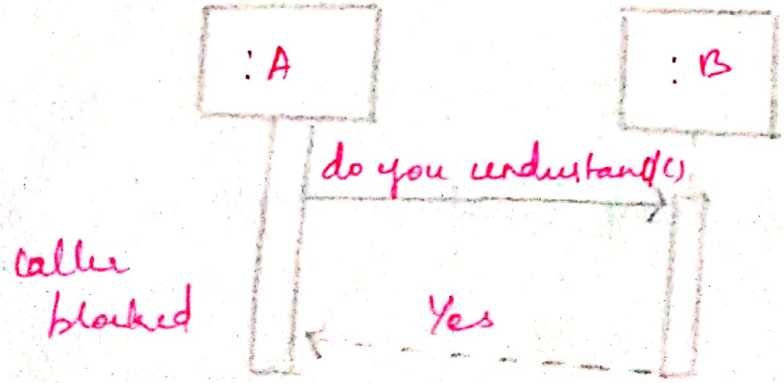

Message to class objects.






Synchronous Message:

The routine that handles the message is completed before the caller resumes execution.



Frame option:

Alt: Alternative multiple fragments, only the one whose conditions is true will execute

Opt: Optional: the fragment executes only if the supplied condition is true

Equivalent to an alt with only one trace.

Par: parallel; each fragment is run in parallel.

Loop: Loop; the fragment may execute multiple times and the guard indicates the basis of iteration

Region: critical region; the fragment can have only one thread executing it at once.

Relationship between sequence diagram and use case

System sequence diagram (SSD)

* A System sequence diagram is a picture that shows a particular session scenario of a use case, the events that external actors generate their order and inter system events.
* All systems are treated as a black box, the emphasis of the diagram is events that cross the system boundary from actors to systems.
* An SSD should be done for the main success scenario of the use case and frequent or complex alternative scenario.

Examples:

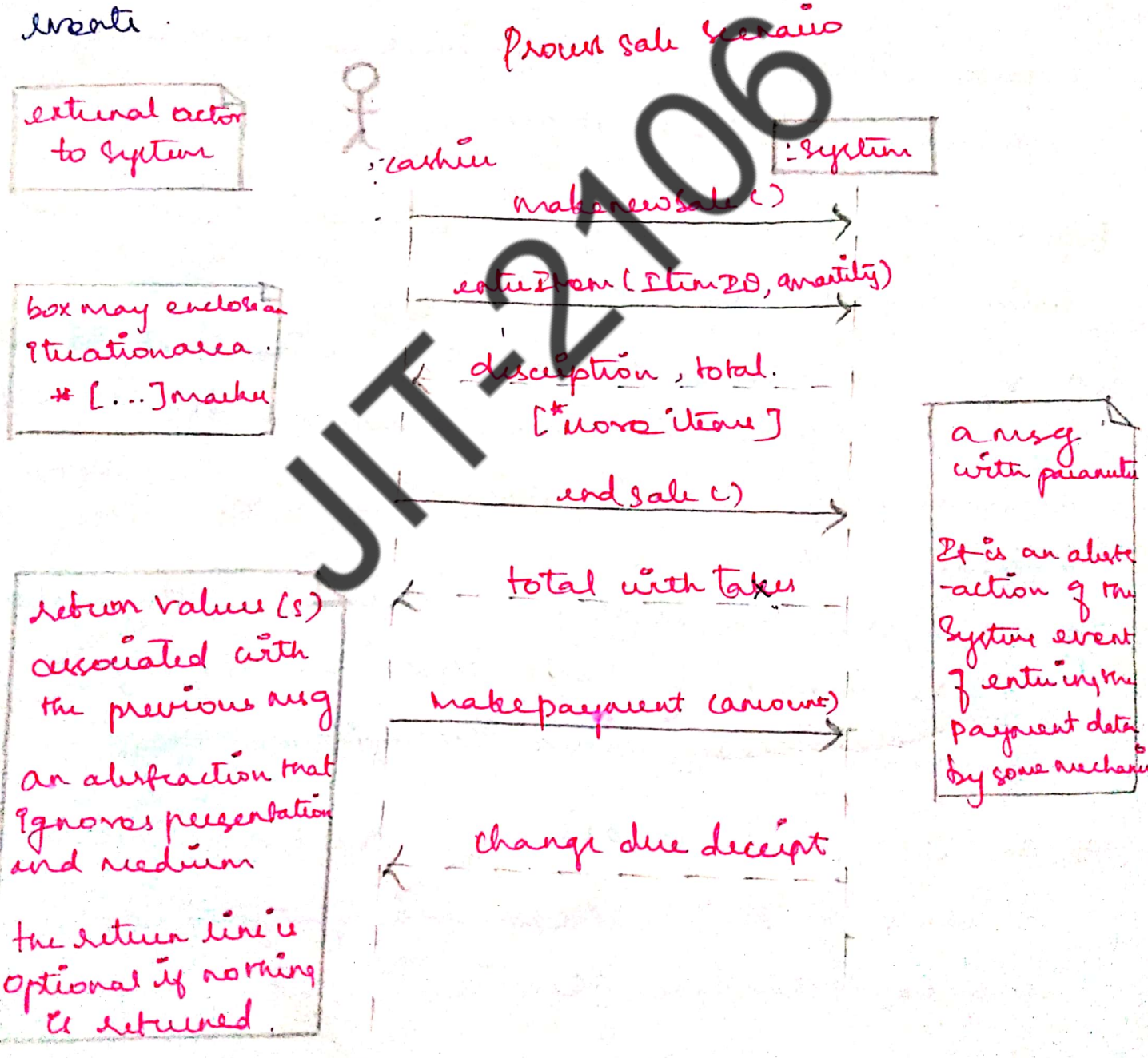
* An SSD show, a particular course of events within a use case, the external actors that interact directly

with the system (as a black box) and the system events that the actor generate.

⇒ Time proceeds downward and the ordering of events should follow their order in the use case. System events may include parameters.

eg: process Sale use case.

⇒ It indicates that the cashier generates MakeNewSale, enterItem, endSale and makePayment system events.



Extra System SSD:

SSD's can also be used to illustrate collaborations between systems such as between the Next Pos Gen and the external credit payment authorizer.

SSDs and use cases:

| Simple cash-only Process Sale scenario |
| --- |
| 1. customer arrives at a pos checkout with goods or services to purchase.<br>2. cashier starts a new sale<br>3. cashier enters item identifier.<br>4 system records sale line item and presents item description, price and running total.<br>5 system presents total with taxes calculated<br>6. cashier tells customer the total and asks for payment<br>7. customer pays and system handles payment. |

SSD derived from use cases.

SSD with use case text

1. customer arrives at a pos checkout with goods and/or services to purchase.
2. cashier starts a new sale.
3. cashier enters item identifier.
4. System records sale line item and presents item description, price and running total.
5. System presents total with taxes calculated.
6. cashier tells customer the total and asks for payment
7. customer pays and system handles payment.

# Logical architecture and package diagram.

Logical architecture : The large scale organisation of software classes into packages, subsystems and layers.

* Logical because no decisions about deployment are implied.

Layer : A very coarse grained grouping of classes, packages or subsystems that has cohesive responsibility for a major aspect of the system.

Typical layers in an OO system.

- ⇒ user interface.
- ⇒ Application logic and domain objects.
- ⇒ Technical services.

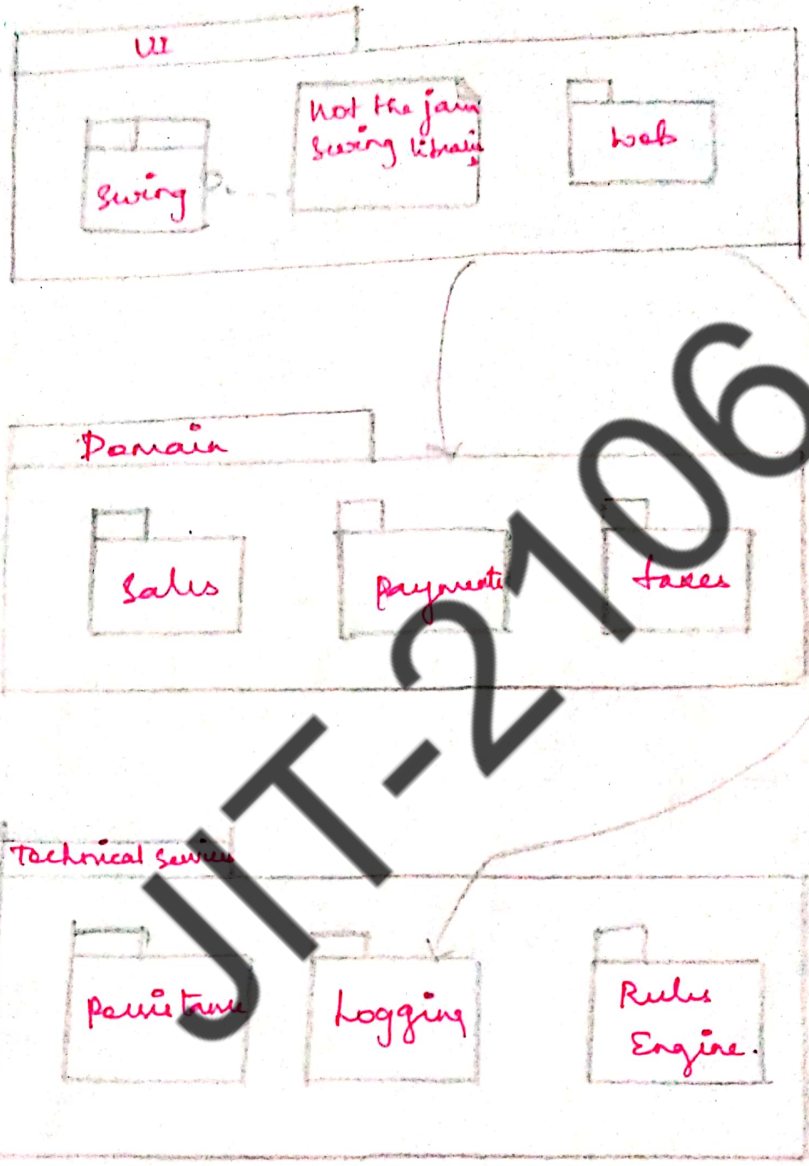
* Application independent, sharable across systems

Relationship between layers.

Strict layered architecture: A layer only calls upon services of the layer directly below it.

Relaxed layered architecture: A higher layer calls upon several lower layers.

Layers shown with UML package diagram.






How to design layers:

⇒ organize the large scale logical structure of a system into discrete layers of distinct related responsibilities.

⇒ cohesive separation of concerns.

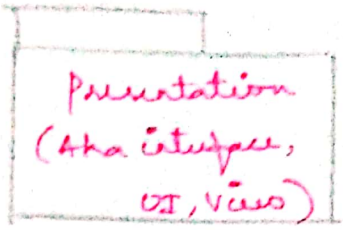
⇒ Lower layers are general services.

⇒ Higher layers are more application specific.

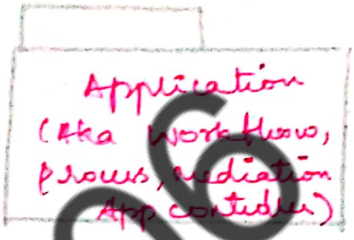
→ collaboration and coupling from higher to lower layers

→ lower to higher layer coupling is avoided

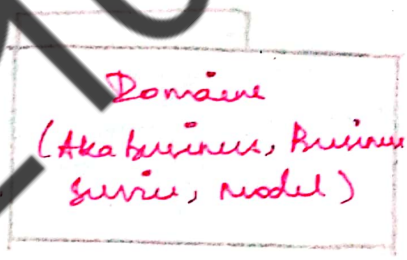
Common layers in Is Architecture



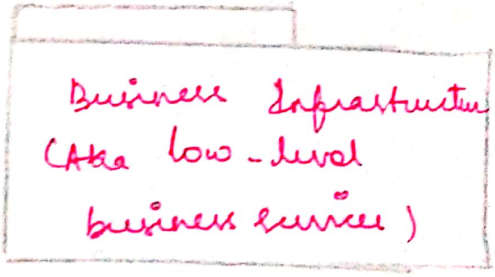

* Gui windows
* Reports
* Speech interface
* Html, XML, XSLT, JSP, Java script



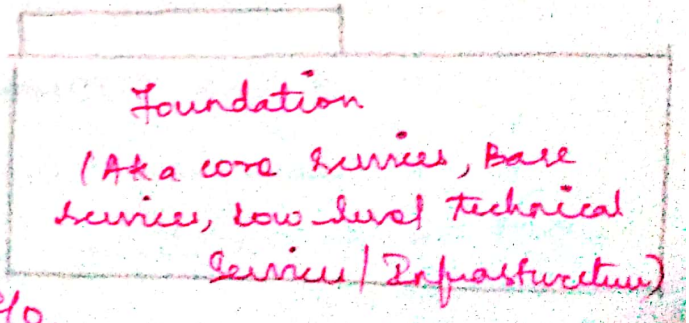

* handles presentation layer requests
* workflow
* session state
* window / page transitions
* consolidation / transformation of disparate data




* handle application layer requests
* implementation of domain rules
* domain services pos, Inventory.




* very general low-level business service used in many business domains.
* currency convertor




* low level technical services, utilities and frameworks
* data structures threads
* math, file, DB and network I/O.

Benefits:

* There is a separation of concerns, a separation of high from low-level services and of application specific from general services.

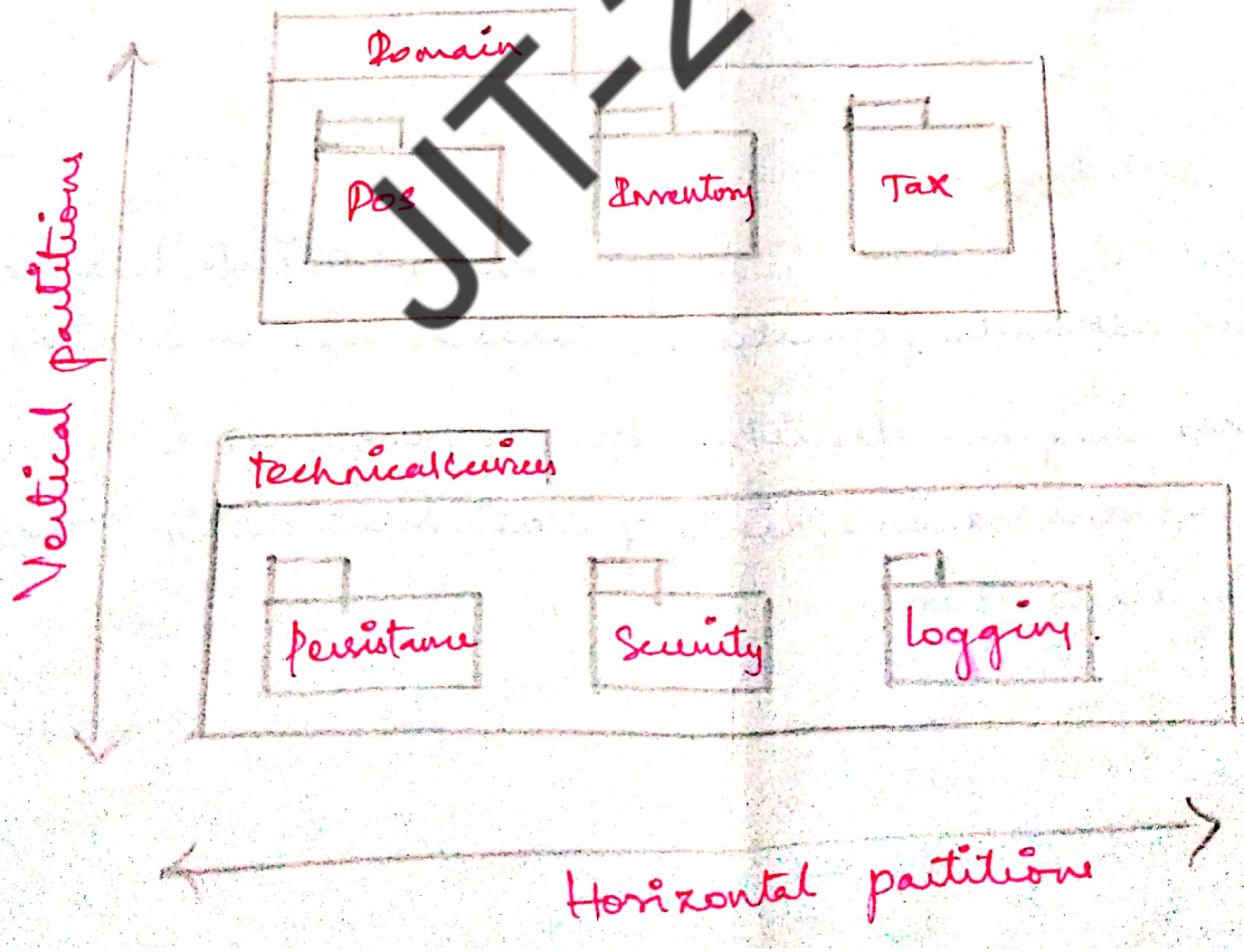
* This reduces coupling and dependencies, improve cohesion increase reuse potential, increase clarity

* Some layers can be replaced with new implementation This is generally not possible for low level technical services or foundation layers but may be possible for UI, application and domain layers.

* lower layers contain reusable functions.

* Some layers can be distributed.





## Model view Separation principle.

→ model → The domain layer of objects

view → user interface (UI) objects

Model objects should not have direct knowledge of view objects.

→ Do not connect or couple model objects directly to UI objects.

eg:- don't let a Sale Object have a reference to a Java swing JFrame window object.

→ Do not put application logic in a UI Object.

→ UI objects should receive UI events and delegate requests for application logic to non-UI objects.