UML → unified modelling language

UML is used for → documenting, visualizing, specifying, construction

Grady Booch, Ivar Jacobson, Jim Rumbaugh → each of these person had their own methods, rules & notations to draw the diagrams to visualize an aspect.

There was a confusion on which methods to actually follow →
- grady booch : booch methods
- Ivar Jacobson : OOSE (object oriented software engineering)
- Jim rumbaugh : OMT (Object modeling technique)

* Grady booch was good at design
* Ivar and rumbaugh were good at analysis

→ Three people combinely called unified → as combined created a tool/language called UML (unified modelling language)

called modelling → to model the system

called language → rules & grammar

model :- a better understanding prototype or a blueprint from which we can implement the original working product

model is simplification of reality

UML → is a language for visualizing, specifying, constructing and documenting software artifacts.

Aims of modelling → There are 4 Aims of modelling

1. models help us to visualize a system as it is (or) as we want it to be
2. models permit us to specify the structure or behaviour of a system
3. models give us a template that guides us in constructing a system.
4. models document the desisions we have made

Requirements → model → Implementation

* system consists of static & dynamic behaviours and we can specify static & dynamic behaviours using diagrams during modelling
* we can visualize the progress of the model that is being developed to the customers
* Once the model is ready, we can apply the forward engineering concept to generate the skeleton code and vice versa is the reverse engineering - we can use skeleton code during implementation
* we can prepare individual documents for analysing, modelling, constructing and deploying

Advantage in UML → we can parallaly draw the diagram and also write the documentation.

- UML 0.8 was the first Standardised version released in 1995.

Principles of modeling → There are 4 principles

1. The choice of what models to create as a profound influence on how a Problem is attacked and how a solution is shaped
2. Every model may be expressed at different levels of precision
3. The best models are connected to reality
4. NO single model is sufficient.

* Assume a small project, with less modules and a less duration. Based on how much the person either technical or non-technical can understand we draw the diagrams
* even though you draw very less diagrams or all the diagrams, the diagrams drawn must cover all the requirements.
* Awording to project complexity some diagrams even may explain most of the details of software
* All the requirements cannot be represented in a particular diagram

Conceptual model of UML

Building blocks of uml :-

- Things → structural things, behavioural things, grouping things, Annotational things
- Relationships → Association relationship, dependency relationship, Generalization relationship, Realization relationship
- Diagrams → Structural diagrams, Behavioural diagrams

* To understand the uml you need to form a conceptual model of the language and this requires learning three major elements and these things are called building blocks of uml.

- Things:
  - Structural things: («Actor), class, interface, use case, Collaboration, Component, node, Active class — NAACCCUI
  - Behavioural things: state machine, Interaction — SI
  - grouping things → Package — Package, Tabbed folder
  - Annotational things → note — notation name :- note; graphical name :- rectangle with dogeared corner; Purpose :- comments
- Relationships
- diagrams

Structural diagrams: Class diagram, object diagram, Component diagram, deployement diagram — CCOD

behavioural diagrams: usecase diagram, sequence diagram, Activity diagram, state chart diagram, Collaboration diagram — SSACU

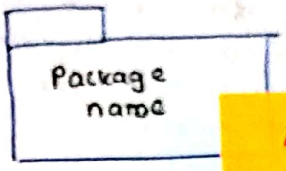
- Relationships notations

Association relationships : ——→

dependency relationships : - - - - →

group thing :-

* Package :- A package is a general purpose mechanism for organizing elements into groups. Structural things, behavioural things and other grouping things maybe placed in a package

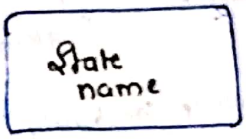


graphical name :- tabbed folder

A package is a name. may contain model el

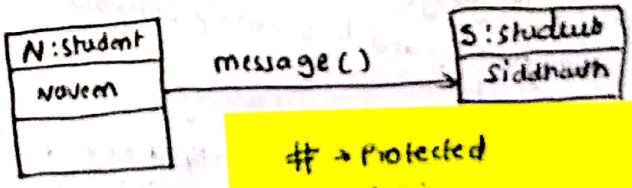
Behavioural things

* State machine :- A state machine is a behaviour that specifies the sequence of states and objects (or) an interaction goes through during its lifetime in response to events, together with its responses to other events.



graphical name :- rounded rectangle

* Interaction : An interaction is a behaviour that compresses a set of messages exchanged among a set of objects within a particular context to a specific purpose.



→ Protected
+ → Public
- → Private

Structural things

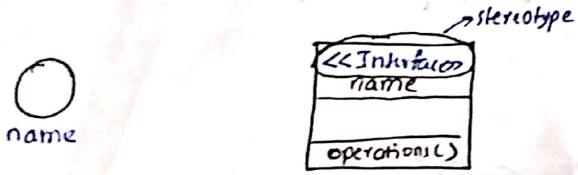
* class :- A class is a description of set of objects that share the same attributes, operations, relationships and semantics. A class implements one or more interfaces. A class is graphically a rectangle usually including its name, attributes and operations.

common modelling techniques for a class
1. modelling vocabulary of a system
2. modelling the distributions responsibilities in a system
3. modelling non software things
4. modelling primitive types

class name
attributes
operations

(or) bird

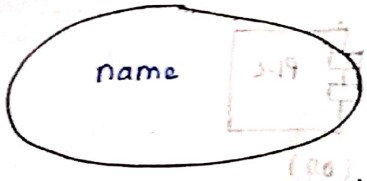
* Interface :- An Interface is a collection of operations that specify the service of a class or component. An interface describes the externally visible behaviour of that element graphically an interface is circle with name.



Icon form expanded form

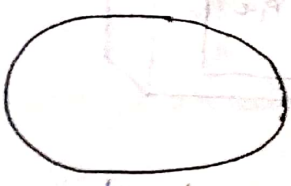
- Interface is connected with a component or a class
- name of the interface.

* use-case :- A usecase is a description of set of sequence of actions that a system performs that yeilds an observable result of value to a particular actor, A use case is used to structure the behavioral things in a model. graphically it is an ellipse with a solid line including its name.

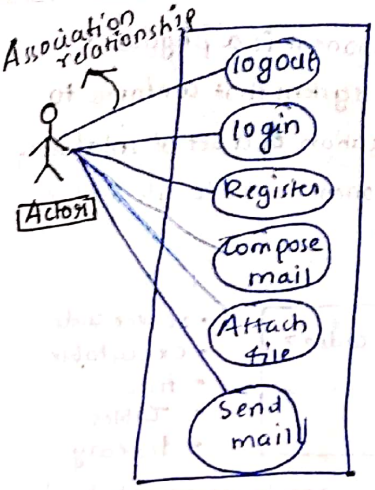


graphical name :- ellipse

(or)



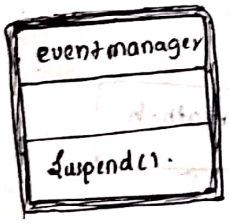
name.



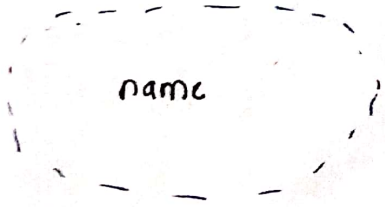
use case diagram

* If the actor has the role to access the features and perform the actions then that is going to produce the observable result.

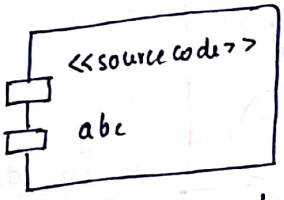
* Active class :- Active class is a class whose objects own one or more processes (or) threads and therefore can initiate control activity. Active class is just like a class but with heavy lines



* Collaboration : Collaboration defines an interaction and is a society of roles and other elements that work together to provide some co-operative behaviour. A collaboration is an ellipse with dashed line



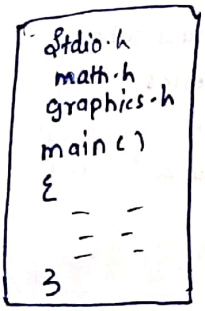
* Component : A component is a physical & replacable part of a system that confoims to and provides realization of a set of interfaces. Graphically a component is Rectangle with 2 tabs

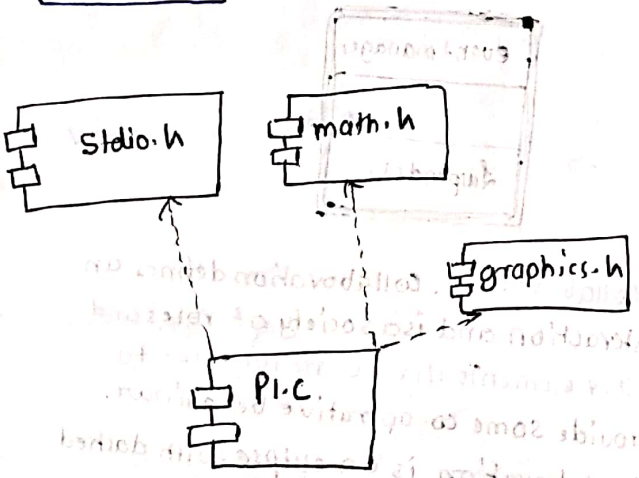


- source code
- executable
- files
- Tables
- library

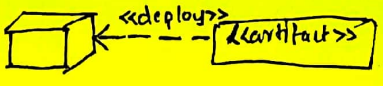
can be represented using component.

ex:-

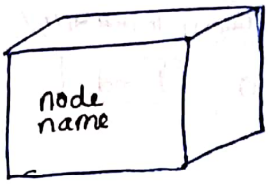




Artifact : An artifact represents a Physical peice of information that is used by a s/w or produced during s/w development

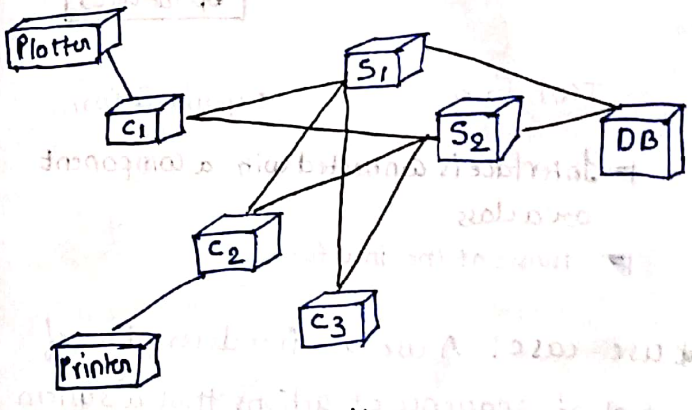


* node :-



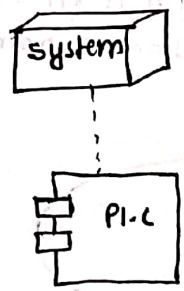
graphical name : cube

A node is a physical element that exists at runtime and represents a computational resource, generally having atleast some memory and often processing capability. A set of components may reside on a node and may also migrate from node to node

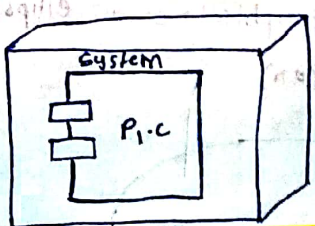


Deployement diagram

ex:- execution of a component by node



(or)



Communication path : is a path or an association between two nodes to exchange information

* Actor :-  : An actor represents a coherent set of roles that users play in use cases
graphical name :- Stick figure

Relationships → is a connection among things

* Association relationship :- Association is a structural relationship that describes a set of links. A link being a connection among objects.
Aggregation is a special kind of association representing a structural relationship between a whole and its parts relationship.

(line with no arrow is bidirectional)

▶ the above diagram means class A objects are linked with class B objects and vice versa

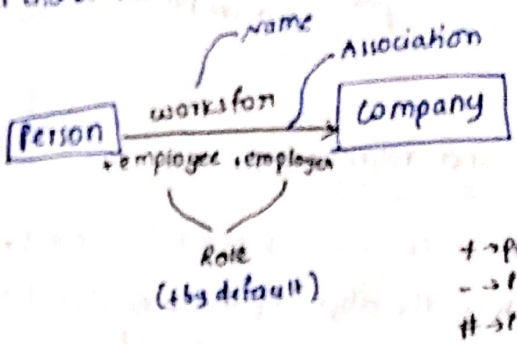
◤ The above diagram means class A objects are linked with class B objects but not vice versa in the above diagram.

Adornments :- There are 4 adornments which we can apply to Association relationship
→ name
→ role
→ multiplicity
→ Aggregation

Name Adornment :- An Association can have a name and you use that name to describe the nature of relationship. So that there is no ambiguity about its meaning.

eg: 

Role Adornment :- when a class participates in an association it has a specific role that it plays in that relationship. A role is just the phase that class at near end of the association presents to the class at the other end of the association.

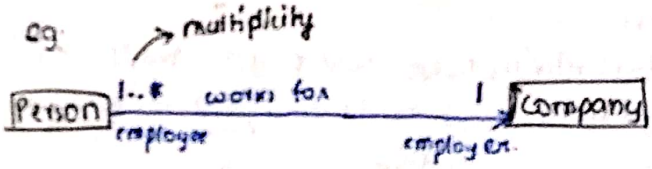
eg: 

+ → public
- → private
→ shielded

multiplicity :- To state how many objects may be connected across an instance of an association This How many is called the multiplicity of the association role.

Notation	meaning
1	one
0..1	0 or 1
1..*	1 or more
0..*	0 or more
2	exactly 2
0..4	0 or 4

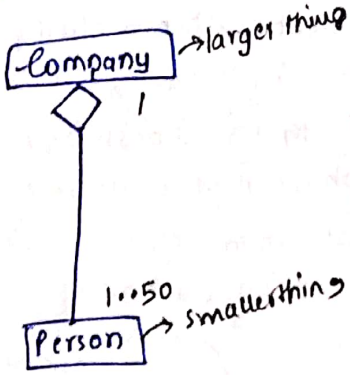
* → can be replaced by any exact number.

eg: 

Aggregation :- If you want to model a "whole/parts" relationship in which one class represents larger Thing which consists of smaller things (parts). This kind of relationship is called Aggregation which represents aggregation

graphical name :- solid line with diamond shape

eg:



Generalization relationship:

generalization is a specialization class. in which the objects of the specialized element (child) are substitutable for objects of the generalized element (parent).

child - shares the structure & behaviour of parent

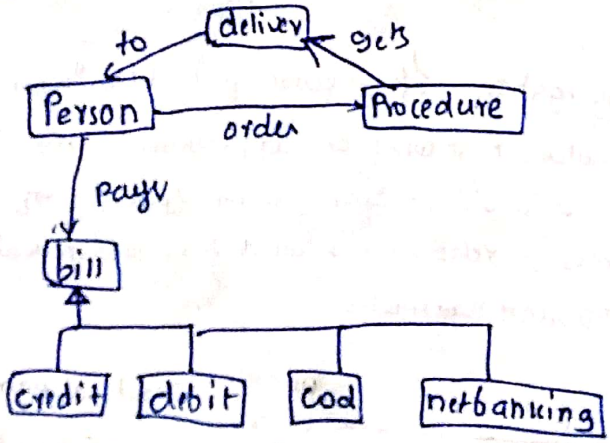
graphical representation

Solid arrow with Hallow head

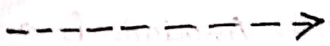
→ also called as Is-a kind of relationship

Scenario

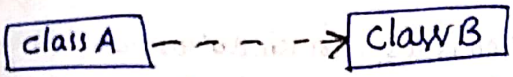
* draw a class diagram
* A person orders products, order contains no of items.
* After ordering person will pay the bill through gateway using credit, debit, cod, net banking



Dependency relationship → is a symantic relationship between two things in which a change to one thing (The independent thing) may effect the symantics of the other thing (The dependent thing)

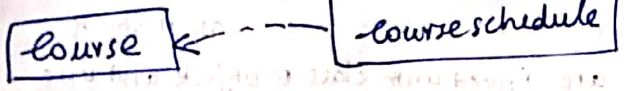


graphical name: dashed line with Arrow

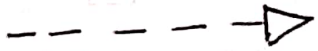


(dependent) (Independent)

* class A depends on class B



Realization relationship



graphical name: dashed line with arrowheaded (hallow)

* Realization is semantic relationship between classifiers, where one classifier specifies a contract that another classifier gaurentees to carry out.

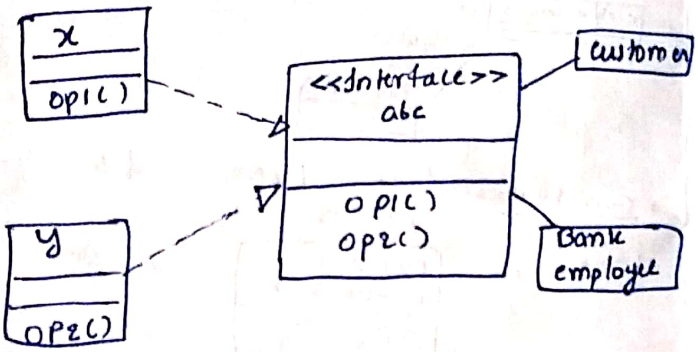
> Consider the following skeleton code

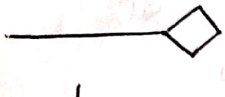
```
interface abc
{ op1();
  op2();
}
class x implements abc
{ op1()
  {
  }
}
```

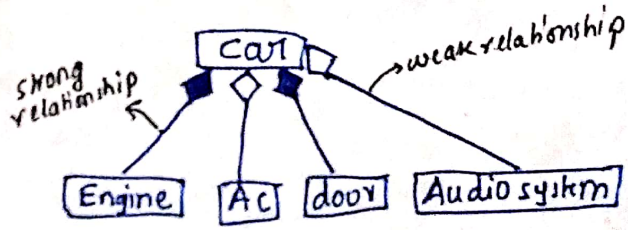
```
Class y implements abc
{
  opz()
  {

  }
}
```

∴



Aggregation	Composition
	
↓ If weak relationship Aggregation is used	↓ If strong relationship composition is used.
An Aggregation is an association that represents a whole-part relationship	A composition is a form of aggregation with a stronger ownership and coincident lifetime of part with the whole.



Diagrams (Structural)

- class diagrams → class interface
  - → Relationships
  - → package etc
- object diagram → object
  - → link
  - → package
  - → note.
- component diagram → node, component,
  - → package
  - → dependency
- Deployement diagram → node
  - → link (or) connection
  - → package
  - → note

Diagrams (Behavioral)

- use case diagram → note <<extend>>
  - → use case
  - → Actor
  - → Association
  - → generalization
  - package
  - <<Include>>

* Sequence diagrams and collaboration diagrams are interaction diagrams (Isomorphic). One diagram of two can be drawn and can be converted into another without any loss of data.

- Sequence diagram → ① object, message, package, note, object lifeline, focus of control
- collaboration diagram → ② package, object, message, note, sequence numbering

- Activity diagram → Join
  - → fork
  - → note
  - → Transition
  - → end state
  - Activity
  - Starting state
  - swimlane
  - package

State chart diagram → state machine
→ Transition
→ starting state
→ ending state
Histy state
note
Package

Rules in the UML → UML have some rules to define how a well formed model should look like!

But to mainly to draw well formed models ← so models can be formed, elided, incomplete, inconsistent

Semantic rules for UML :- NSVIE

Names	what you can call things, relationships, diagrams
Scope	The context that gives specific meaning to a name
Visibility	How those names can be seen & used by others
Integrity	How things properly & consistently relate to one another
Execution	What it means to run or simulate a dynamic model

Common mechanisms in the UML :-

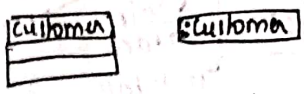
Presence of these mechanisms apply consistency throughout the language

① **Specifications:** The UML's specifications provide a semantic backplane that contains all the parts of all the models of a system. Each part related to one another in a consistent fashion. The UML diagrams are thus simply visual projections into the backplane, each diagram revealing a specific interesting aspect of the system.

② **Adornments:** most elements in UML have an unique and direct graphical notation that provides a visual representation of most important aspects of the elements.

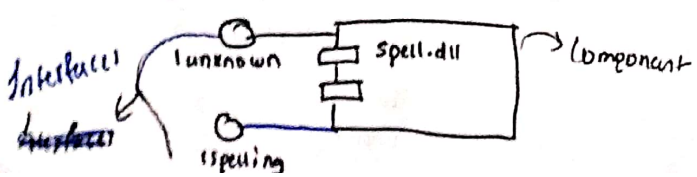
③ **Common divisions:-** In modeling object oriented systems, the world is often gets divided in a couple of ways

(i) There is a division of class and object. A class is an abstraction; An object is one concrete implementation / manifestation of that abstraction
eg: In UML, we model classes & objects



(ii) There is a seperation of interface & implementation. An interface declares a contract & an implementation represents one concrete realization of that contract.

eg:-

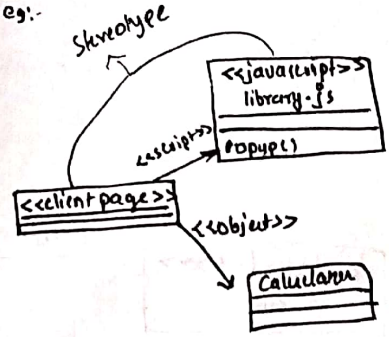


④ **extensibility mechanisms :-**

- Stereotypes
- Tagged values
- constraints

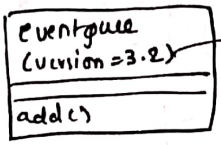
- **Stereotypes :-** It extends the vocabulary of the UML, through which new building blocks can be created out of existing ones

eg:-



- **Tagged values :** It extends the properties of UML building blocks

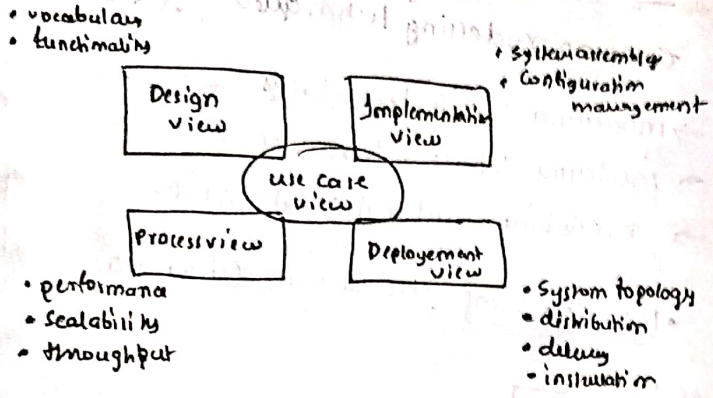
eg:-



- **Constraint :-** A constraint extends the semantics of UML building blocks (used to add new rules)

eg:-

Architechture / 4+1 view Architechture



→ systems Architechture is perapbs most important to manage different viewpoints & control the iterative & incremental development of system throughout its lifecycle.

→ Architechture is a set of significant decisions about
- organization of software
- selection of structural elements & their interfaces by which system is composed
- Their behaviour, as specified in collaboration among those elements
- The composition of these structural & behavioral elements into progressively larger subsystems
- The architectural style that guides this organization : the static & dynamic elements & their interfaces, their collaborations & their composition

→ Software Architechture is not only concerned with the structure & behaviour, but also with the usages functionality, performance, resilence, reuse, constraints, trade offs, & esthetic concerns.

usecase view → encompasses behaviour of system as end user.

design view → contains class, interfaces, collaborations from vocabulary

Process view → encompasses threads & process from system's concurrency & synchronization mechanism

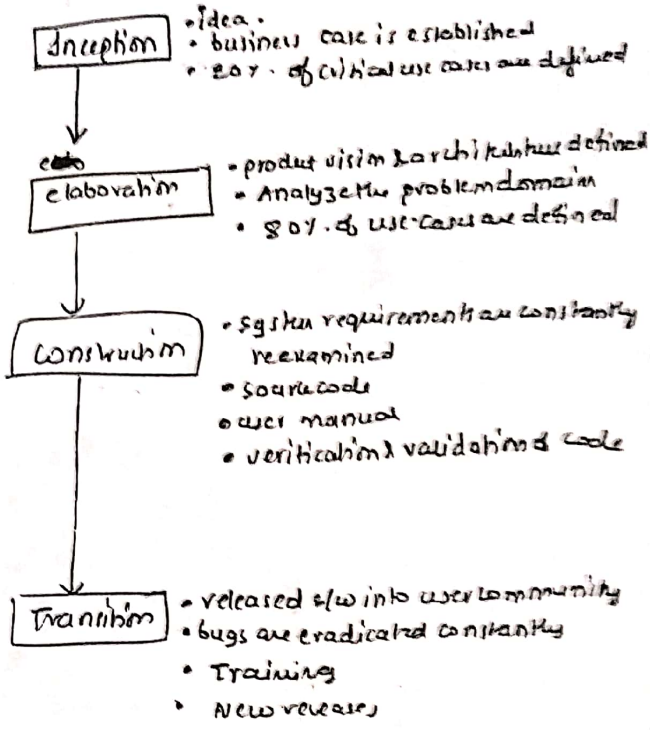
Implementation view → encompasses components & files to assemble & release the physical system

deployement view → encompasses nodes that form systems hardware topology in which system it is being executed.

* Each of these five views are standalone

Software development lifecycle (SDLC)

- uml is largely process independent, but a Process must be
  - use case driven
  - Architechture - centric
  - iterative & incremental



- Iteration cuts across all four phases
- An iteration is distinct set of activities, with a baselined plan & evaluation criteria in each result, either internal or external

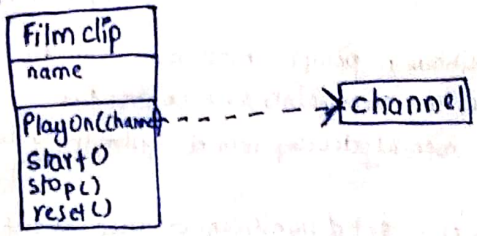
Relationships → definition, types & ?

Common modelling techniques : - (under relationship)

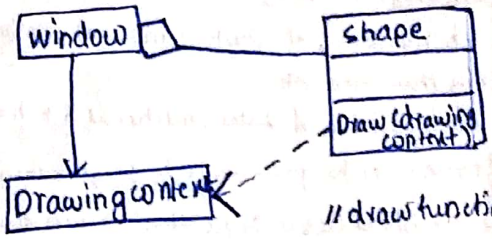
→ modelling simple dependency
→ modelling single inheritance
→ modeling structural relationships.

- modeling simple dependencies (diagram)

ex: 1



ex: 2

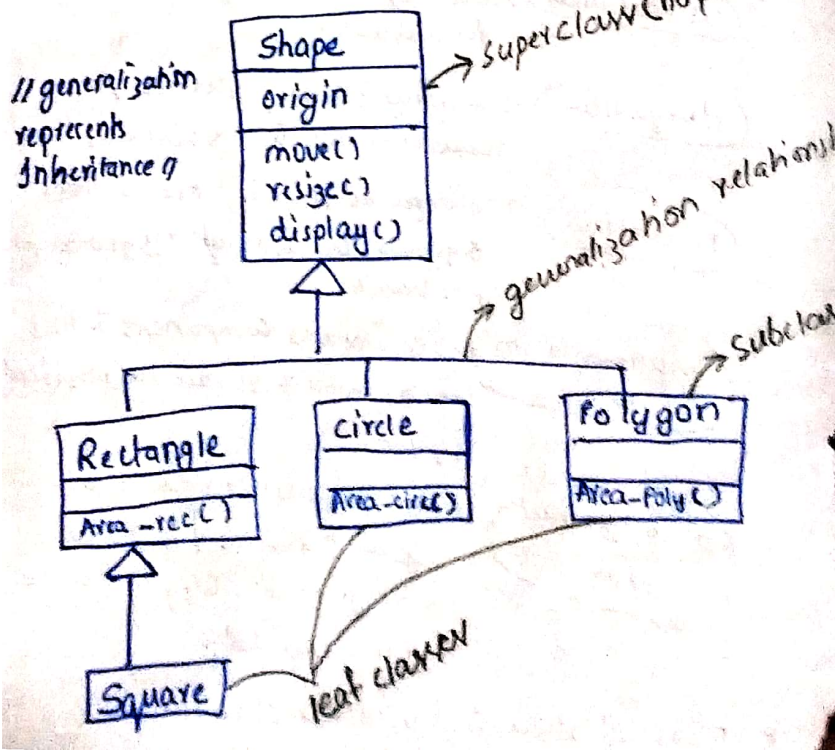


// draw function of the shape class takes an argument of type drawing content //

(or)

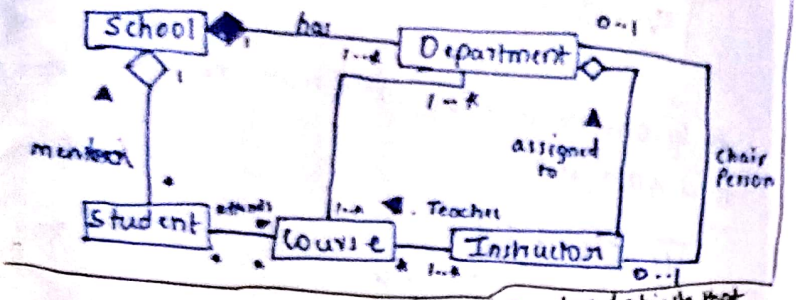
// shape uses drawing content //

- modelling Inheritance / single Inheritance



* In Inheritance, we we generalization.

- modeling structural relationships (diagram)

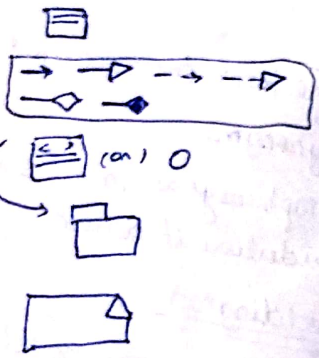


Class diagrams → class is a description of set of objects that share same attributes, operations, relationships & semantics. A class implements one or more interfaces

* Static design view of the system.

Terms / Concepts

Notations
- → Class
- → Relationships
- → Interface
- → Package
- → note



- every class has → name → attributes → operations → responsibilities.

Common modelling techniques (for the class)

1. modeling the vocabulary of system → set of classes, interfaces & etc.
2. modeling simple collaborations
3. modelling logical database schema

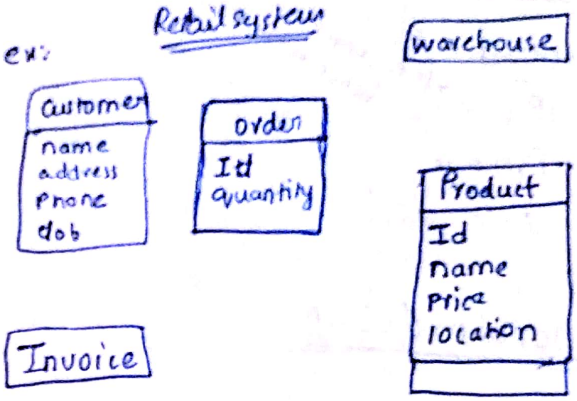
> modeling the vocabulary of system involves the identifying class names, attributes & operations

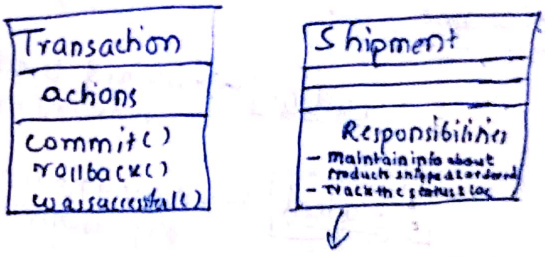
> modeling the simple collaborations involves the identifying the relationships between various classes

> modeling logical database schema involves the description of how the data is stored logically. Tagged value attributes are used as an extensibility feature to represent logical database schema

> every technique above has 2 things
1. steps to model the technique
2. diagrams

ex: Retail system





Responsibilities helps in identifying operations & attributes.

Step 1 :- Identify class names

Step 2 : Identify attributes

Step 3 : Identify operations

Step 4 : If there is any difficulty identifying attributes & operations, we can use responsibilities.

- In designing logical database schema, we use forward engineering & reverse engineering

forward engineering → we take a set of primitives & build into a working system & then observe what system can do & cannot do

reverse engineering → to reverse the exact opposite

Object diagram : → contains objects & links

* to represent the design view of the system

notations :

→ Object
→ link → is a connection between two objects
→ Package
→ note.

Representation of object

Simple name :

S : student → named object representation

(or)

: student → Anonymous object representation

(or)

S → Orphan object representation

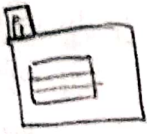
agent : → orphan instance (or)

: student → multi-object.

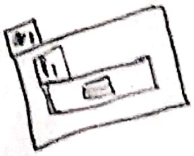
A Student class

Path name :-

P1 :: S : : student



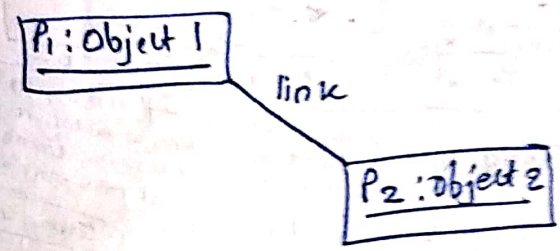
P1 → is a package.
S → object
student → class
if P1 is inside the package q1



then we represent as

q1 :: P1 : : student

* only one relation is possible from one link to another link



* The terms instance and object are synonyms.
graphically an instance is represented by underline its name.

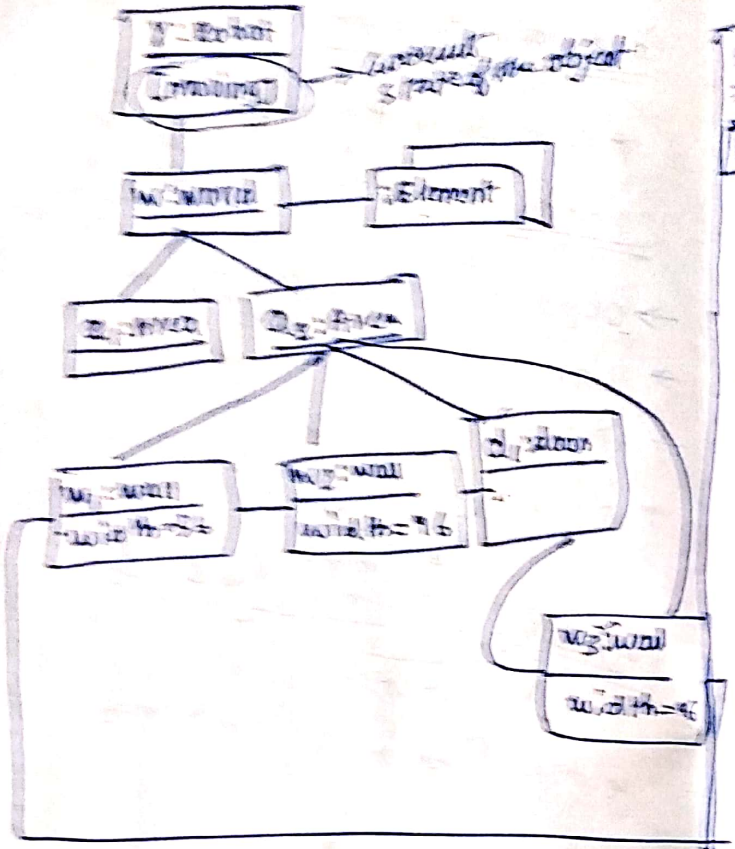
Object diagram :

* Object diagrams model the instances of things contained in class diagrams
* An object diagram shows a set of objects and their relationships.
* you use object diagrams to model the static design view or static process view of a system.

Common modelling techniques :-

→ modeling object structures.

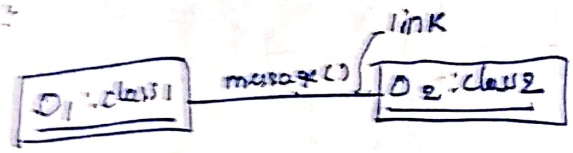
- identify the mechanism (function or behaviour)
- for each mechanism, identify classes, interfaces & other elements.
- consider the scenario
- expose state & attribute values of each object
- expose links among objects



Object :- link :- message :-

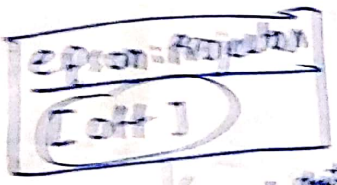
P1 : Person — Specification of communication between one object to another

Ex :-



* we can use the message for the communication

Instance : An instance specification describes an entity based on corresponding class specification



Current state of object

Interaction diagrams

① Interaction is a behaviour that comprises a set of messages exchanged among a set of objects within a context to accomplish a purpose

② Interaction diagram is a diagram that models dynamic aspect of the system (or) dynamic view of the system.

→ Sequence diagram
→ Collaboration diagram
} Isomorphic

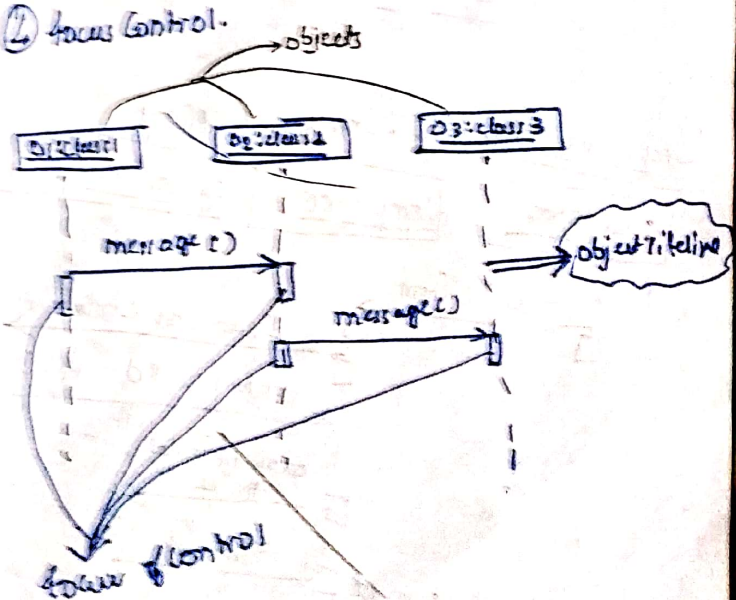
③ Sequence diagram - It is used to model the time ordering of messages

④ Notations :-

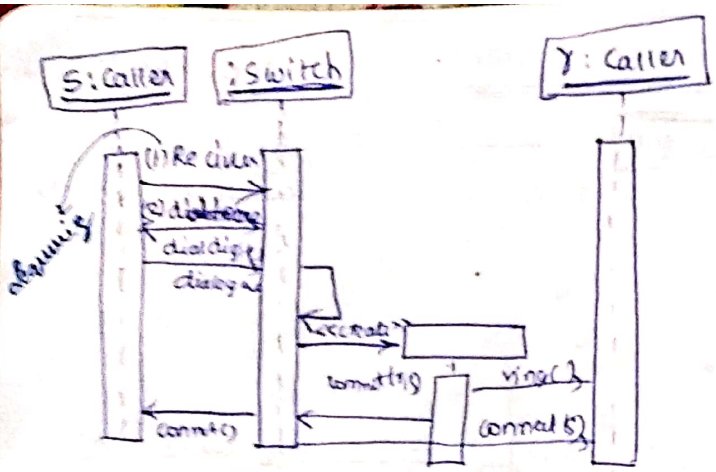
→ object
→ link → is a connection between two objects
→ message
→ Activation
→ time

Basic steps in Identifying Sequence diagram

① Identifying the objects
② messages
③ object lifeline
④ focus control.



* In place of message you can use <<destroy>> <<stereotype>>, calls, call(), return value, return



Call Action → invokes an operation on an object. An object may send a message to itself resulting in local invocation of operation

Return → Returns a value to the caller

Send → Sends an signal to an object

Create → creates an object

Destroy → <<destroy>> destroys an object; and object may commit suicide by destroying itself.

⇒ Draw a sequence diagram for loan registration.

Collaboration diagram

* to modeling flows of control
  - set context for interaction
  - set stage for interaction
  - connect relevant parts
  - in time order specify message
  - convey the message.

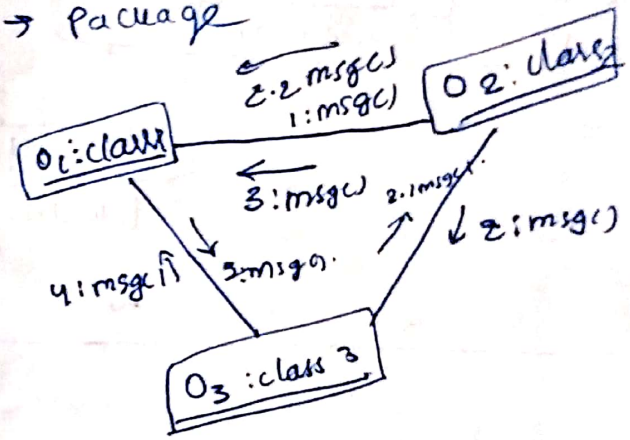
notations :-

→ object

→ link

→ message

→ package

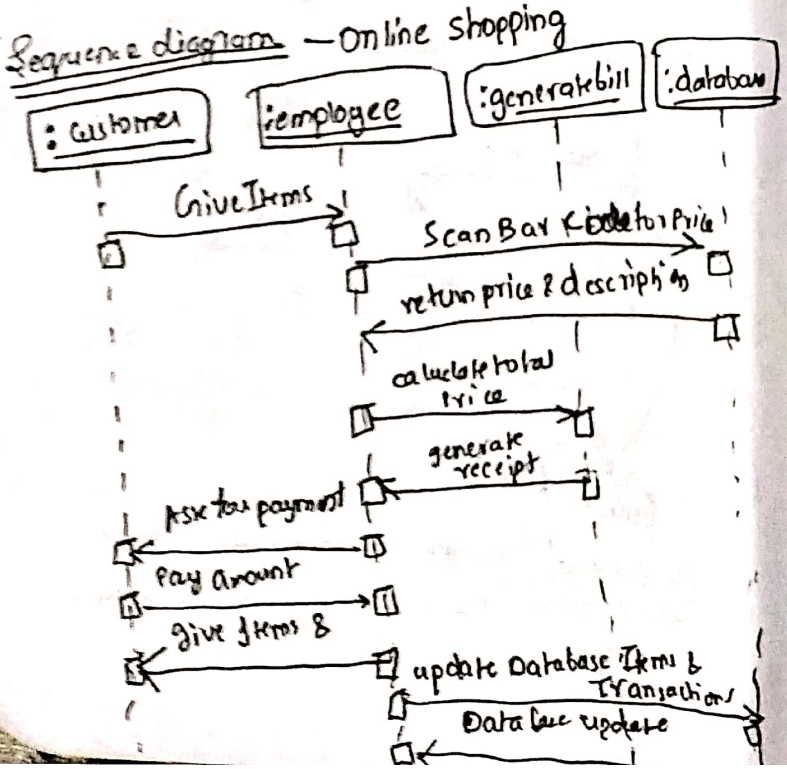


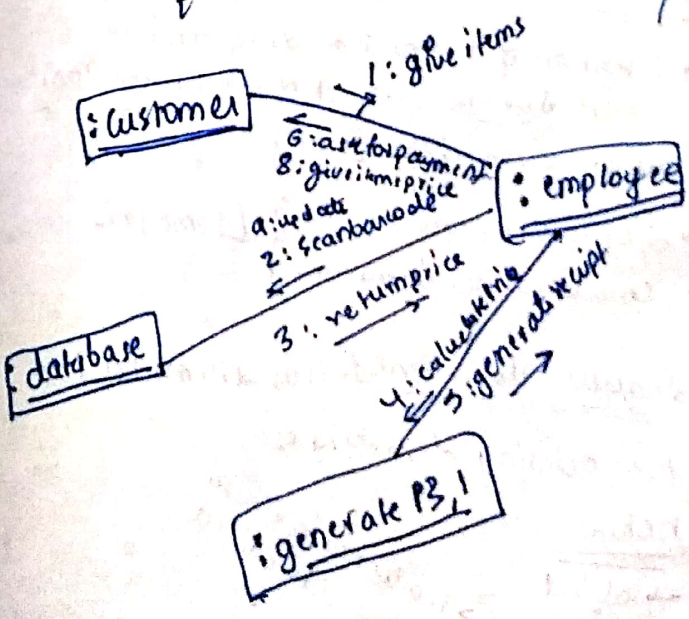
Sequence numbering

① Procedural (or) nested flow of sequencing

② Flat flow sequencing (or) non procedural sequencing

Sequence diagram — Online shopping



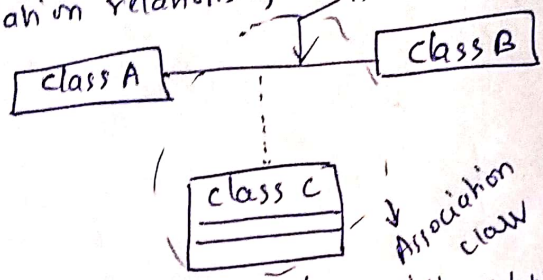


Association class

① essentially a class attached to an association

② It is used to model the property of an association relationship

eg:

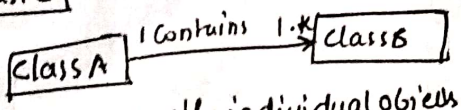


- Association class is a part of association relationship between 2 classes.

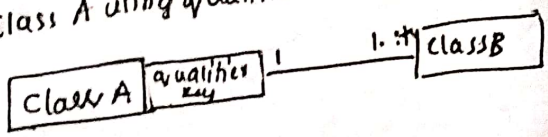
qualified association (optional) / qualifier key

- Qualified association has a qualifier that is used to select an object or objects from a larger set of related objects based upon qualifier key.

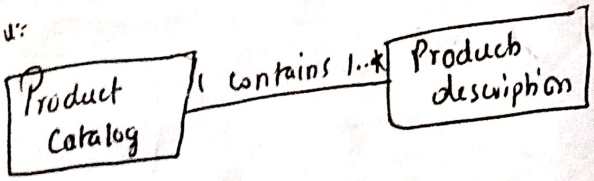
Basic :-

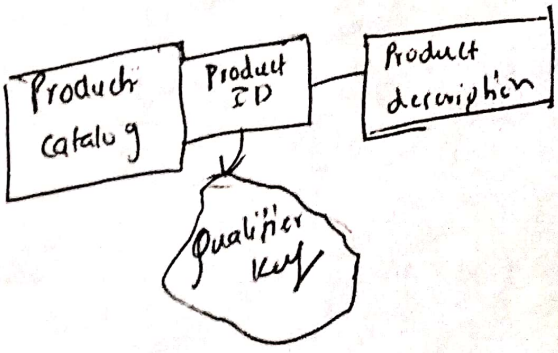


- we can identify individual objects within Class A using qualified association.

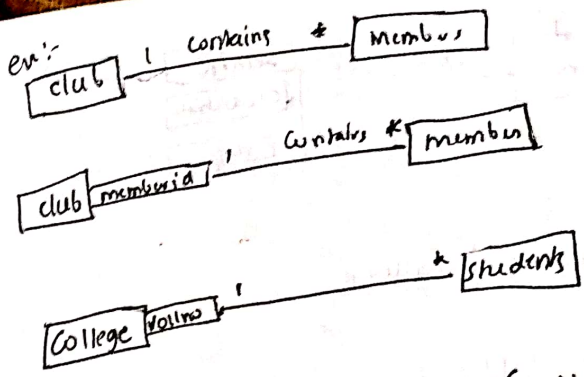


ex:-





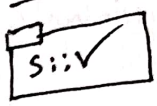
ex:-



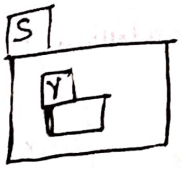
Package :- collection of elements (9 diagrams)

Simple :-

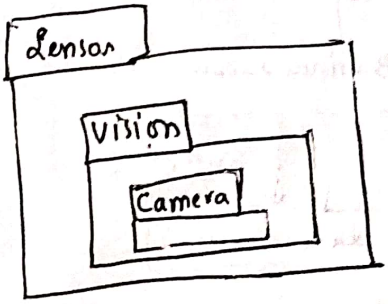
Path name :-

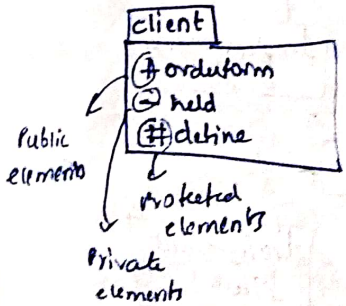


(or)

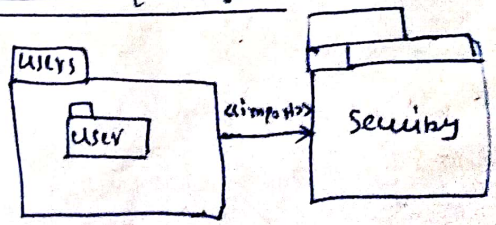


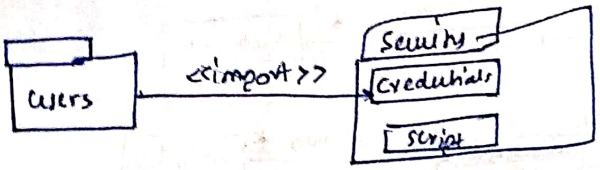
ex:-



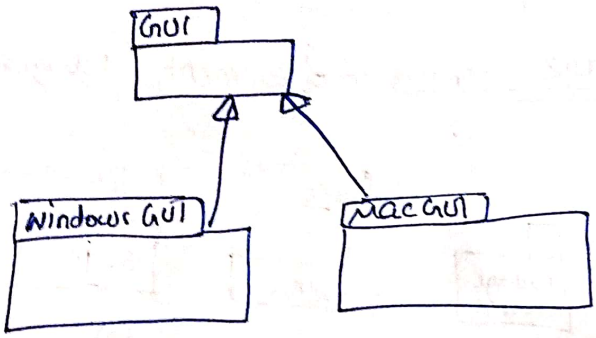


Importing the package

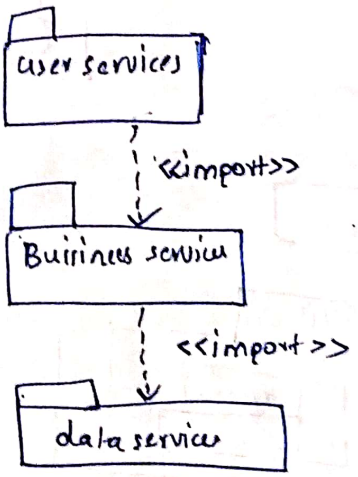




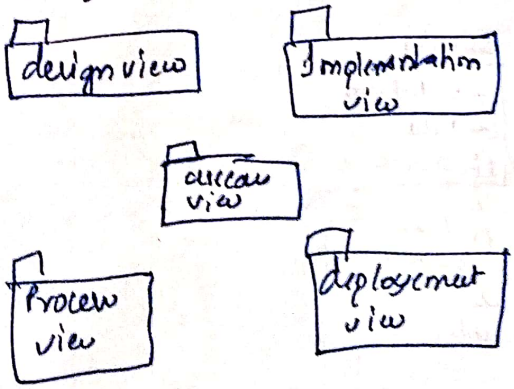
generalization in packages



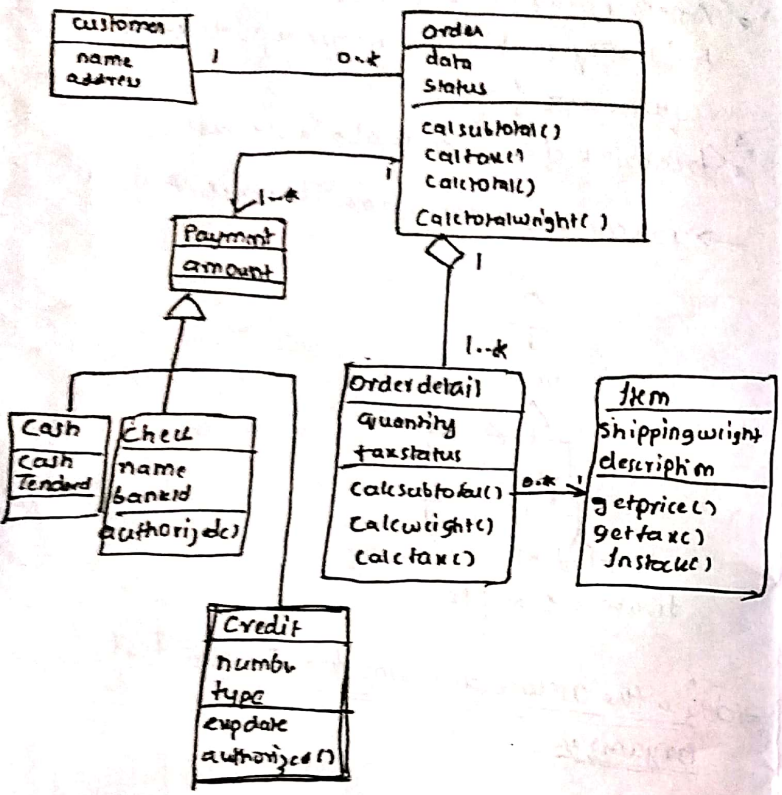
1. modelling groups of elements



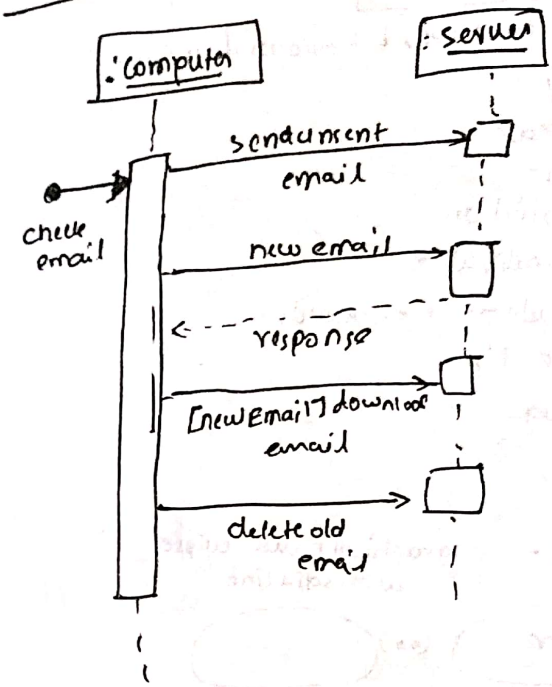
2. modelling Architechtural views



Class diagram example



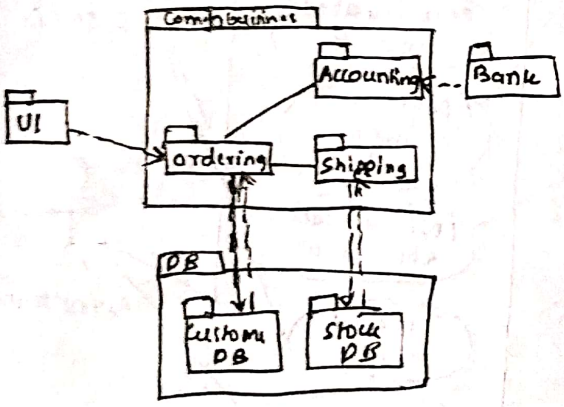
Sequence diagram



⇒ class diagram : college festival organizing diagrams

College
Student
event
event managers
Volunteers
Audio

Package diagram



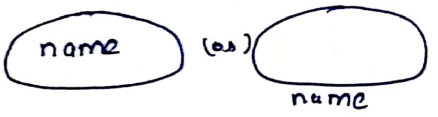
UNIT-3

Usecase diagram

* used to model the behaviour of a system

notations:
- → use case
- → Actor
- → Association
- → generalization
- → <<include>> & <<extend>> relationship
- → Package
- → note

use case :- graphical name :- ellipse with said line

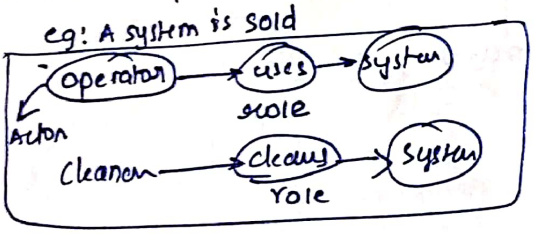


Actor:

graphical name : stick figure

- Actors should be identified from discription
- Actors represents a 'role'

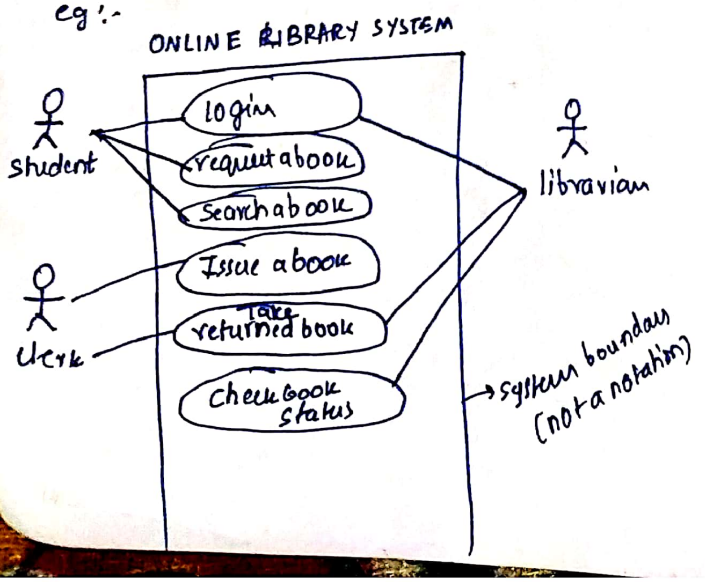
eg: A system is sold



- Actors are outside of the system

* A servicel feature provided by system can be individually represented as a usecase.

eg :-

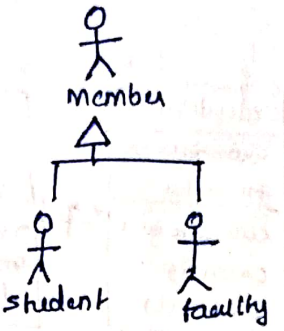


* we can either identify actors first or the usecases first

* Another way of identifying a usecase is to identify that "every usecase will describe requirements of a system".

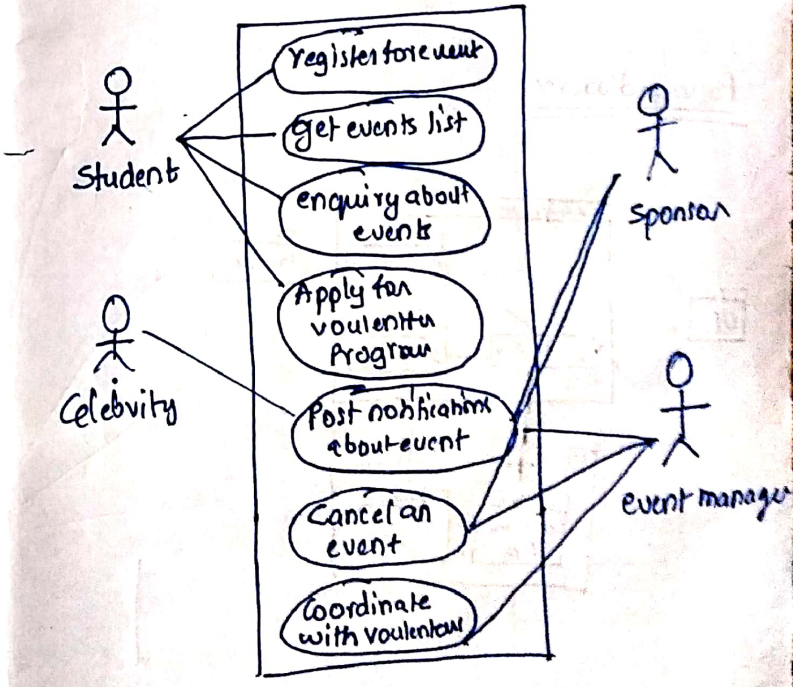
* behaviour of a system is also a usecase

→ you can also generalization between Actors



In above example

⇒ draw the usecase diagram for College fest organizer



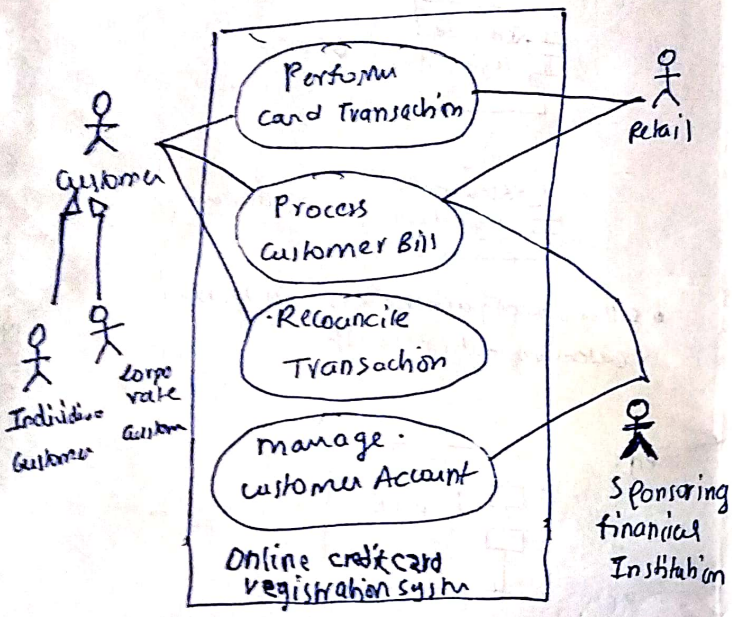
Note: 1. Actors must be connected to actions only using Association

2. A usecase captures the intended behaviour of system

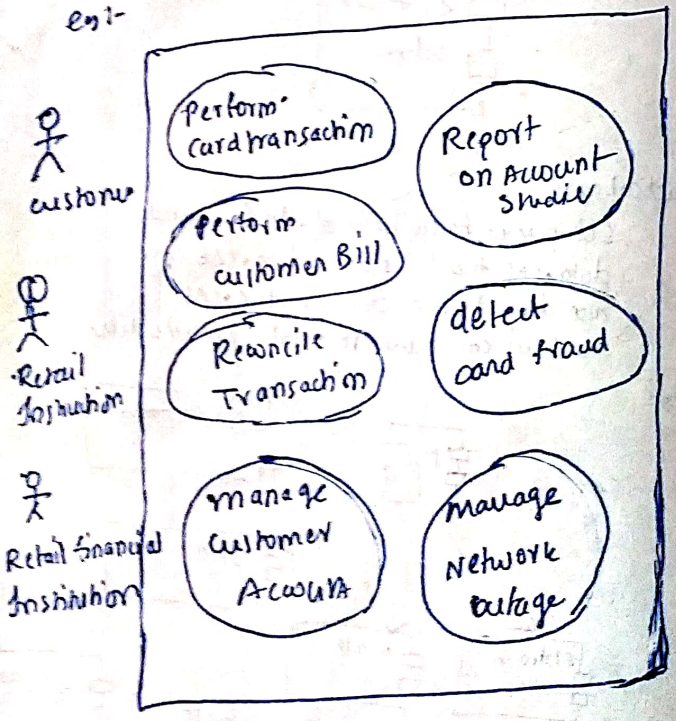
Subsystem :- A subsystem is simply a part of a system that is used to decompose a complex system into independent parts

Use case diagram:- common modelling technique

→ modelling requirements of a system
→ modelling context of the system

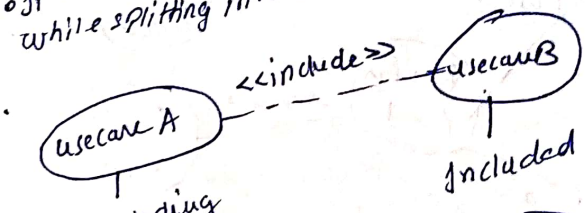


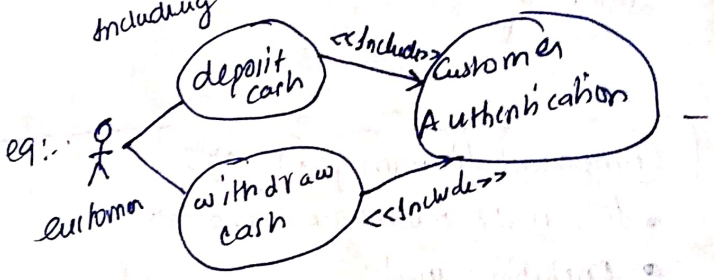
eg:-



① <<Include>> Relationship :-

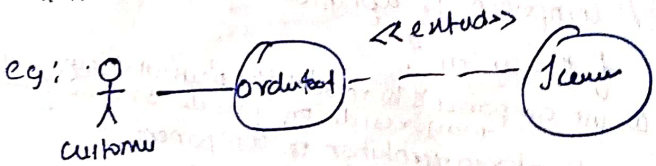
Include is a directed relationship between two usecases which is used to show that behaviour of the included usecase is inserted into the behaviour of the including usecase.
• It could be used to simplify large use case while splitting into several usecases





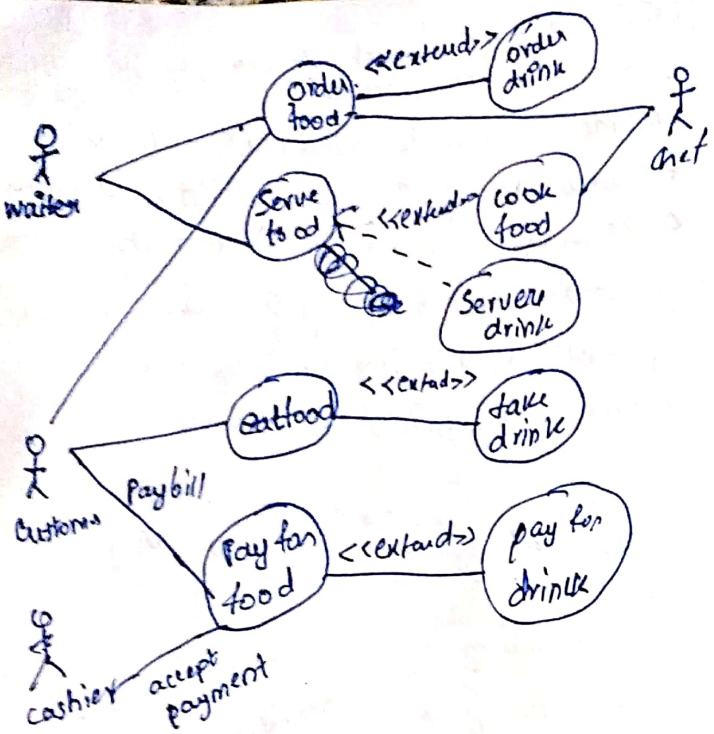
<<extend>> : is a directed relationship

that specifies how & when the behaviour defined usually extending usecase can be inserted into the behaviour defined in the extended usecase.



Use case diagram common modelling techniques

- identify usecases
- Identify packages
- Explain usecases & relationships
- Use note notation for comments
- explain use case diagram



Component diagram

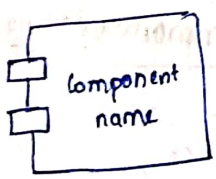
def.. backside

- It is used to model the components
- relations used :-
  - → component
  - → dependency relationship
  - → node
  - → package.

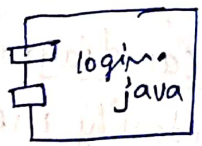
"A component is a physical replacable part of the system".

- we use components to model the physical things that may reside on a node such as executables, ...etc

graphical representation of component :-



eg:-

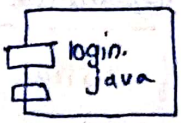


graphical name:- rectangle with 2 tabs

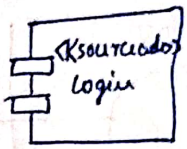
component types

- source code components
- executable components
- library components
- files "
- documents "
- Tables "

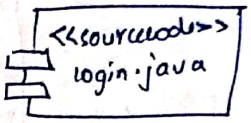
eg:



✓ valid



✓ valid

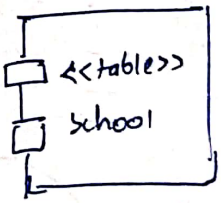


✗ Invalid

- either we can use the <<stereotype>> or
- extension but not both

eg:

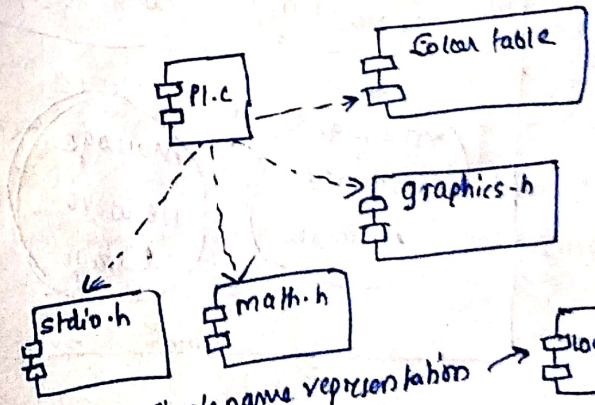
eg:-



Practical eg:-

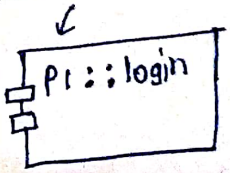
Setup.exe taken helps of .dll files and data.cab files to install. & data.cab can also depend upon data_validate.cab and we can have internal dependencies

eg:

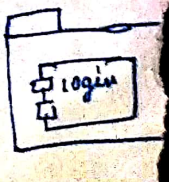


Pathname :

- → Simple name representation → login.java
- → pathname representation



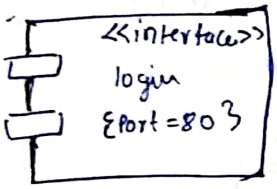
ie, clearly



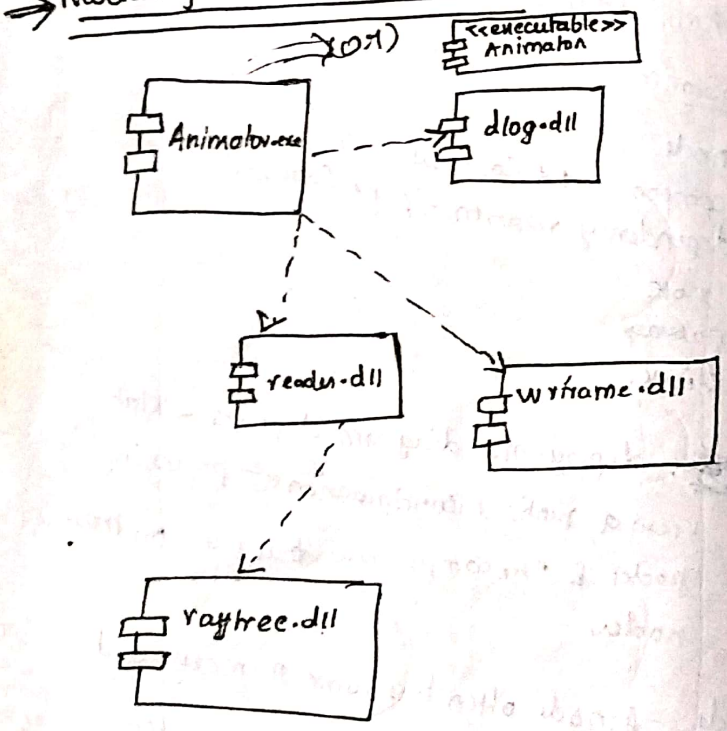
Common modelling techniques (Component diagram)

1. modelling executables & libraries
2. modelling tables, files and documents
3. modelling an API
4. modelling source code
5. modelling an executable releases.
6. modelling a physical database
7. modelling Adaptable systems

eg:-

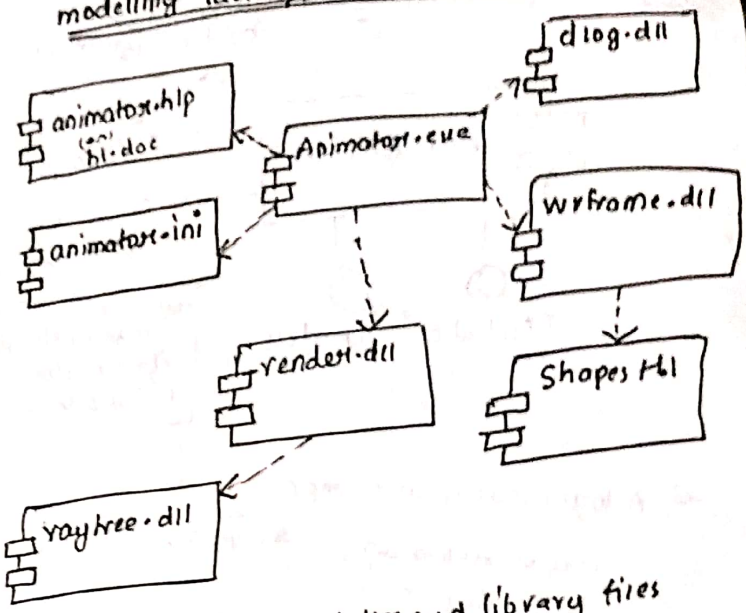


Modelling executables & libraries :-



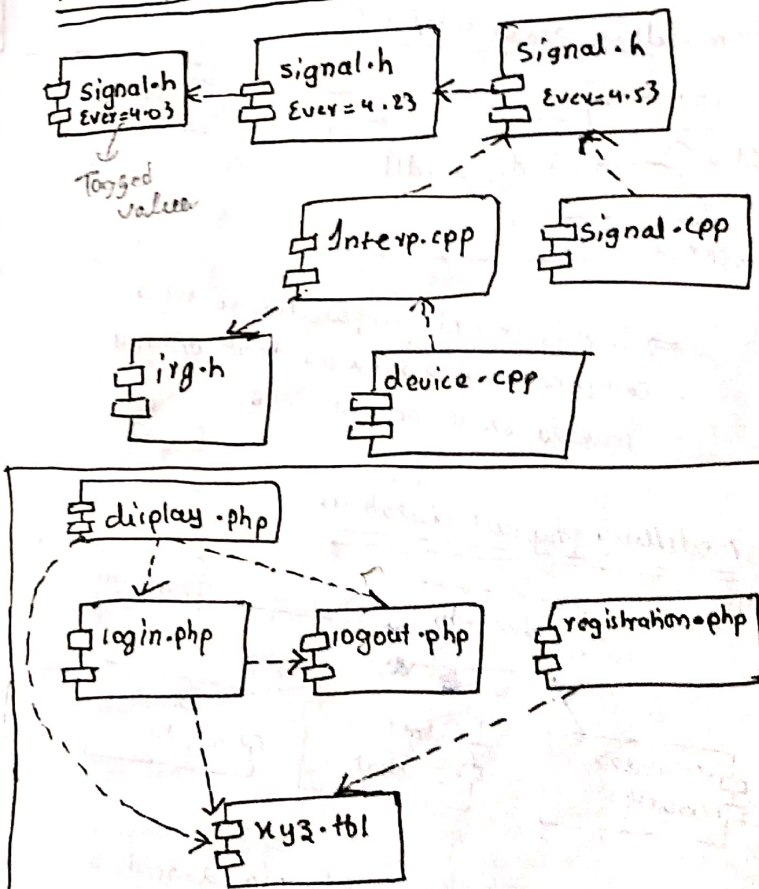
- In this modelling technique the component diagram contains only executables & library files.
- we can use an extension for a component or a <<stereotype>> but not both.

modelling Tables, files & documents



- along with the executables and library files we have documents, files & tables.

modelling source code :-

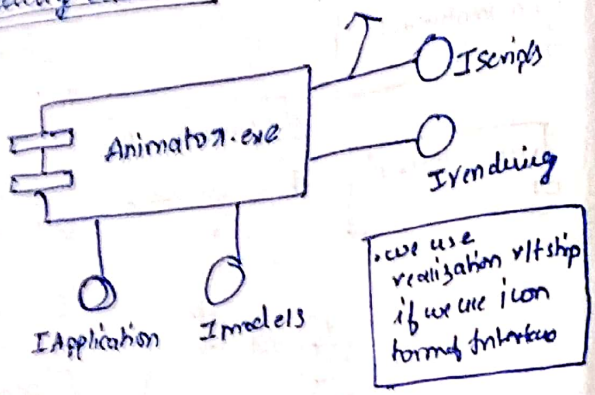


example to understand working of component

*using component diagram, we can understand how changes in one component affects the other components, which is one of the advantages of component diagram.

modelling an API

Realization relationship.

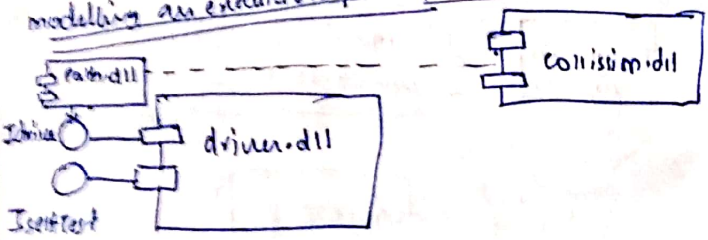


• we use realization r/ship if we use icon forms interface

⇒ A login page is an example .exe will be running, but the user works with the interface

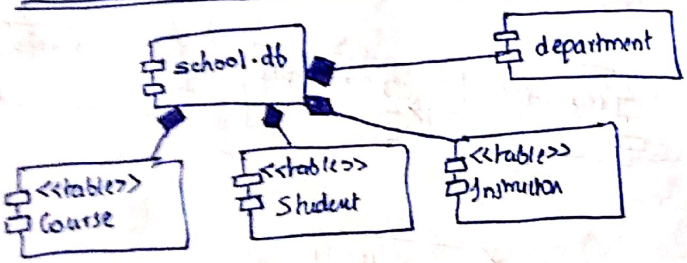
⇒ Before modelling itself we need to know the working of how interface works.

modelling an executable ~~package~~ release



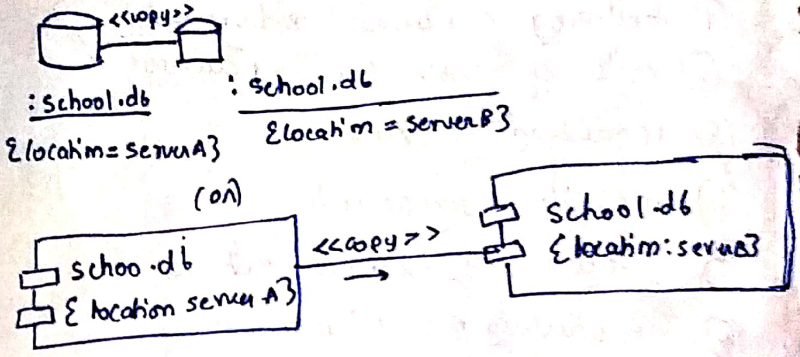
⇒ In this technique, we deal with the components of a software that contains installation component files.

modelling physical database



• all these components can be placed inside a node

modelling Adaptable systems :-



• A replica is available and in the case of failure the replica can be replaced back again.

• Adaptable systems are used in banks

Deployement diagram :

→ is a structural diagram

→ is used to model system Architechture, hardware Architechture or deployement view

notations :

→ node

→ component (& Optional)

→ dependency relationship (& Optional)

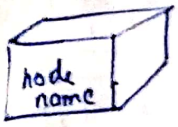
→ note

→ package

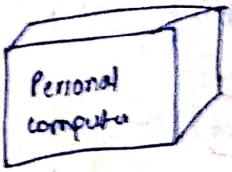
→ Link.

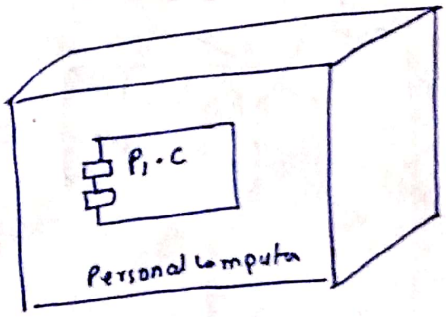
: deployement diagram depicits a static view of runtime configuration of processing nodes & the components that run on those nodes

node :- A node often has some memory and processing capabilities. A node is a place where we can execute components.

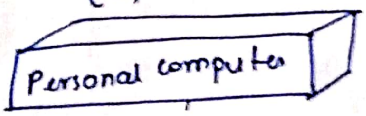


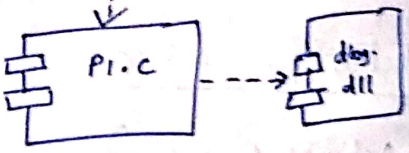
eg:



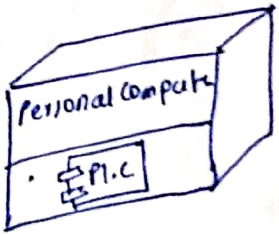


(or)





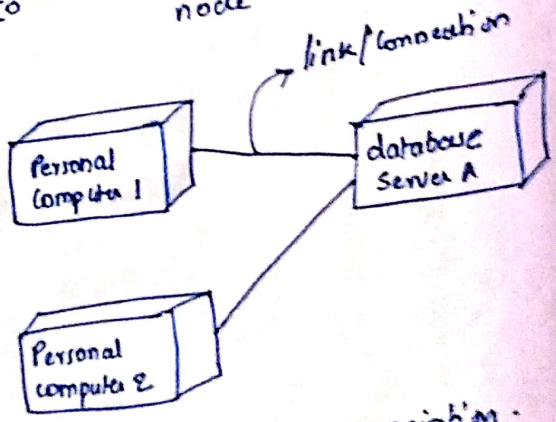
(or)



(not possible in tool)

link :- Connection between one node to another node

eg:



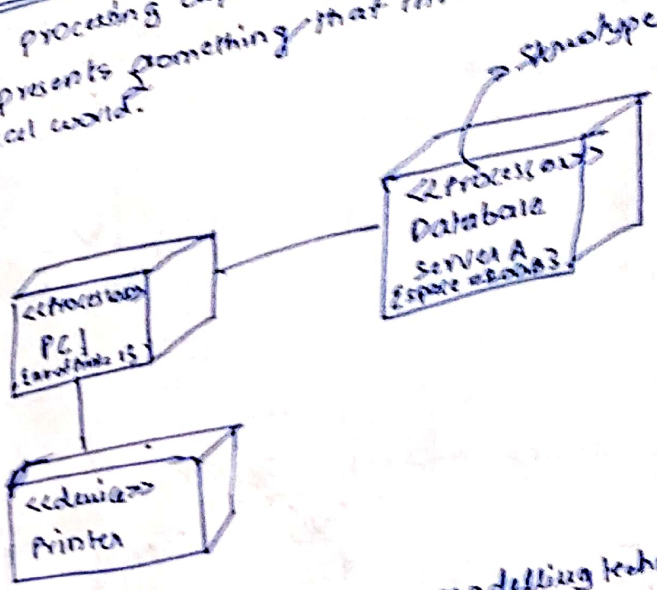
Note: link is not same as association.
The deployement diagram a solid line is called link

- There are two types of
  1. device node
  2. Processor node

Processor node :- A processor node that has processing capability, meaning that it can execute a component.

device node: A device node that has no processing capability, in general represents something that interfaces to the real world.

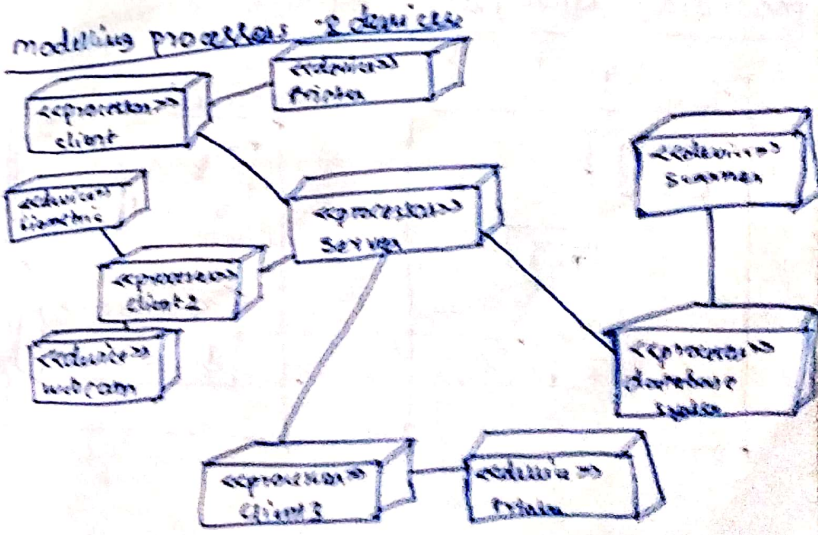
eg:-



Deployement diagram (common modelling techniques)

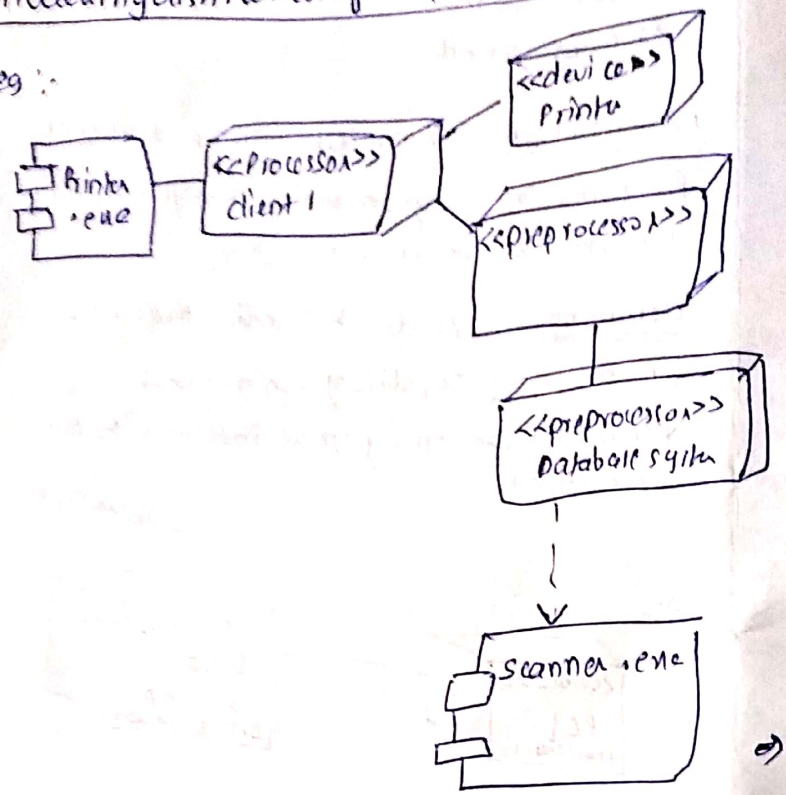
- → modelling processors & devices
- → modelling distributions of components
- → modelling Embedded Systems
- → modelling client/server systems
- → modelling fully distributed system

modelling processors & devices

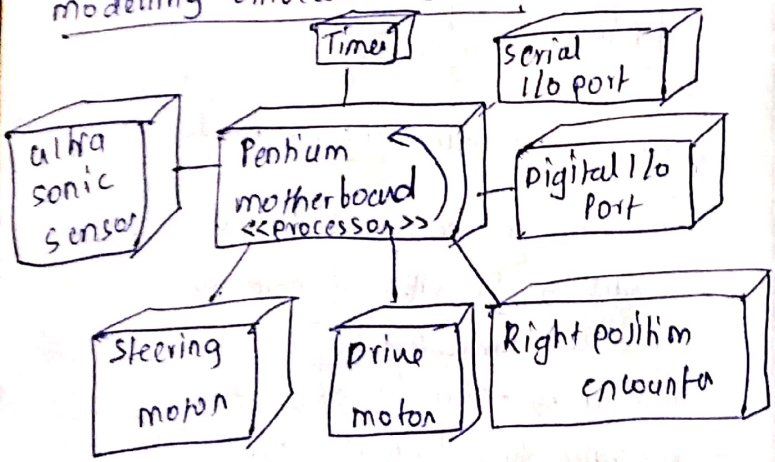


modelling distribution of components

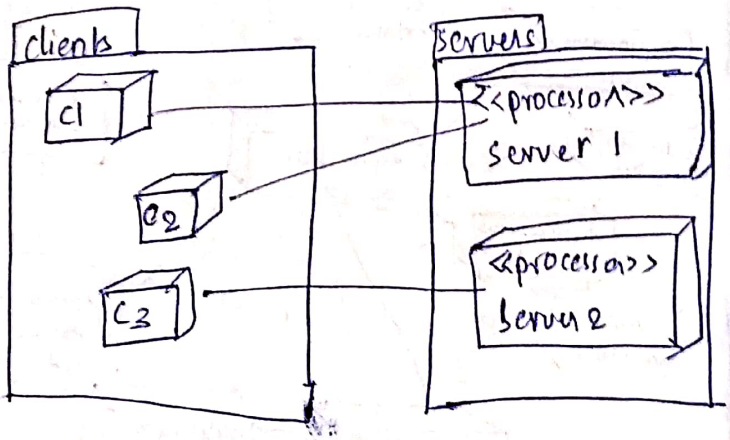
eg :-



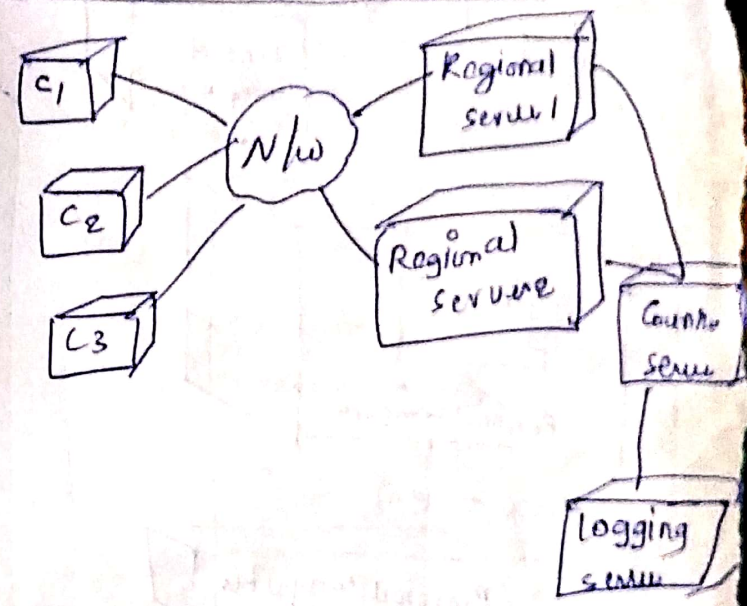
modelling embeeded systems.



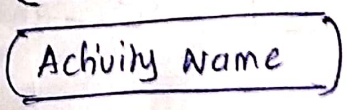
modelling client/server systems.



modelling fully distributed systems



→ Activity (or) Action :



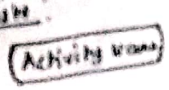
Activity diagram:-

- used to show business flow or an activity flow, but this is not same as flowchart.

notations:

- → Activity (or) Action
- → Transition (or) - control flow
- → Initial state (or) starting state
- → final state (or) ending state
- → decision
- → merge
- → fork and join
- → Time event
- → swimlance
- → note
- → package

Activity:



Transition:

Initial state:

final state:

decision:

fork:-

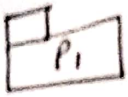
merge:-

join:

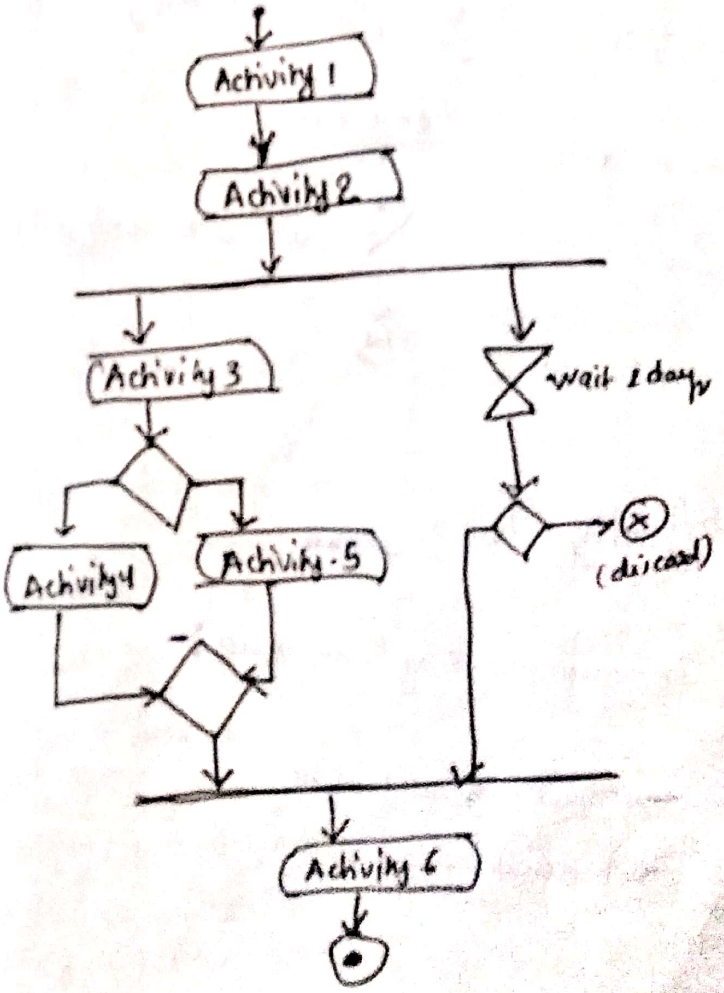
Time event:

note:

Package:



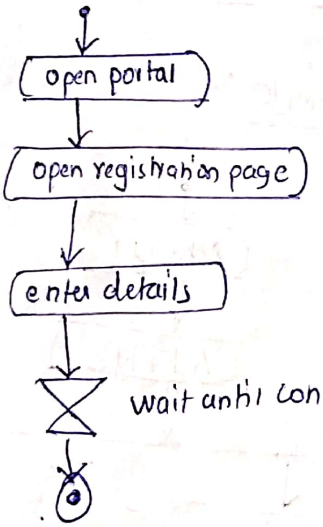
Basic Example :-



Phases :→

1. Identify initial & final state
2. Identify Activities or Actions based on given description
3. use controlflow or a transition flow between Activity to Activity
4. If required, use decision box
5. use merge, fork, join and line segments if required based on the situation.
6. (optional) if Required use swimlands concept

eg:. (simple) Student enrollment



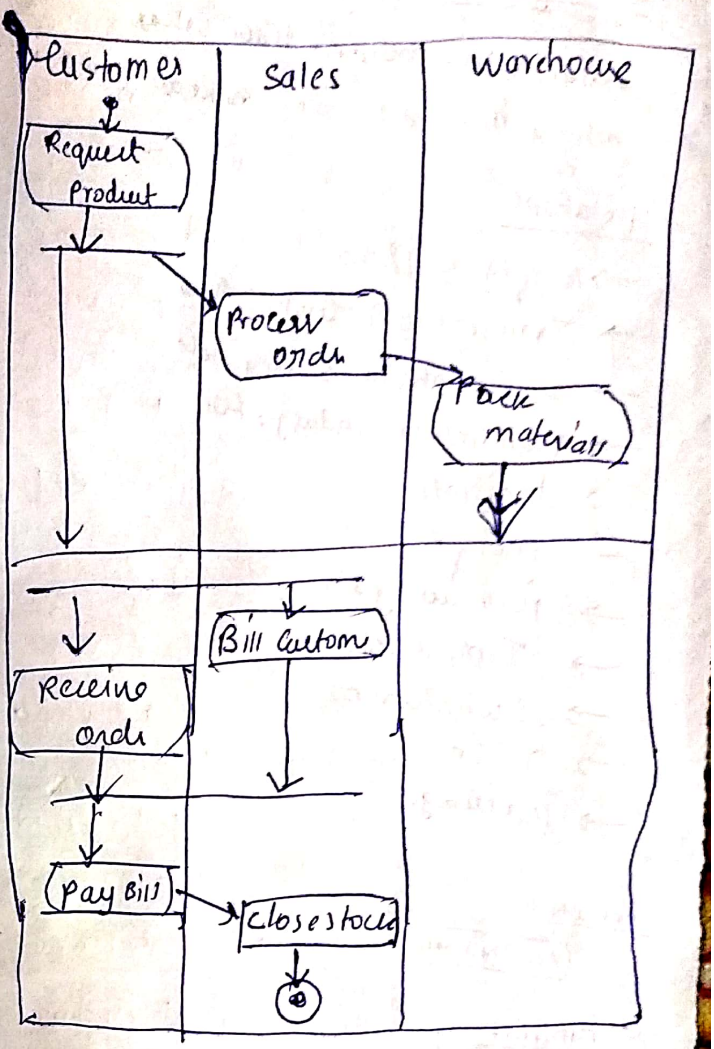
Activity Diagram

Common modelling techniques :-

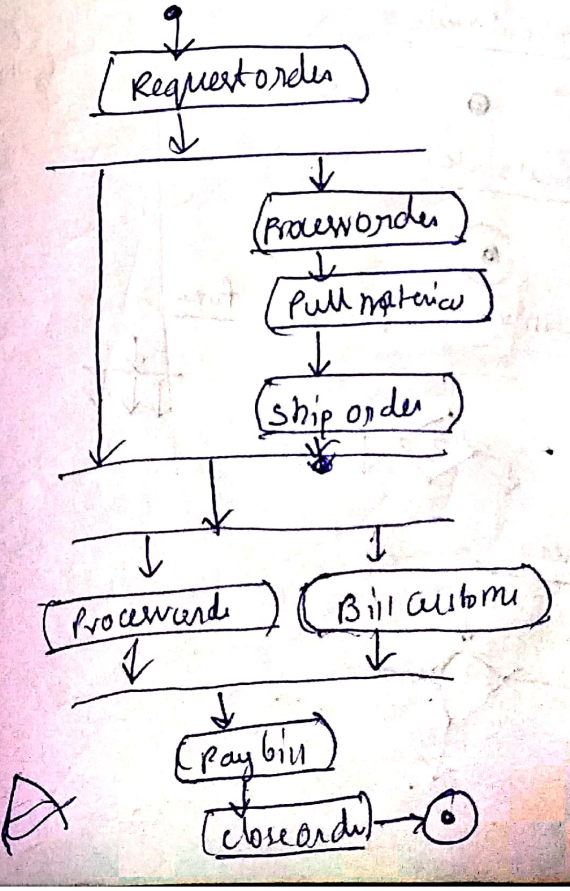
→ modelling workflow
→ modelling object flow

modelling workflow swimlane :-

eg.



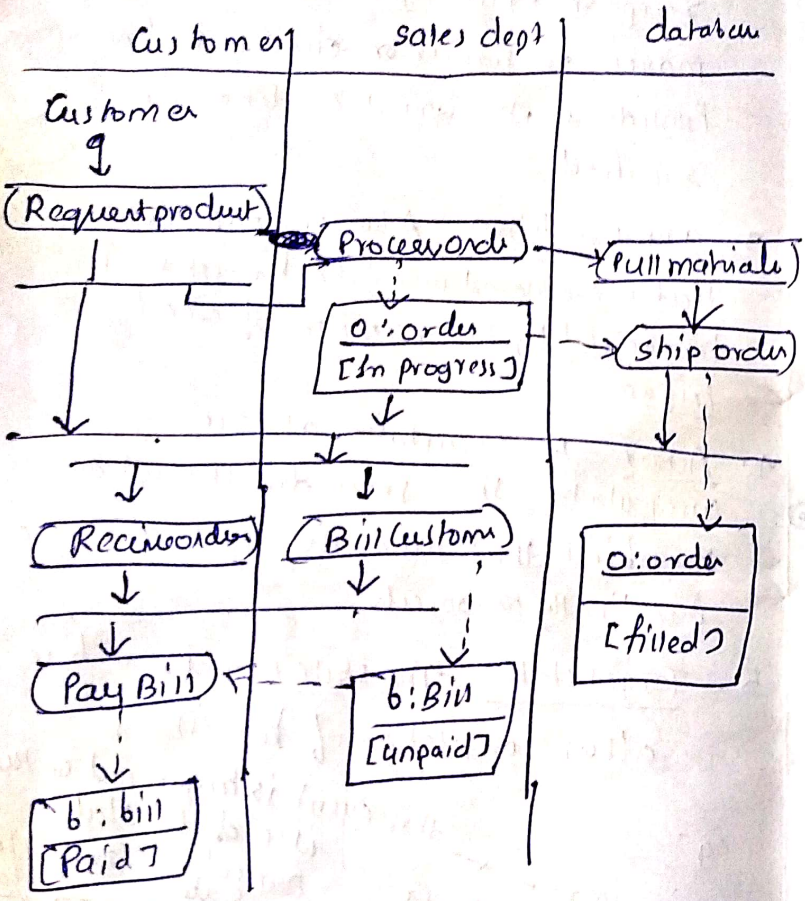
without swimlane



⇒ ~~State chart diagram~~

Modeling object flow

- - - ⇒ (represents the object flow & state of the object)

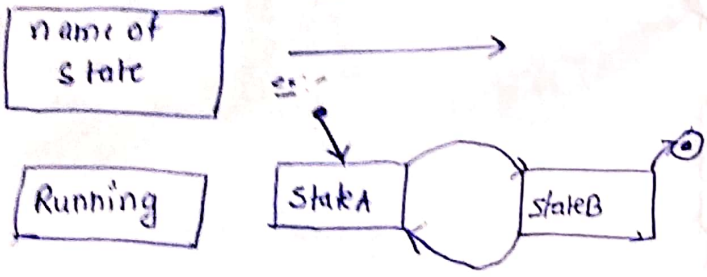


Statechart diagram :

Notations :

→ State Machine (or) state
→ Transition
→ initial (or) starting state
→ final (or) end state
→ history state
→ note
→ Package.

State :



Transition :

- A state is a condition or a situation during the life of object during which it satisfies some condition, performs some activity (or) wait for some event
- An event is the specification of a significant occurrence that has a location in time and space.

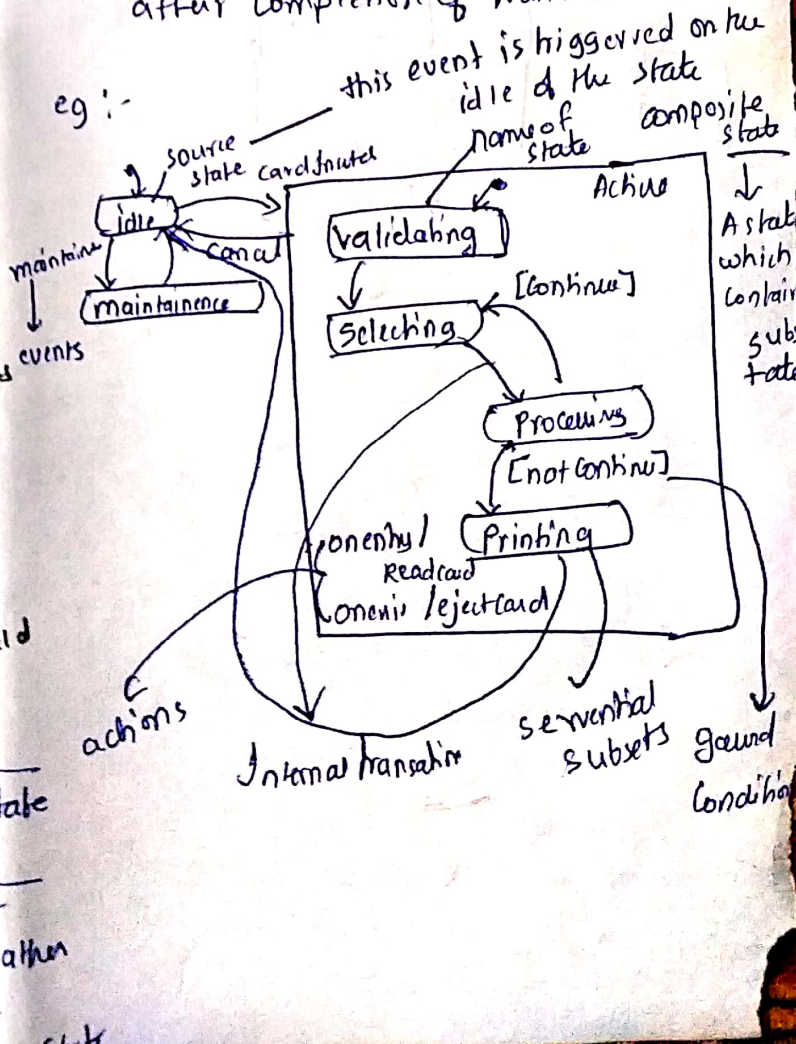
Parts of a state :-

	description
name	→ A text String that distinguishes a state from other state
entry / exit Action	→ Actions executed on entering and exiting from state
Internal Transition	→ Transitions that are handled without causing a change in the state
Sub states	the nested structure of a state
Events	a list of events that are not handled in that state but rather are postponed & queued for handling by object in another state

Parts of a transition

1. Source code : The state affected by the transition
2. Event Trigger :- The event whose reception by the object & the source state makes the transition eligible to fire Providing its gaurd condition is satisfied
3. gaurd condition : A boolean expression that is evaluated when the transition is triggered by the reception of event trigger.
4. Action : An executable. atomic computation that may directly act on the object that owns the state machine & indirectly to objects.
5. Target state : The state that is active after completion of transition.

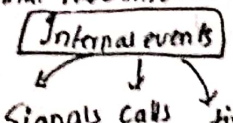
eg :-



Events & Signals

External event : event between system & its internal actors

Internal Events : events that pass among the objects that live inside the system.



* **A time event** : that represents the passage of time

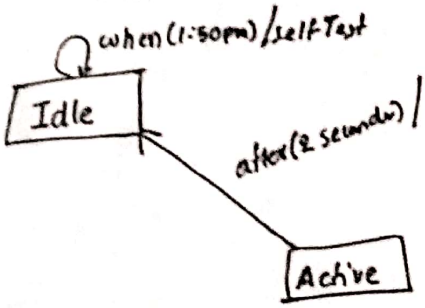
→ In UML, you model a time event by using a keyword "after" followed by some expression that evaluates to a period of time.

eg: "after 2 sec"

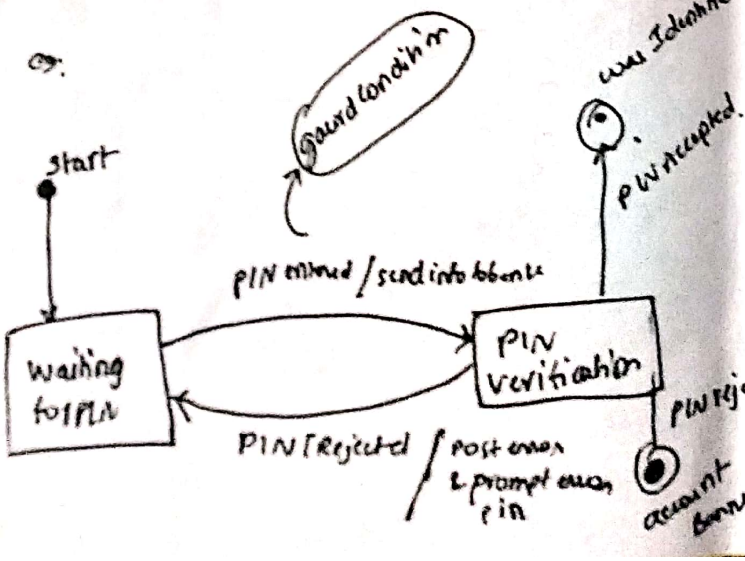
* **Change event** :- is an event that represents a change in state or the satisfaction of some condition

→ In uml, you model a change event by using the keyword "when" followed by some boolean expression.

eg:

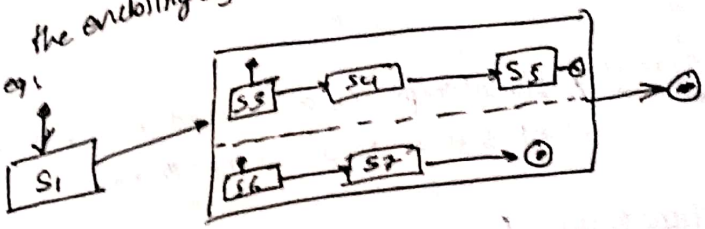


eg:

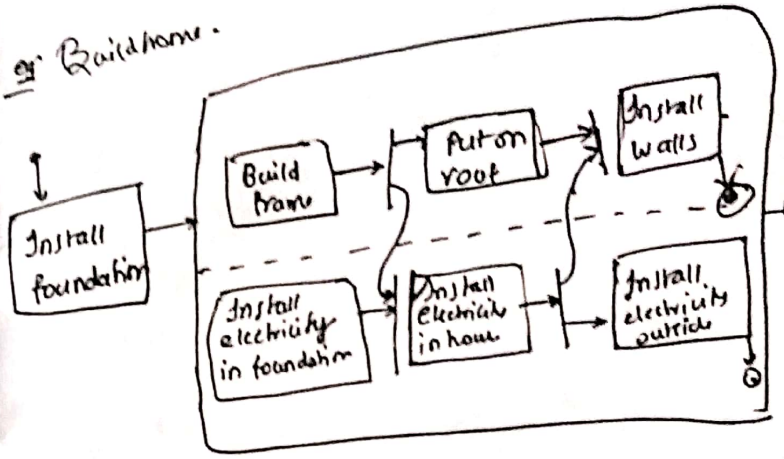


Concurrent substates :- allows us to specify two or more states that execute in parallel in the context of the enclosing object.

eg:



eg: Build house.

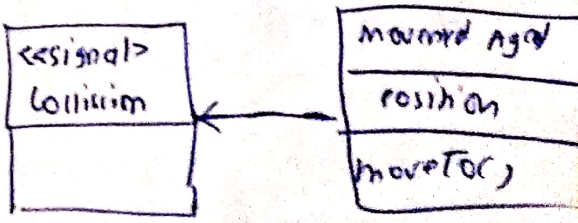


Common modeling techniques :

→ modelling states of an object

→ signals

- A signal may be sent as action of a state transition in a state machine or the send of a msg in an interaction
- The execution of an operation can also called signals
- In the UML, you need the relationship b/w an operation & the events that it can send by using a dependency relationship stereotyped as <<send>>

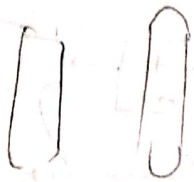


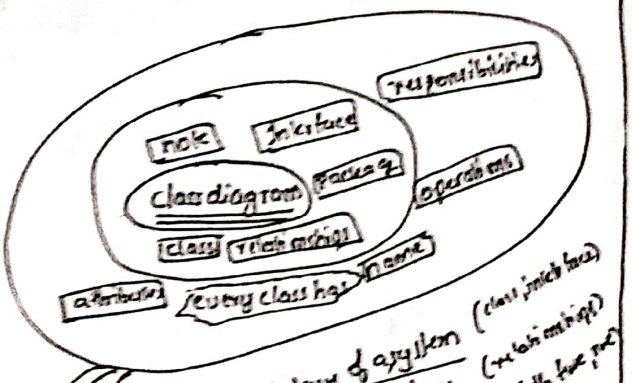
→ Calls events :- A signal event represents the occurrance of a signal

a Call event represents the dispatch of an operation.

Signal A signal represents a named object that is dispatched asynchronously by one object & then received by other.

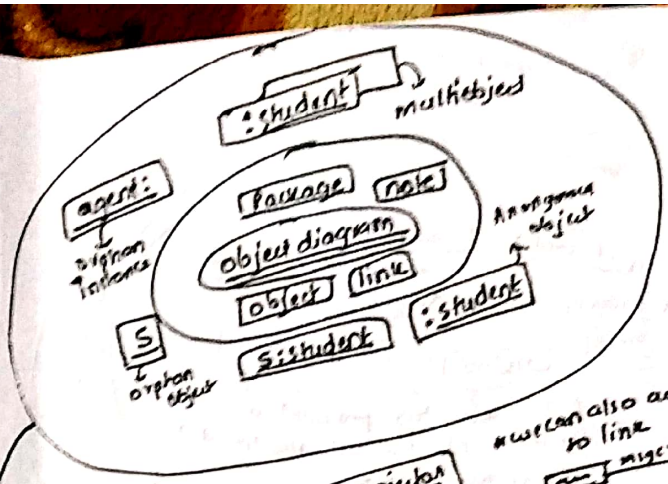
Time event : A time event is an event that represents the passage of time.





modelling vocabulary of system (class, interface)
modelling simple collaborations (relationship)
modelling logical database schema (db table, pk)

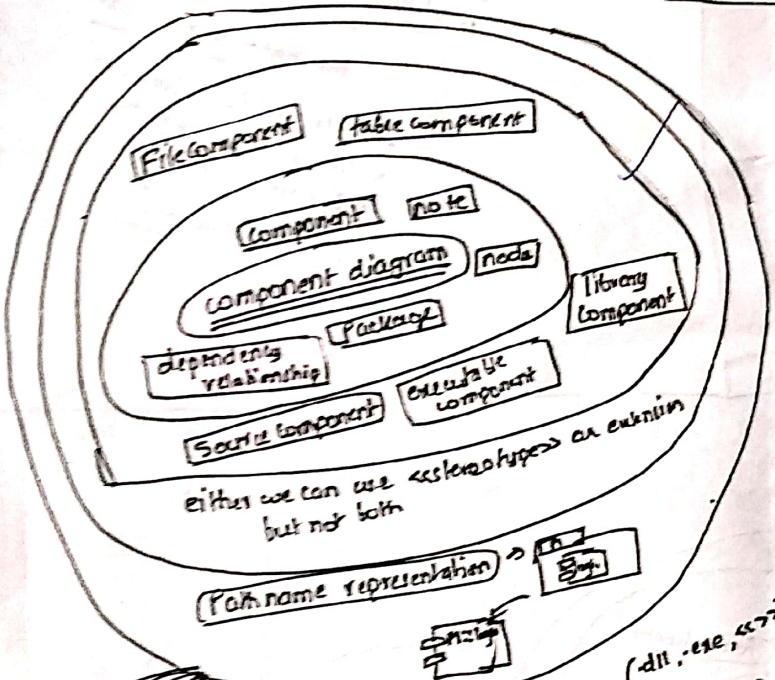
class diagram



object diagrams
→ instance
→ state of object → [option:projector OFF]
→ modelling object structure
- identity mechanism - consider scenario - form links

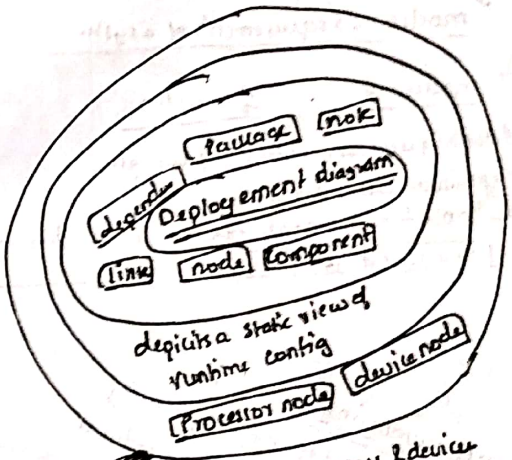
we can also add messages to link

Object diagram



modelling executables & libraries (.dll, .exe, <<>>)
modelling tables, files, documents (.dll, .exe, .tlb)
modelling an API (using interfaces)
modelling source code (showing changes in one component affects another)
modelling executable files (dealing with instantiation of component files)
modelling physical database (using .db & composition)
modelling adaptable systems (using examples of bank replicas)

Component diagram

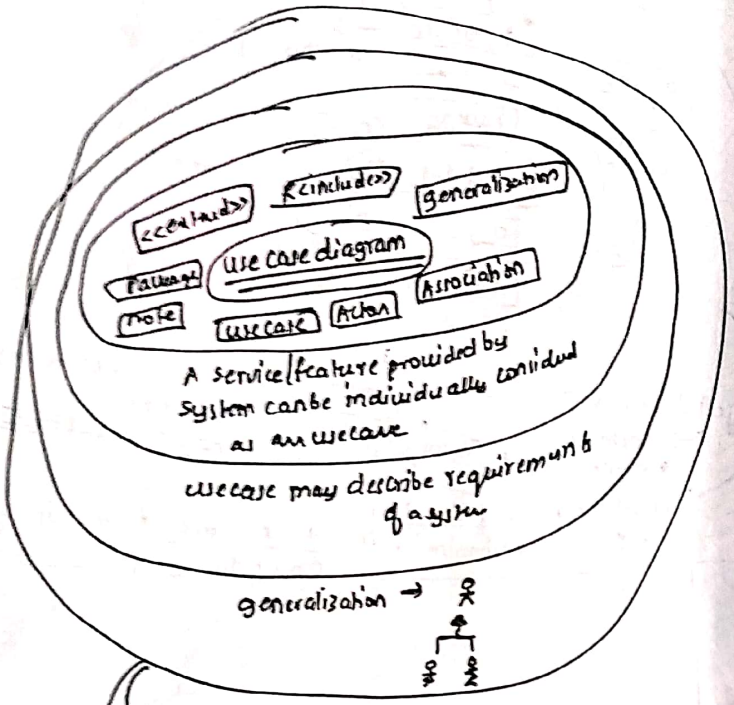


modelling processors & devices
modelling distribution of components
modelling embeeded systems
modelling client/server system
modelling fully distributed system

deployement diagram

Events & signals
→ modelling a family of signals
→ modelling exceptions
↳ Stereotyped classes

Relationships
→ modelling simple dependencies
→ modelling single inheritance
→ modelling structural relationship

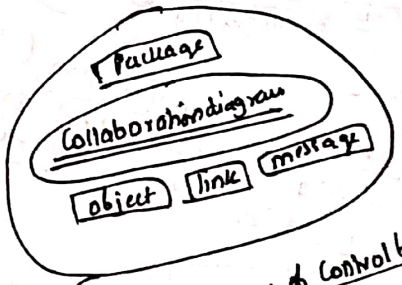


modelling requirements of a system

modelling control of a system

- Identify use cases
- Identify packages
- explain use cases & relationships
- use note for comments
- explain use case diagram

use case diagram



modelling flows of control based on ~~on time ordering of~~ messages organization

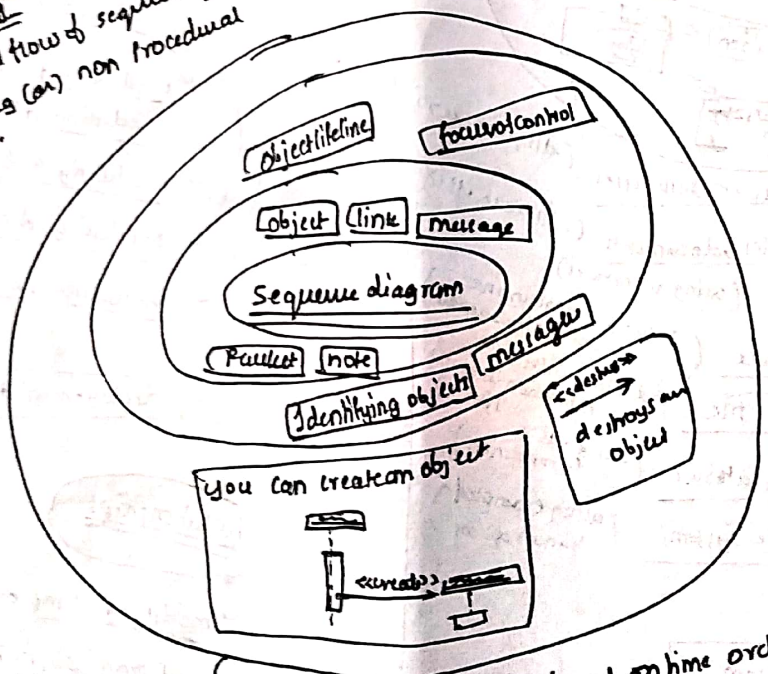
- set context for interaction
- connect relavent paths
- convey the msg()

Collaboration diagram

Sequence numbering

① Procedural (or) nested flow of sequencing

② flat flow sequencing (or) non procedural sequencing.



modelling flows of control based on time ordering of messages

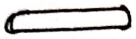
- Set context for interaction
- Connect relavent paths
- Convey the msg()

Sequence diagram

Common modelling technique

- modelling workflow
- modelling object flow

Activity diagram

 **Action** is a named element that is fundamental unit of executable functionality

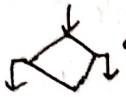
Operation (classname ::) : An action that transmits an operational call (behaviour)

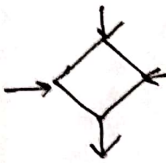
Opaque Action An opaque Action is introduced for implementing specific actions or for use

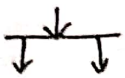
- Initial node [no incoming edges]

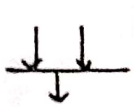
◉ **Final node**: Terminates Activity of Program

⊗ The **flow final node** terminates a flow and destroys all incoming tokens arriving at it. It has no impact on other flows

 **decision** : has one control flow, multiple outgoing control flows

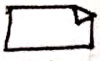
 **Merge** : If one flow even reaches out of many it just combines without waiting for another

 **Fork**

 **join** All flows should satisfy

→ Transition

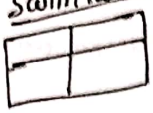
 Passage

 note

 **time event** A time event specifies a point of time by an expression. The expression may be absolute or relative to other points of time

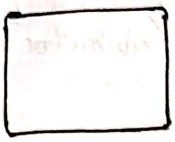
send signal It creates an signal instance from its inputs and transmits it to target object.

Swim lanes : are used to organize responsibility for actions & sub activities according to class.



State diagram

 **State**: A state models a situation during which some (implicit) invariant conditions hold.

 **Composite state** : may contain one or more orthogonal states

 **Orthogonal state** : A Composite state with atleast 2 regions

- Initial
- ◉ Final state
- ○ entry point
- ⊗ exit point

Common modelling technique

- modelling states of an object.

Use Cases ... Terms & concepts

- every usecase must have a unique name that makes it distinguishable
  - called as Simple name
  - Otherwise Pathname (Inside a package)

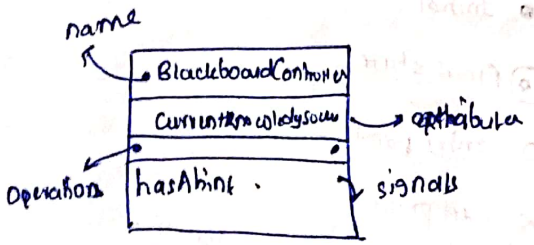
- System is a thing which we are developing
- A subsystem is a grouping of elements of which some constitute a specification of behaviour offered by other contained elements
- View: is a projection of model

System & subsystems

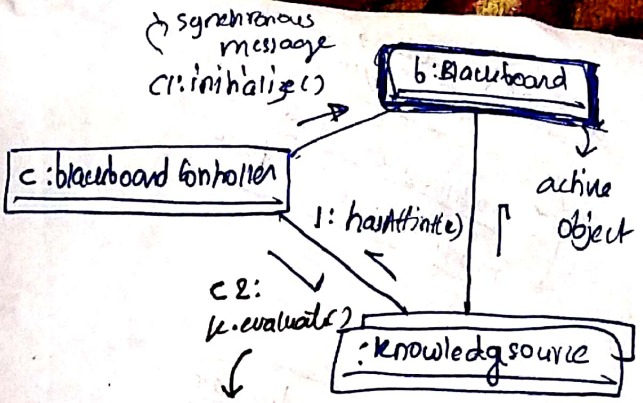
- modelling Architecture of a system
- modelling systems of systems (abstraction).

Process & threads

An Active class object is an object that owns a process or a thread that can initiate control activity.



- There are two standard stereotypes that apply to active classes.
  1. process: (heavyweight flow)
  2. Thread: (lightweight flow)



Asynchronous message

Synchronous message : →

Asynchronous : →

Three types of synchronizations:

1. Sequential
2. gaurded
3. concurrent

Common modelling techniques

1. modelling multiple flows of control
2. modelling interprocess communication

Time & space : (used in time critical system)

- modelling Timing constraints
- modelling distribution of objects.
- modelling objects that migrate