UNIT-III

## Introduction to UML :-

What, then, is a model? Simply put,

A model is a simplification of reality.

→ A model provides the blue prints of a system.

→ Models may encompass detailed plans, as well as more general plans.

Why do we model? There is one fundamental reason.

We build models so that we can better understand the system we are developing.

Through modeling, we achieve four aims.

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 decisions we have made.

"we build models of Complex system because we cannot Comprehend such a system in it's entirely."

## Principles of Modeling:-

→ Choose your model well - the choice of model profoundly impacts the analysis of the problem and the design of the solution.

→ Every model may be expressed at different levels of precision - the same model can be scaled up (or down) to diffrent granularities.

→ The best models are Connected to reality - Simplify the

model, but don't hide the important details.

→ No single model suffices -- Every non-trivial system has diffrent dimensions to the problem and it's solution.

Object oriented modeling:-

Object:- Informally, an object represents an entity, either Physical, Conceptual, or Software.

→ An object is an entity with a well-defined boundary and identify that encapsulates state behavior.

→ A state is a condition or situation during the life of an object, which satisfies some condition.

Basic principles of object orientation:-

The following are the basic principles.

→ Abstraction
→ Encapsulation
→ Modularity
→ Hierarchy
→ poly morphism

Abstraction:- The essential characteristics of an entity that distinguishes it from all other kinds of entities.

Encapsulation:- Hides implementation from clients.

→ clients depend on interface.

Modularity:- Breaks up something complex into manageable pieces.

→ It helps people understand complex systems.

Hierarchy:- Elements at the same level of the heierarchy should be at the same level of abstraction.

Example for a model:-

→ If you want to build a house for your family, you can start with a pile of lumber, some nails, and few basic tools, but it's going to take you a lot longer and your family will certainly be more demanding than the dog.

→ In this case, unless you've already done it a few dozen times before, you'll be better served by doing some detailed planning before you pound the first nail or lay the foundation.

→ At the very least, you'll want to make some sketches of how you want the house to look.

→ If you want to build a quality house that meets the needs of your family and of local building codes, you'll be need to draw some blueprints as well, so that you can think through the intended use of the rooms and the practical details of lighting, heating and plumbing.

* Examples for object Oriented Orientation:-

* Examples for Abstraction:-

→ A Student is a person enrolled in classes in the university.

→ A professor is a person teaching classes at the university.

→ A Course is a class offered by the university.

* Examples for Encapsulation:-

→ For example, professor clark needs to have her maximum course load increased fforom three classes to four classes

Per Semester.

→ Another object makes a request to professor Clark to set the maximum course load to four.

→ The attribute, Maxload, is then changed by the SetMaxload() operation.

## Example for Modularity:-

→ way to manage complexity is to break something that is large and complex into a set of smaller, more manageble pieces.

→ These pieces can then be independently developed as long as their interactions are well understood.

## * Example for Hierarchy:-

→ For example a financial application may have different types of customers and accounts.

→ Hierarchy is neither an organizational chart nora functional decomposition.

**Polymorphism:-**

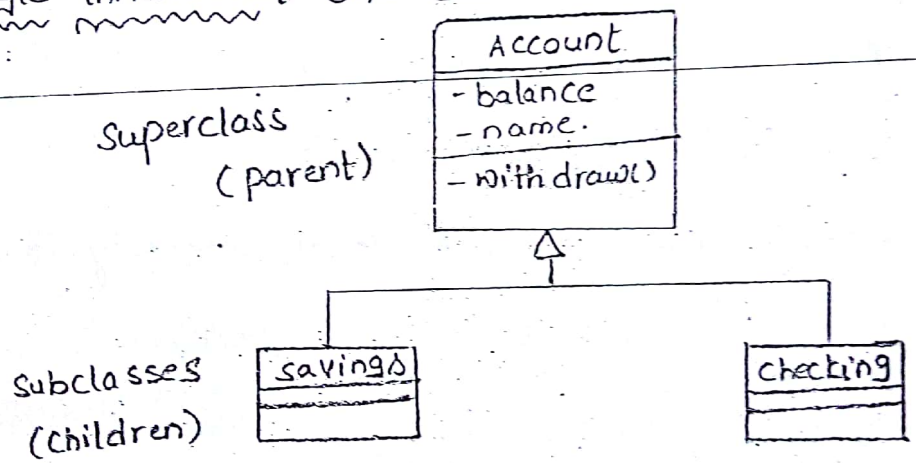
The ability to hide many different implementations behind a single interface.

**Inheritance:-**

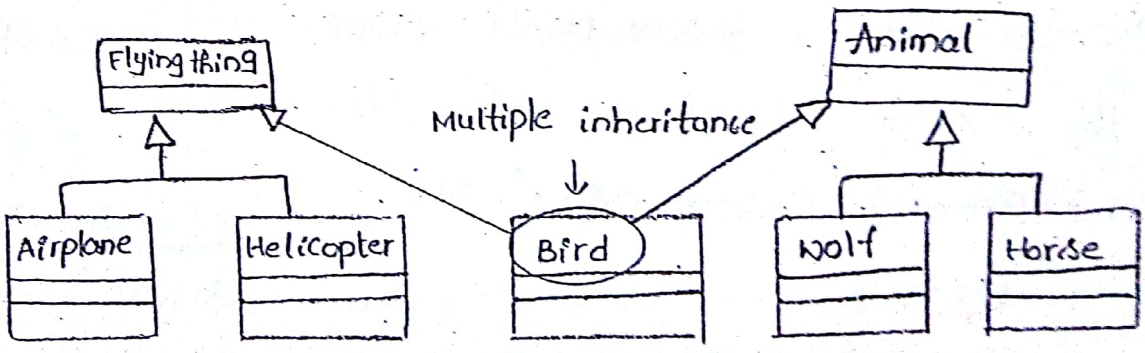
* A Sub class inherits it's parent's attributes operations, and relationships into super class is known as Inheritence.

We have 2 types 1) Single inheritance
2) Multiple inheritance.

* **Single inheritance:-** One class inherits from another.




* **Multiple inheritance:-** A class can inherit from several other classes.




**What is UML?:-**

→ UML - Unified Modeling language

→ UML is a modeling language, not a methodology or process.

→ Fuses the Concepts of the Booch, OMT, OOSE methods.

-> Developed by Grady Booch, James Rumbaugh and Ivar Jacobson at Rational Software.

-> Accepted as a standard by the object Management Group (OMG), in 1997.

Definition:- UML is a modeling language for visualising, specifying, Constructing and documenting the artifacts of Software, -Systems.

=) Visualising :- A picture is worth a thousand words; a graphical notation articulates and unambiguously communicates the overall view of the system (problem-domain).

=) Specifying :- UML provides the means to model precisely, unam -biguously and Completly, the system in question.

=) Constructing :- Models built with UML have a "design" dimension to it; these are language independent and can be implemented in any programming language.

=) Documenting :- Every software project involves a lot of documentation from the inception phase to the deliverables.

Documentation is (among others) for:

-> Requirements

-> Design

-> Tests

-> UML providers the notations for documenting some of these artifacts.

## Conceptual model of UML:-

In UML we have following building blocks.

- → Things
- → Relationships
- → Diagrams.

Things:- In things we have following

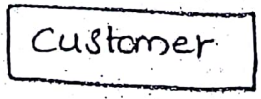
- → Structural
- → Behavioral
- → Grouping
- → Annotational

---

### * Structural Things:-

The nouns of UML models; usually the static parts of the system in question.

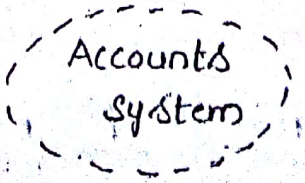
→ Class:- An abstraction of a set of things in the problem domain that have similar properties and/or functionality.

Notation:- 


⇒ Interface:- A Collection of operations that specify the services rendered by a class (or) Component.

Notation:- 

⇒ Collaboration:- A Collection of UML building blocks (classes, interfaces, relationships) that work together to Provide some functionality with in the system.

Notation:- 

=> Use Case :- An abstraction of a set of functions that the system Performs; a use case is "realized" by a Collaboration.

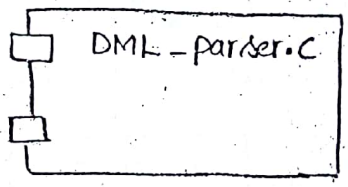
Notation:-


=) Active class :- A class whose instance is an active object; an active object is an object that owns a process (or) thread (units of execution).

Notation :-

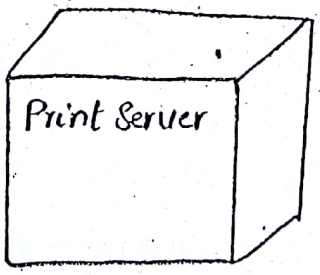

=) Component :- A physical part (typically manifests it self as a piece of software) of the system.

Notation :-




=) Node :- a physical element that exists at run-time and represents a computational resource (typically, hardware resources).

Notation :-




* Behavioral Things :-

The Verbs of UML models; usually the dynamic Parts of the system in question.

=) Interaction :- Some behaviour Constituted by messages exchanged among objects; the exchange of messages is with a view

to achieving some purpose.

Notation:- Parse →

⇒ State machine:- A behaviour that specifies the sequence of states an object goes through, during it's lifetime. A "state" is a condition or situation during the lifetime of an object during which it exhibits certain characteristics and/or performs some function.

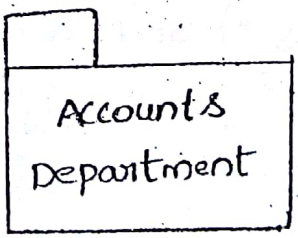
Notation:- Engine Idling

* Grouping Things:-

The organisational part of the UML model; provides a higher level of abstraction (granularity).

⇒ Package:- A general-purpose element that comprises UML-elements - structural, behavioural or even grouping things. Packages are conceptual grouping of the system and need not necessarily be implemented as cohesive software modules.

Notation:-

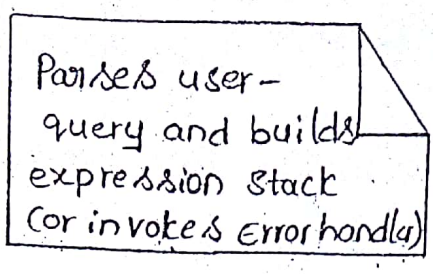



* Annotational Things:-

The explanatory part of the UML model; adds information/meaning to the model elements.

⇒ Note:- A graphical notation for attaching constraints and/or comments to elements of the model.

Notation :-




Relationships :-

Articulates the meaning of the links between things.

⇒ Dependency :- A semantic relationship where a change in one thing (the independent thing) causes a change in the semantics of the other thing (the dependent thing)

Notation :- - - - - - ->

(arrow-head points to the independent thing)

⇒ Association :- A structural relationship that describes the connection between two things.

Notation :- ———

⇒ Generalisation :- A relationship between a general thing (called "Parent" or "superclass") and a more specific kind of that thing (called the "child" or "subclass"), such that the latter can substitute the former.

Notation :- ———▷

⇒ Realization :- A semantic relationship between two things where in one specifies the behaviour to be carried out, and the other carries out the behaviour.

"a collaboration realizes a use case"

Notation :- - - - - - - ▷

(arrow-head points to the thing being realized)

* Diagrams :-

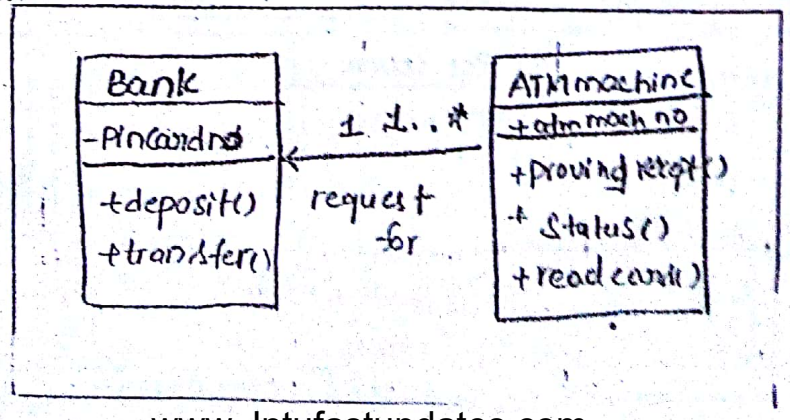
The graphical presentation of the model. Represented as a connected graph - vertices (things) connected by arcs (relationships).

→ UML includes nine diagrams - each capturing a different dimension of a software-system architecture.

- ^ Class Diagram
- ^ Object Diagram
- ^ Use Case Diagram
- ^ Sequence Diagram
- ^ Collaboration Diagram
- ^ State chart Diagram
- ^ Activity Diagram
- ^ Component Diagram
- ^ Deployment Diagram

⇒ Class Diagram :- The most common diagram found in OOAD, shows a set of classes, interfaces, collaborations and their relationships. Model the static view of the system.

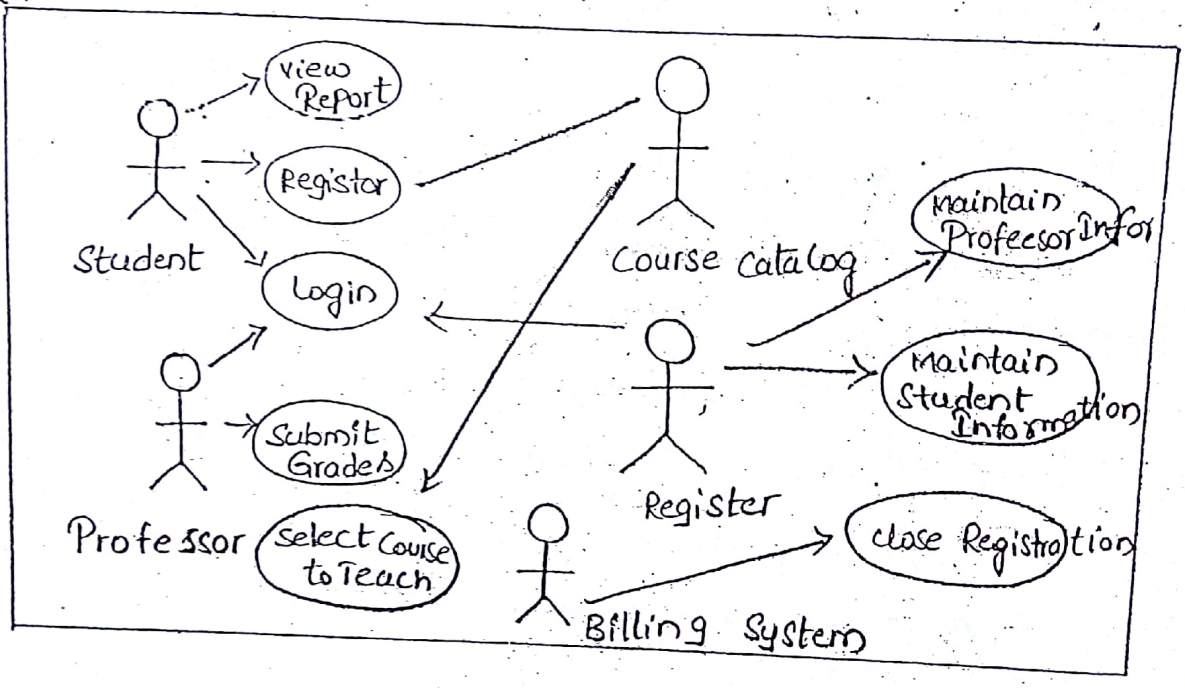
Example :-



⇒ Object Diagram:- A sanpshort of a class diagram; models the instances of things Contained in a class diagram.

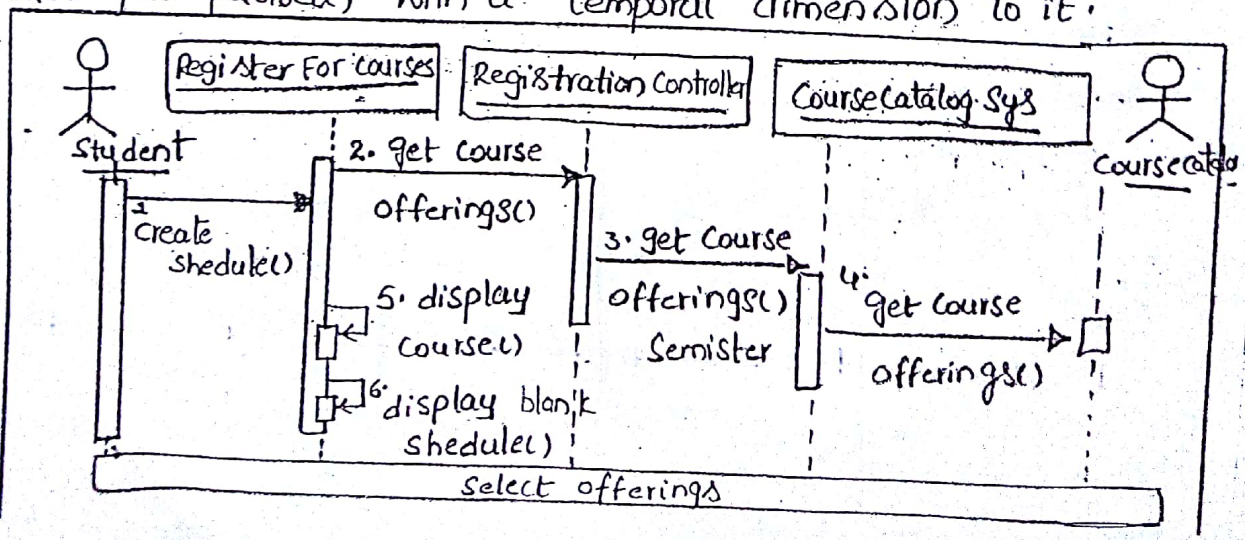
⇒ Use Case Diagram:- Shows a set of "usecases" (sets of functionality performed by the system), the "actors" (typically, People / systems that interact with this system [Problem-domain]) and their relationships, Models WHAT the system is expected to do.

Example:-



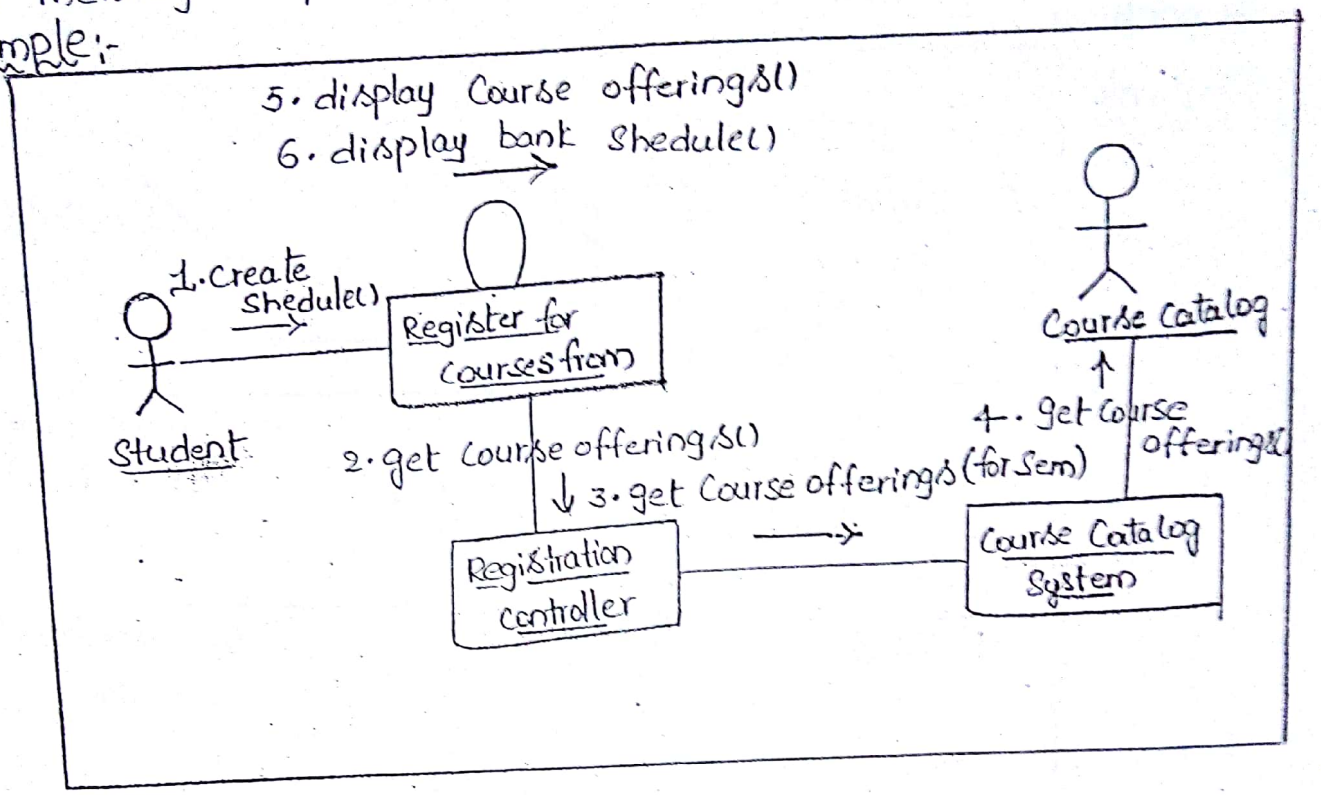

⇒ Sequence Diagram:- models the flow of control by time Ordering; depicts the interaction between various objects by of messages passed, with a temporal dimension to it.

Example:-



Collaboration Diagram :- Models the interaction between objects, with out the temporal dimension; merely depicts the messages passed between objects.

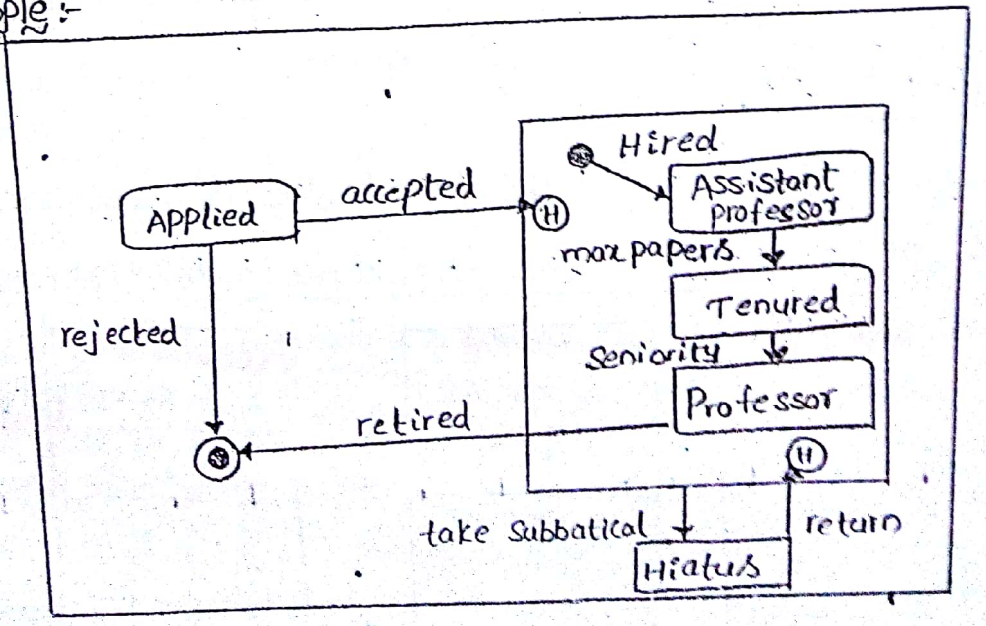
Example :-




⇒ State Chart diagram :-

Shows the diffrente state machines and the events that leads to each of these state machines state - chart diagrams show the flow of control from state to state

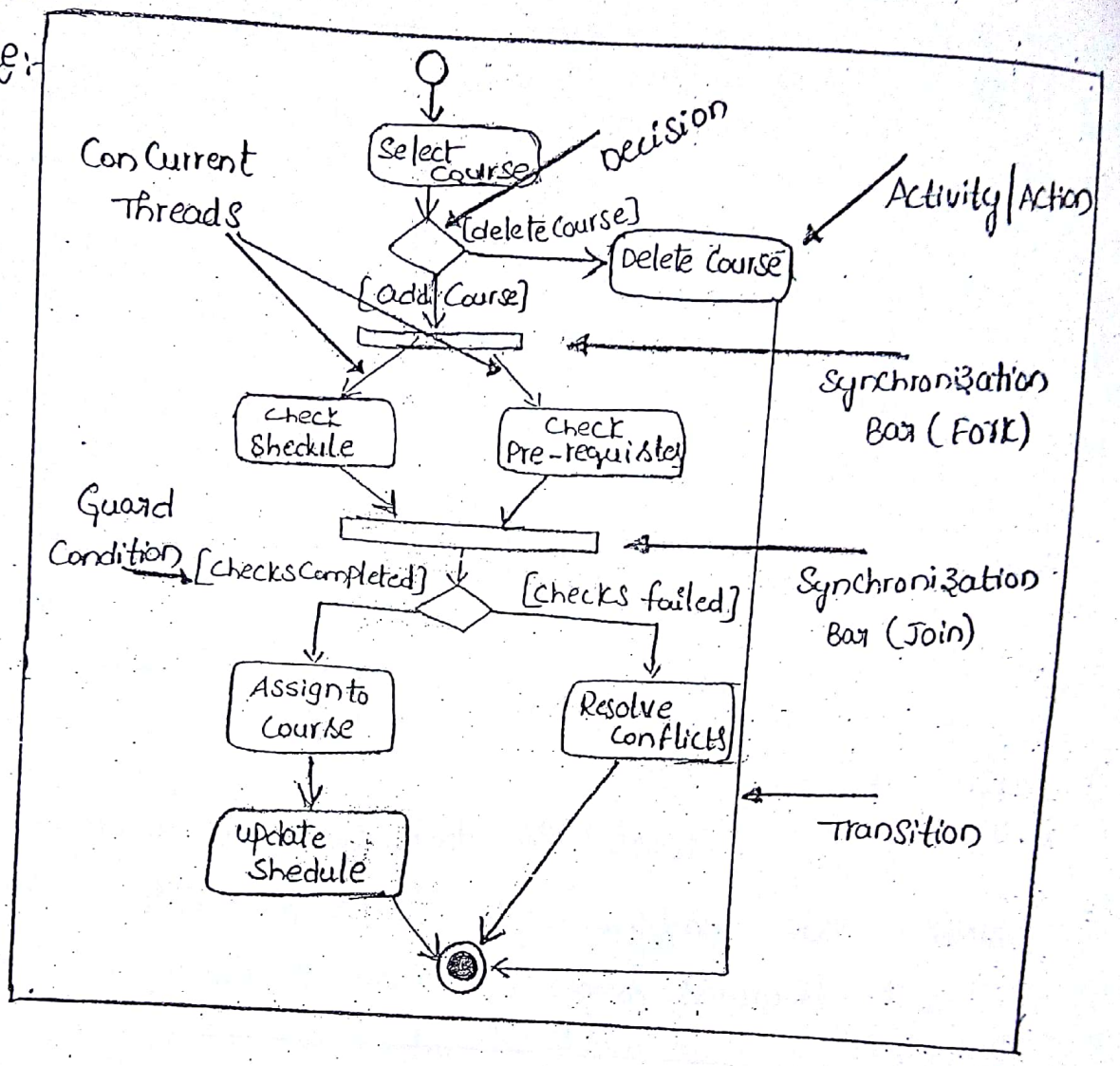
Example :-



⇒ Activity Diagram :-

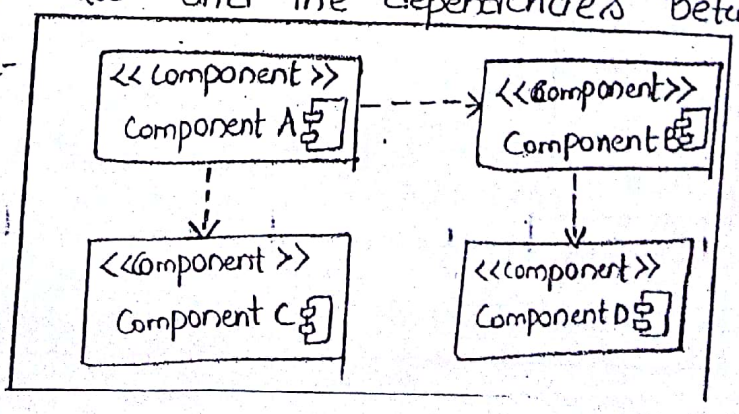
Shows the flow from activity to activity; an "activity" is an ongoing non-atomic execution with in a state-machine.

Example :-



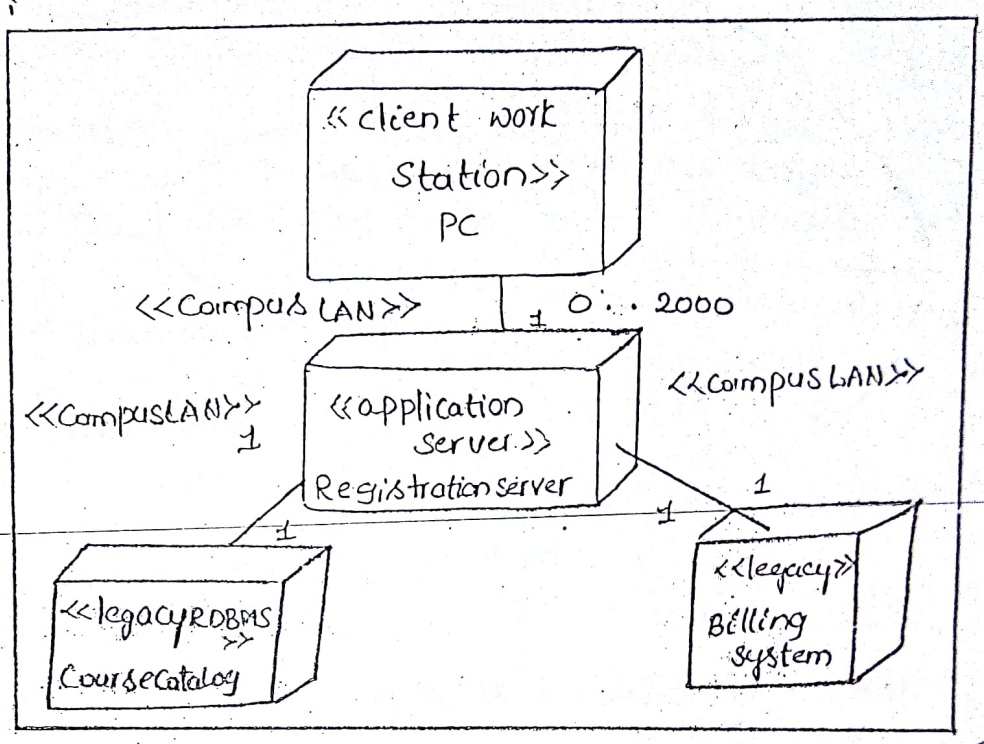

⇒ Component Diagram :-

Shows the physical packaging of software in terms of Components and the dependencies between them. Example :-



Deployment Diagram: Shows the Configuration of processing nodes at run-time and the Components that live on them.

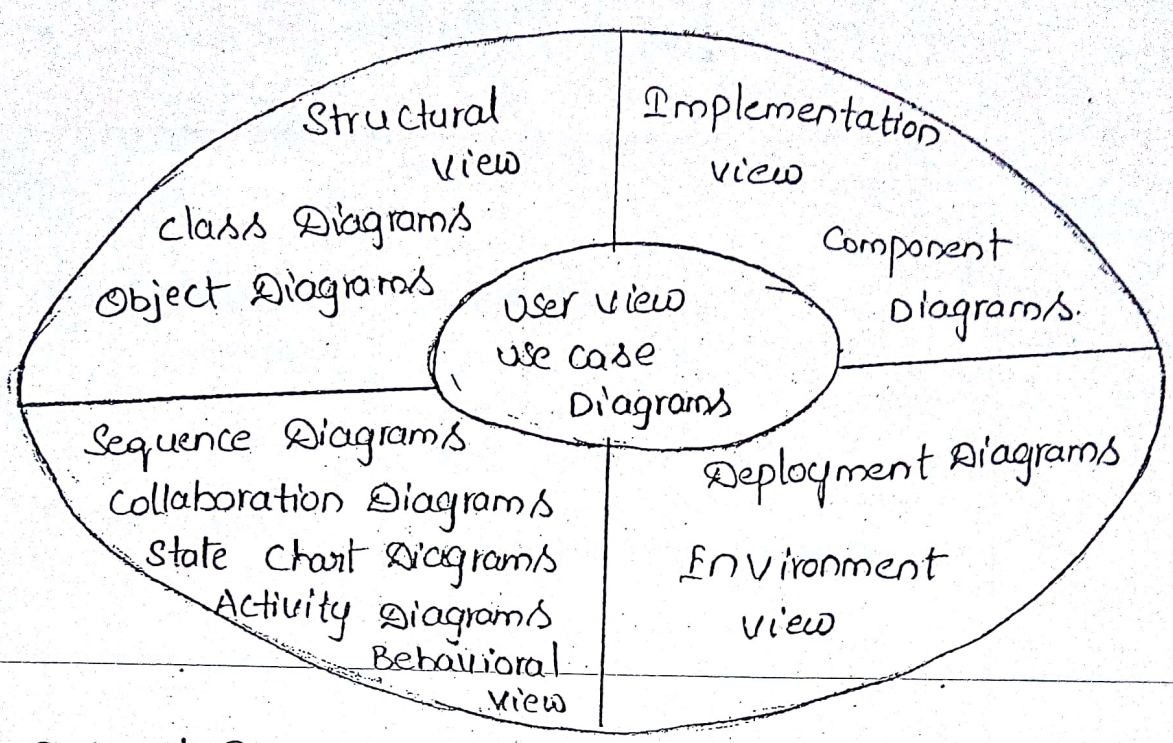
Example:-




ARCHITECTURE:-

Architecture is the set of significant decisions about.

- The organization of a software system
- The selection of the structural elements and their interfaces by which the system is Composed.
- Their behaviour, as specified in the collaborations among those elements.
- The composition of these structural and behavioral elements into progressively larger subsystems.
- The architectural style guides this organization: the static and dynamic elements and their interfaces, their collaborations, and their composition.




* Use Case View :-

^ The use Case view of a system encompasses the use cases that describe the behavior of the system as seen by its end users, analysis, and testers.

→ The view doesn't really specify the organization of a software system.

→ Rather, it exists specify the forces that shape the system's architecture. With the UML, the static aspects of this view are captured in use case diagrams.

→ The dynamic aspects of this view are captured in interaction diagrams, state chart diagrams, and activity diagrams.

* Structural view

→ The design view of a system encompasses the classes, interfaces, and collaborations that form the vocabulary of the problem and it's solution.

→ This view primary supports the functional requirements of the system, meaning the services that the system should Provide to it's end users.

→ with the UML, the static aspects of this view are captured in class diagrams and object diagrams; the dynamic aspects of this view are captured in interaction diagrams, state chart diagrams and activity diagrams.

* Behavioral view:-

→ The process view of a system encompasses the threads and processes that form the system's concurrency and synchronization mechanisms.

→ this view primarily address the performence, scalability, and throughput of the system.

→ With the UML, the static and dynamic aspects of this view are Captured in the same kinds of diagrams as for the design view, but with a focus on the active classes that represent these threads and processes.

* Implementation view:-

→ The implementation view of a system encompasses the components and files that are used to assemble and release the physical system.

→ this view primarily addresses the configuration management of the system's releases, made up of some what independent components and files that can be assembled in various ways to produce a running system.

→ With the UML, the static aspects of this view are captured in interaction diagrams, statechart diagrams, and activity diagrams.

Deployment view:-

→ The deployment view of system encompasses the nodes that form the system's hardware topology on which the system executes.

→ This view primarily addresses the distribution, delivery, and installation of the parts that make up the physical system.

→ With the UML, the static aspects of this view are captured in deployment diagrams; the dynamic aspects of this view are captured in interaction diagrams, and activity diagrams.

⇒ Each of these five views can stand alone so that different stake holders can focus on the issue of the system's architecture that most concern them.

→ These five views also interact with one another - nodes in the deployment view hold components in the implementation view that, in turn, represent the physical realization of classes, interfaces, collaborations, and active classes from the design and process views.

→ The UML permits you to express every one of these five views and their interactions.

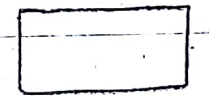
## Deployment View :-

The deployment view of a system encompasses the nodes that form the system's hardware topology on which the system executes.

⇒ Each of these five views can stand alone so that different stake holders can focus on the issues of the system's architecture that most concern them.

## Diagrams in UML :-

class →



Interface →

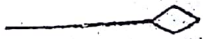
Association →



Directed Association →



Aggregation →

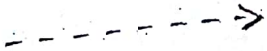


Composition →

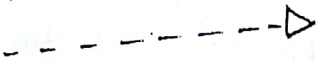
Generalization →



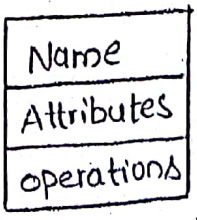
Dependency →



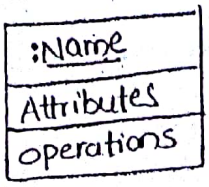
Realization →

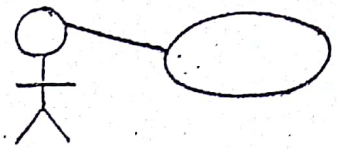


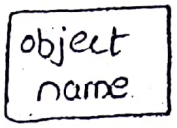
class diagram →



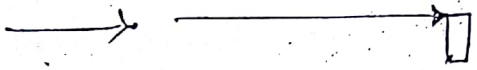

Object diagram →

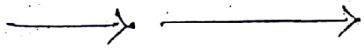


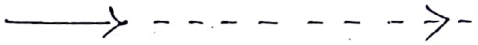
Use Case diagram → 

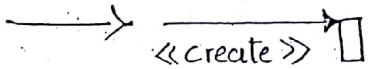
Collaboration diagram → 

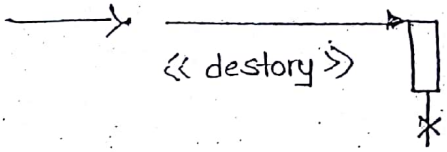

Stimulus → 

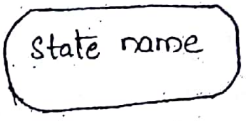
Call → 

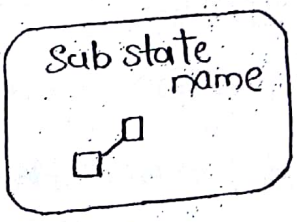
Send → 

Return → 

Create → 


Destory → 


State chart diagram → 


Sub machine state → 

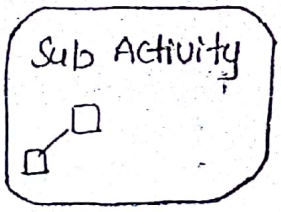

Initial state → 

Final state → 

Junction point → 

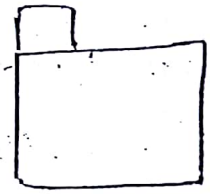
Transition → 

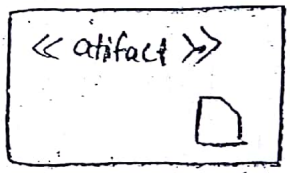
Activity diagram → 

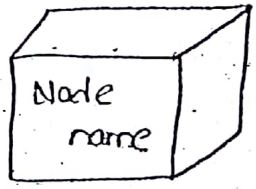
Sub Activity diagram 

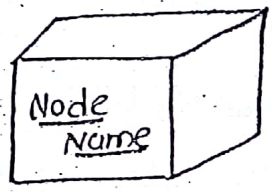

Decision → 

Component diagram → 

Package → 

Artifact → 


Node → 


Node Instance → 


## * RULES OF THE UML :-

→ Like any language, the UML has a number of rules that specify what a well-formed model should look like.

→ A well-formed model is one that is semantically self-consistent and in harmony with all its related models.

→ The UML has semantic rules for.

* Names → what you can call things, relationships, and diagram.
* Scope → the context that gives specific meaning to a name.

* Visibility → How those names can be seen and used by others.
* Integrity → How things properly and consistently relate to one another.
* Execution → What it means to run or simulate a dynamic model.

→ Models built during the development of a software-intensive system tend to evolve and may be viewed by many stakeholders in diffrent ways and at different times.

→ For this reason, it is common for the development team to not only build models that we are well-formed, but also to build models that are.

* Elided → Certain elements are hidden to simplify the view.
* Incomplete → Certain elements may be missing
* Inconsistent → The integrity of the model is not guaranted.

## COMMON MECHANISMS IN THE UML :

→ A building is made simpler and more harmoinous by the conformance to a patern of common feature.

→ The same is true of the UML.

→ It is made simpler by the presence of four common mechanisms that apply consistently throughout the language.

1) Specifications
2) Adornments
3) Common divisons
4) Extensibility mechanisms.

* Specifications:-

→ The UML is more than just a graphical language. Rather behind every part of it's graphical notation there is a specification that providers ~~example~~, a textual statement of the syntax and semantics of that building block.

→ For example, behind a class icon is a specification that provides the full set of attributes, operations (including their full signatures), and behaviours that the class embodies; visually, that class icon might only show a small part of this specification.

→ The UML's diagrams are thus simply visual projections into that back plane, each diagram revealing a specific interesting aspect of the system.

* Adornments:-

→ "Adron" the model - i.e, enhance the model. Adds to the meaning and/or semantics of the element to which it pertains.

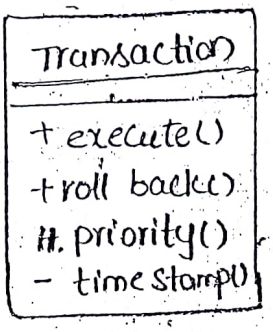
"Notes" are the mechanism provided by UML for adorning a mode:

→ Graphical symbol to render Constrains, Comments, etc.

→ A note that renders only a Comment has no Semantic impact on the element it is adorning; at most adds meaning to it and/or provides guidelines for implementation.

→ For example, adorned to indicate that it is an abstract class with two public, one protected, and one private operation.

→ Every element in the UML is notation starts with a basic symbol, to which can be added a variety of adornments specific to that symbol.




Common Divisions:

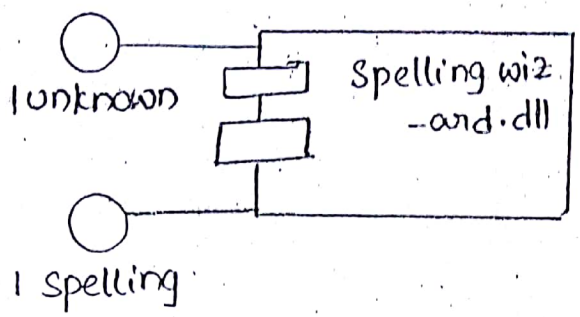
In modeling object-oriented systems, the world often gets divided in at least a couple of ways.

→ First, there is the divison of class and object.

→ A class is an abstraction; an object is one concrete manifestation of that abstraction.

→ In the figure, there is one class, named Customer, together with three objects: Jan (which is marked explicitly as being a Customer object), : Customer (an anonymous Customer object), and Elyse (which in it's specification is marked as being a kind of Customer object, although it is not shown explicitly here.

→ Second, there is the separation of interface and implementation.
→ An interface declares a contract, and an implementation represents one concrete realization of that contract, responsible for faithfully carrying out the interface's complete semantics.



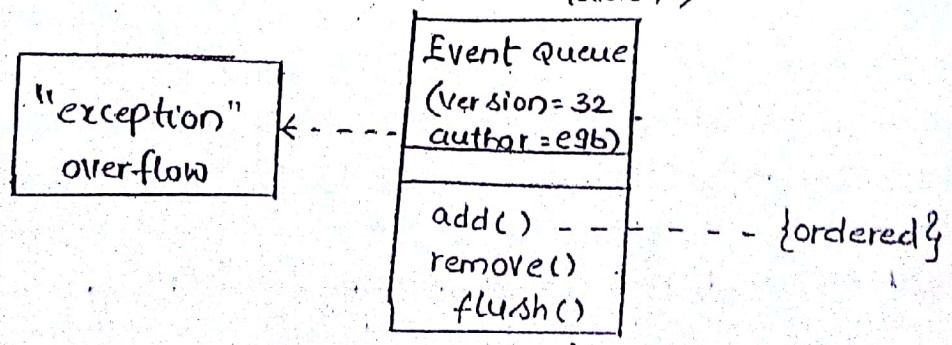

* Extensibility Mechanisms:

In this we have following types.

Stereotypes:

→ used to create new building blocks from existing blocks.
→ New building blocks are domain-specific.
→ A particular abstraction is marked as a "stereotype" and this stereotype is then used at other places in the model to denote the associated abstraction.

Notation: <<meta-class>>



* Tagged values :-

→ used to add to the information of the element (not of its instances)

→ Stereotypes help create new building blocks; tagged values help create new attributes.

→ Commonly used to specify information relevant to code generation, Configuration management, etc

Notation : { version = 1.4 }

* Constraints :-

→ used to create rules for the model

→ Rules that impact the Semantics of the model and specify conditions that must be met.

→ Can apply to any element in the model attributes of a class, relationship, etc.

Notation: { incomplete, disjoint }