02-07-25

1. Introduction to UML

UML:

[Unified Modelling language]: It is a standard visual modelling language to specify visualize, Construct, and documenting the Artifacts of Software Systems.

-> It is a pictorial language used to make Software blueprints.

Model:

It Provides the blueprints of the System.

According to UML, a model may be structured emphasizing the organization of the system or may be behavioural emphasizing the dynamics of the System.

Why do we model:-

-> we built models so that we can better understand the system we are developing.

(or)

-> It is the Central Part of all activities that leadup to the development of good Software.

Importance of Modelling:-

-> It helps us to visualize a System as it is or we want it to be.

-> It Permits us to specify the structure or behaviour of the System.

-> It gives us a template that guide us in Constructing a System.

-> we build models to better understand the System we are building often exposing opportunities for Simplification and reuse.

-> we build models to manage risk.

Principles of Modelling:

1) The choice of what models to create has a profound influence on how a problem is attack and how a solution is shaped.

2) Every model may be express at different levels of precision.

3) The best models are connected to reality.

4) No single model is sufficient, every non-trivial systems is best approached through a small set of nearly independent models.

Applications

→ UML can be used in software intensive systems
→ Enterprise information system
→ Banking and financial services..
→ Telecommunications
→ Transportation
→ Defence or Aerospace
→ Retail
→ Medical Electronics
→ Scientific
→ distributed web Services.

Objective oriented Modeling: [OOM]

→ It is an approach to Software development that uses real-world concepts to represent a system as a collection of interacting object.

→ In Software field they are several approaches for building a model.

The two most common ways.

* Algorithmic Perspective
* object Oriented Perspective.

Algorithmic Perspective:

It is an traditional view of Software development. In this approach the main building blocks of Software is Procedure and function. this view leads the developers to focus on issues of control and decomposition of larger algorithms into smaller ones.

As requirements change and system grows systems built with an algorithmic focus turn out to be very hard to maintain.

Object Oriented perspective:

It acts as a contemporary view of Software development. Here the main building blocks of Software system is Objects or class. It is very easy to maintain & enhance.

Benefits:-

→ Modularity
→ Reusability
→ Scalability
→ easy of maintainance
→ better documentation

Applications of OOM:-

→ Software development
→ database design
→ embedded systems
→ Game development
→ Simulation Systems.

Conceptual Model of UML:-

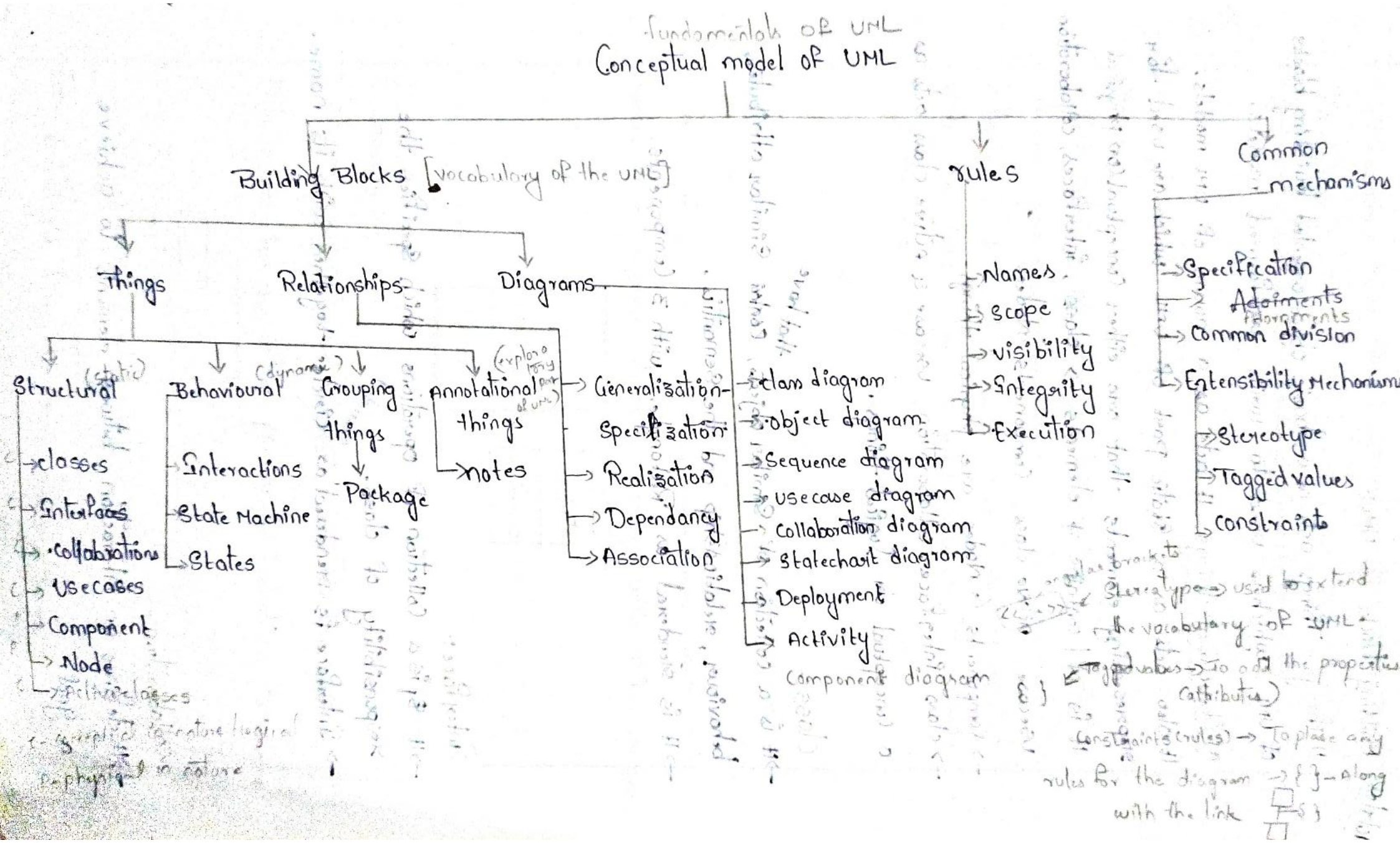
UML:- It is a standard and visual language for describing & modeling software blueprints.

(or)

It is a language for visualizing, specifying, constructing and documenting the artifacts of software systems.

→ It contains the fundamentals of UML

1) Building blocks of UML (syntax / vocabulary)
2) Things / Rules (semantics)
3) Some common mechanisms.


Fundamentals of UML
Conceptual model of UML
Building Blocks [vocabulary of the UML]
Things
Relationships
Diagrams
Rules
Common mechanisms
Structural (static)
Behavioural (dynamic)
Grouping things
Annotational things (explanatory part of UML)
classes
Interfaces
Collaborations
Usecases
Component
Node
Interactions
State Machine
States
Package
Notes
Generalization
Specialization
Realization
Dependancy
Association
class diagram
object diagram
Sequence diagram
usecase diagram
Collaboration diagram
statechart diagram
Deployment
Activity
Component diagram
Names
scope
visibility
Integrity
Execution
Specification
Adornments
Common division
Extensibility Mechanisms
Stereotype
Tagged values
constraints
Stereotype → used to extend the vocabulary of UML
Tagged values → to add the properties (attributes)
Constraints (rules) → To place any rules for the diagram → { } along with the link

10/1/25 Building blocks of UML:-

1) Things:- Things acts as a basic object oriented building blocks of UML which is used to write wellformed models.

a) Structural things:- It acts as a nouns of UML models. It also acts as a static part of UML which are used for representing elements that are either Conceptual (or) physical It is of / contains 7 elements → class, interfaces, collaboration usecase, Active class, Components, Nodes-

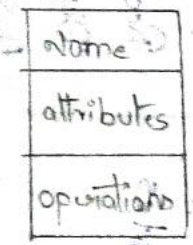
→ Components & Nodes are physical things.

→ class, interface, collaboration, usecase & Active class acts as a conceptual or logical things.

Class:-

→ It is a collection of similar objects, that have contain similar attributes, behaviours, relationship and the semantics.

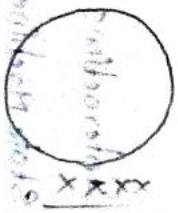
→ It is rendered as Rectangle with 3 Components.



Interface:-

→ It is a collection of operations which specifies the responsibility of class.

→ Interface is rendered as grounded circle together with its name.



Collaboration:-

→ It defines an interaction between elements to achieve a goal.

→ Collaborations have structural as well as behavioral dimensions.

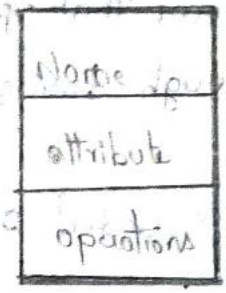
→ It is represented with dashed lines ellipse with its name.

Usecase:-

It is a collection of actions defining the interactions between roles and system. It is used to structure the behavioural things in a model. and it is represented with the solid ellipse with its name written inside or below the ellipse.

Active class:-

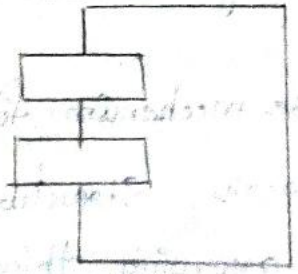
It is rendered just like a class but with heavy lines. That means it is represented as a rectangle with thick borders.



Component:-

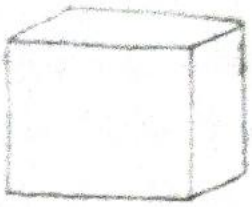
It acts as a physical and replacable part of the system

It is represented as tabed rectangle including its name.



Ex:- Executable files, documents, library pages etc.

Node:-
It acts as a physical element that exist at runtime and represents a computational resource.



Ex- Smartphone, pc's, laptops & embedded systems

b) Behavioural things:-
It acts as a dynamic parts of UML and it is a verb of UML model representing the behaviour overtime and space.

*Interaction:- It is a behavioural that consists of group of messages exchanged among a set of objects to perform a specific task. It is represented as "directed line" with the name of operation. ⟶

*State Machine:- It is a behaviour that specifies the sequence of states and objects goes through during its life time and response to events.
It is represented with the rounded rectangle including its name

c) Grouping things:- It acts as a organisational part of the UML model.

*Package:- It is a general purpose mechanism for organising all elements into groups. That means, Structural things, behavioural things and other grouping things may be placed in a package and it is very purely conceptual

(Logical). It is usually represented with tabbed folder including its name and contents.

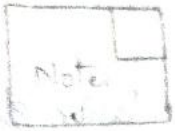
d) Annotational things:-

It acts as explanatory part of UML diagram.

Note:-

It is used to render comments, constraints etc for an UML model.

It is represented with dog-eared corner.



16/7/15 2. Relationships:-

Shows how the elements are associated with eachother and these Association describes the Functionality of an application.

There are 4 types.

→ Generalization- Specialization

→ Association

→ Realization

→ Dependency:

Dependency:-

It is a Semantic relationship between two things in which change in one thing may affect the semantics of other thing. It is represented as dashed lines with a thick arrow head.

- - - - →

Association:-

It describes the connections between two or more things.

→ Aggregation:- It is a special kind of association representing a structural relationship between a whole and its parts. And it is represented as solid line include its name / lable and often contain other adornments (adornaments) such as multiplicity and role names.

Generalization - Specialization:-

Parent class child class

It is also known as is-a relation ship and it describes the inheritance relationship in world of objects. That means, the objects of specialized elements are substitutes for objects of generalized elements. It is represented as solid line with an hallow arrow head pointing to the parent.

Realization:-

It is a semantic relationship in which one class specifies something that another class will perform. It is represented as dashed line with the hallow Arrow head.

Rules of UML:-

→ It contains the rules that specify what a well-formed model should look like.

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

1) Name:- What you can call things, relationships and diagrams.

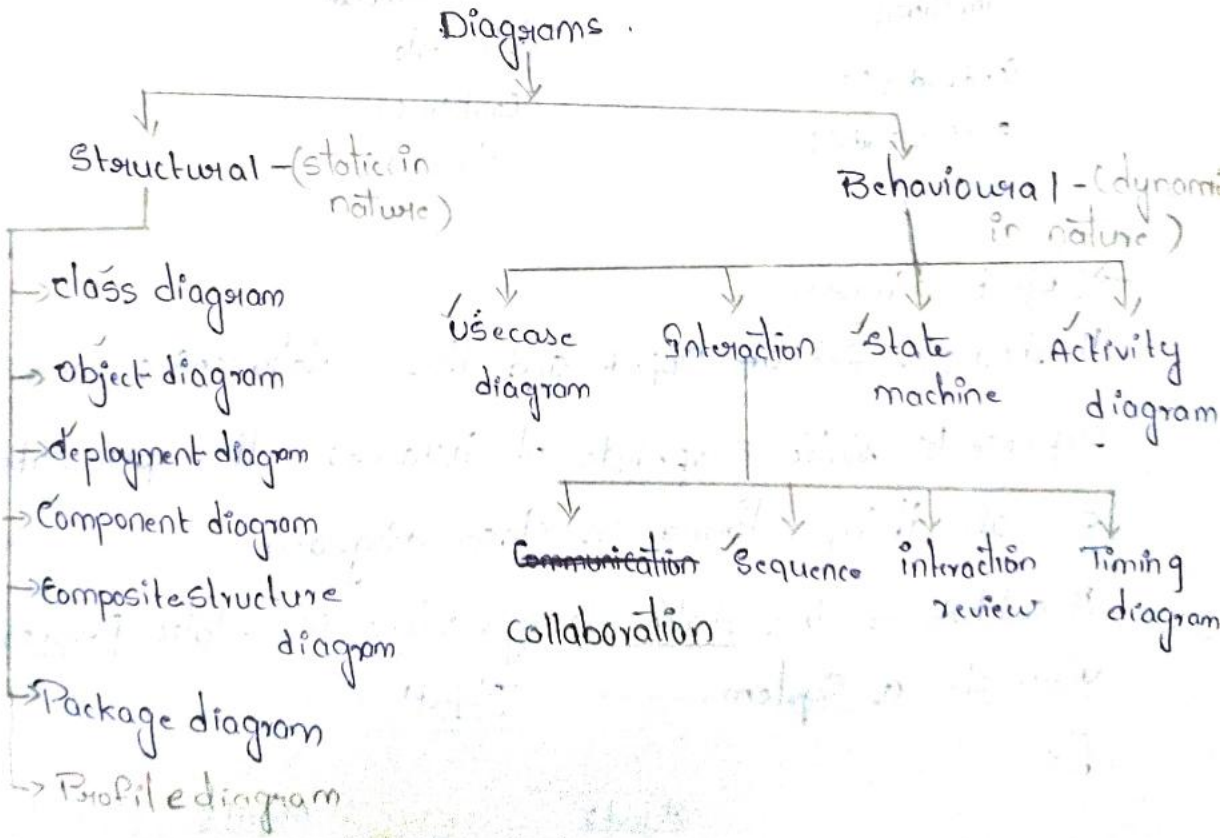
2) Scope:- The context that gives specific meaning to a name.

3) Visibility:- How those names can be seen and used by others.

4) Integrity:- How things properly & consistently relate to one another.

5) Execution:- What it means to run and simulate the dynamic model.

Diagrams:-



Diagrams in UML:-

A diagram is a combination of things and Relationships

→ class diagram:-

→ It is the most commonly used diagram.

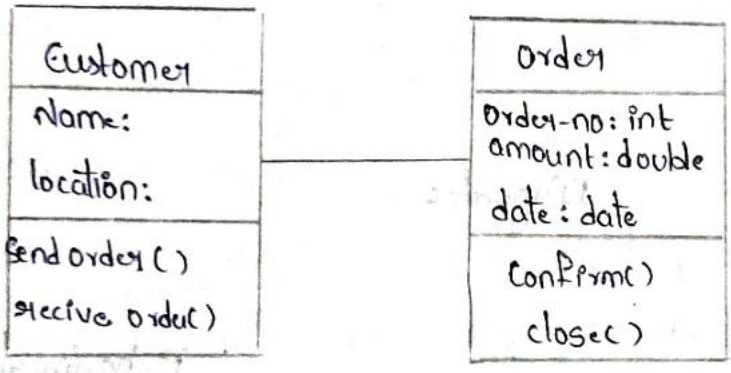
class diagram found in modelling object oriented systems.

→ It shows a set of classes, interfaces and their relationships, collaboration.

→ It addresses the static design view of system.

→ The class diagram that includes active class addresses the static process view of system.

Ex:-

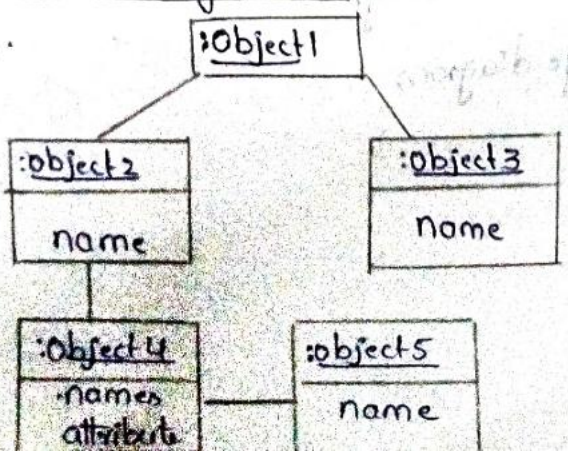


→ Object diagram:-

It shows a set of object and the relationships and it represents static snapshots of instances. [It represents subset of things found in class diagram.

It addresses the static design view or static process view of a system.

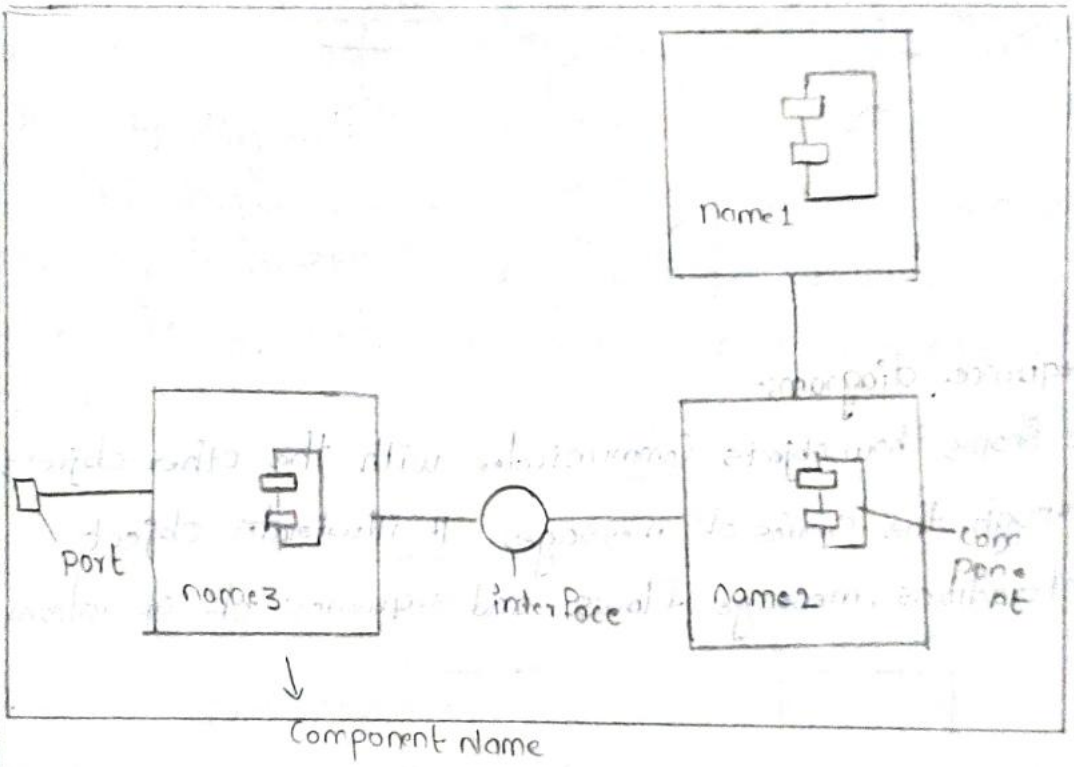
Ex:-



→ Component diagram:-

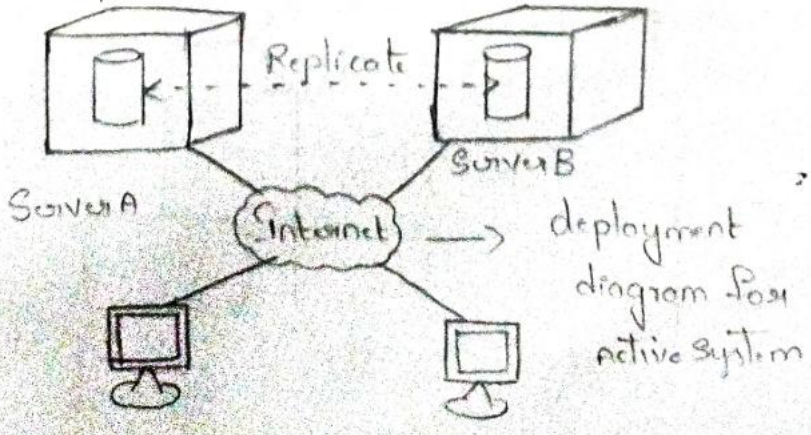
It shows the organizations and dependencies among set of components and it addresses the static implementation view of system. It is related to the class diagram in that a component typically maps to one or more classes, interfaces or collaborations.

→ It shows an encapsulated class and its ports, interfaces and internal structure consists of nested components and connectors.



→ Deployment diagram:-

It shows the configuration of runtime processing nodes and the components that live on than.



Behavioural diagram:-

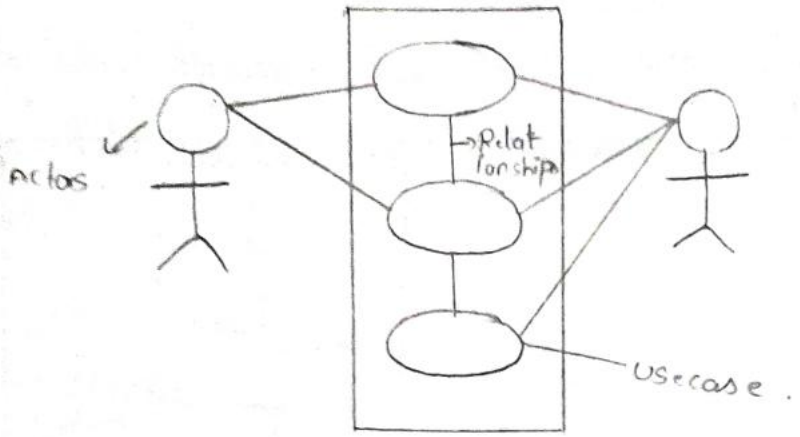
It visualises dynamic system and how they responds to events over time. They shows what happens in a system

→ usecase diagram:-

It shows the interactions between users and system.

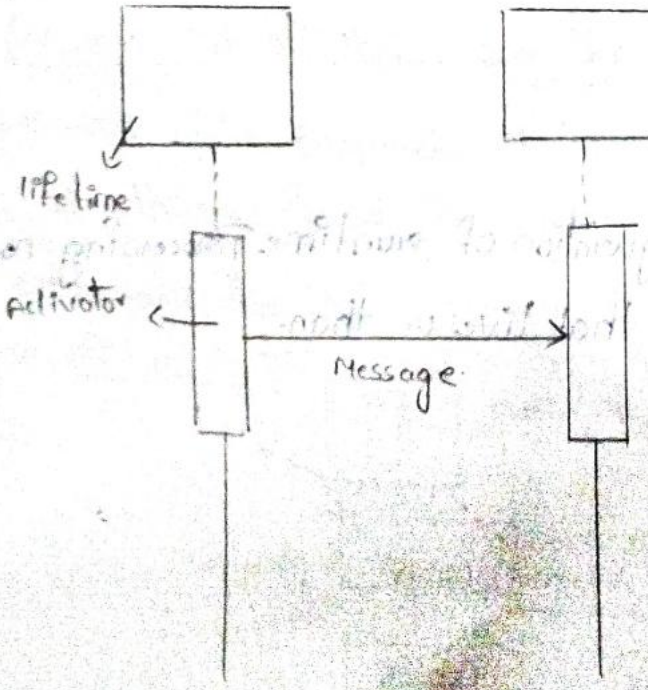
It addresses the static usecase view of a system.

This diagrams are especially important in modelling the behaviour of system.



→ Sequence diagram:-

It shows how objects communicates with the other objects through the series of messages. It illustrates object interactions, message flows and sequence of operations.

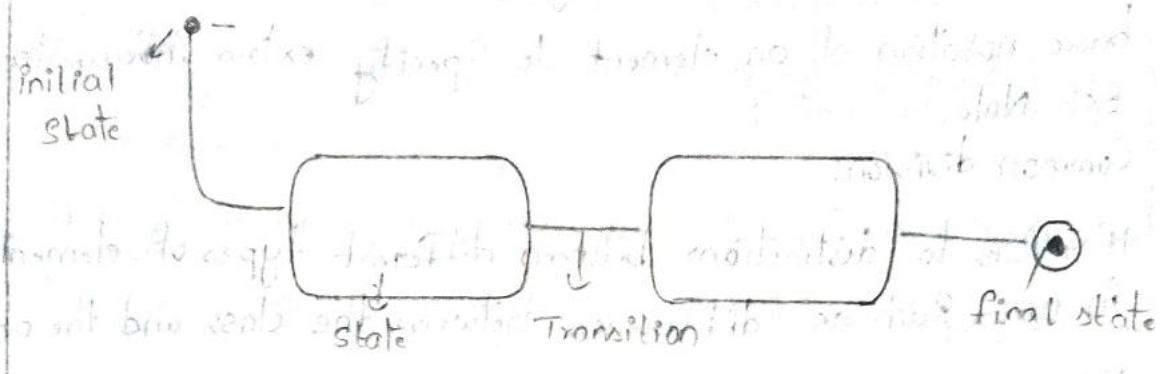


State chart diagram:-

→ It is also known as state machine or state diagram.

→ It is used to represent how a system behaves to different conditions and events.

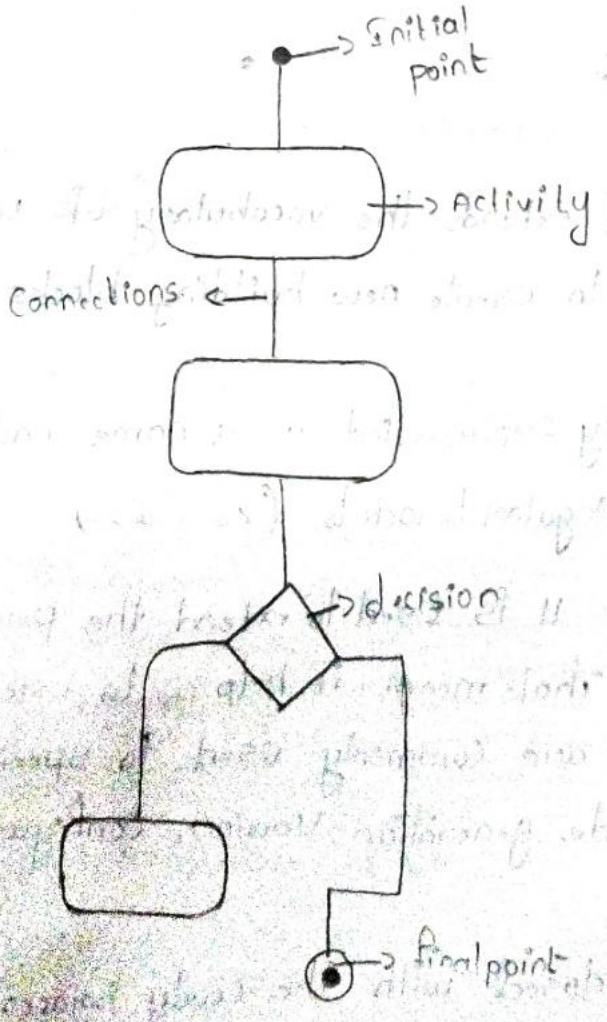
→ It comprises state machines consists of states, transitions, events and activities.



Activity diagram:-

This diagram shows the flow of control from one activity to another activity within a system.

→ The activity starts from initial point and ends at final point.



22/7/25 Common Mechanisms:

Specifications:
It Provides the textual description for graphical elements used in UML diagrams.

Adornments:
These are the textual & graphical items added to the basic notation of an element to specify extra information.
Ex:- Note [comments]

Common division:
It refers to distinctions between different types of elements in UML such as difference between the class and the object.

Extensibility Mechanisms:
Sometimes user might need to represent information through notations which are not available in UML. at that time we can use Extensibility Mechanisms.
-> Stereotypes
-> Tagged values &
-> Constraints.

1) Stereotypes:- It extends the vocabulary of UML that means these are used to create new building blocks from the existing blocks.
It is graphically represented as a name enclosed by guillimets (or) Angular brackets (<< >>)

2) Tagged values: It is used to extend the properties of UML building blocks. That means, it help us to create new attributes. These are commonly used to specify information relevant to code generation, version, configuration, ~~Autosave~~ Author etc.
-> These are enclosed with the curly braces and are

·written under the element name· -> { }.

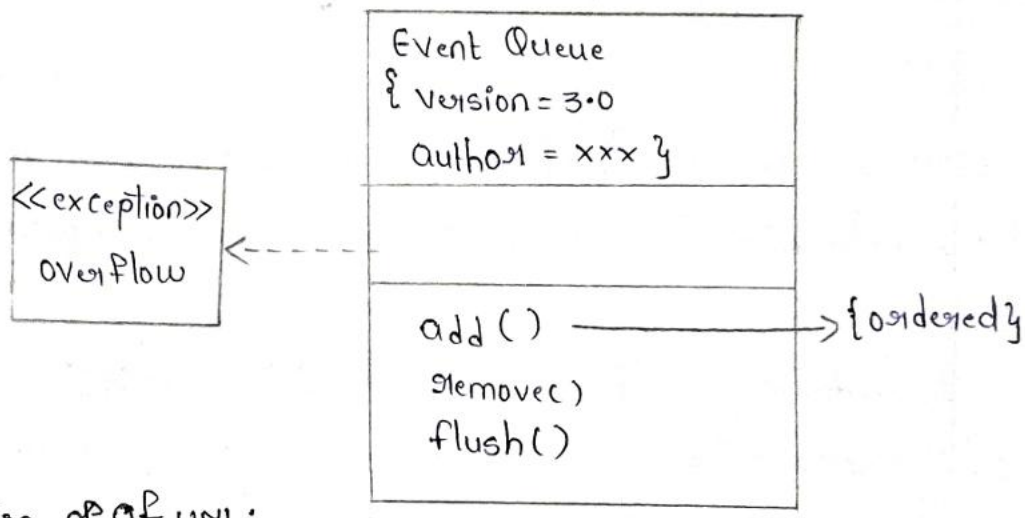
Constraints:-

These are used to extend the Semantics of UML. That means it is used to create new rules or modify the existing ones.

-> The constraints can be applied to any element in the model that is class, attributes, relationships etc.

-> Graphically it is represented as a string enclosed by curly braces. -> { }.

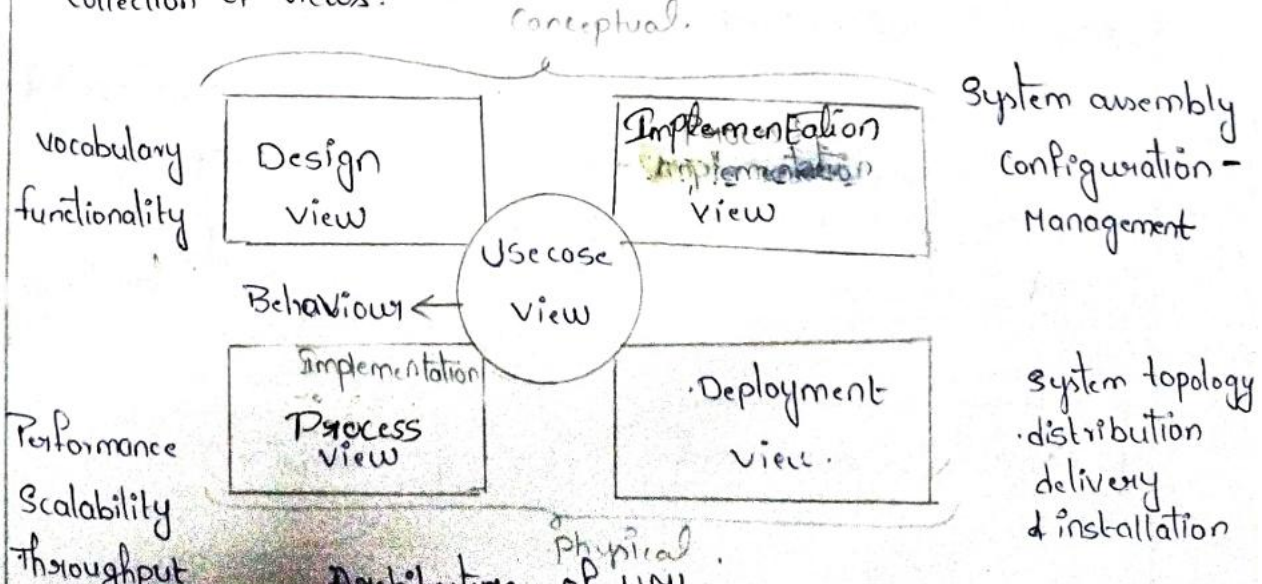
-> It is placed near the associated elements are connected to that elements by dependency relationships.



23/1/25 Architecture of UML:

↓

Collection of views.

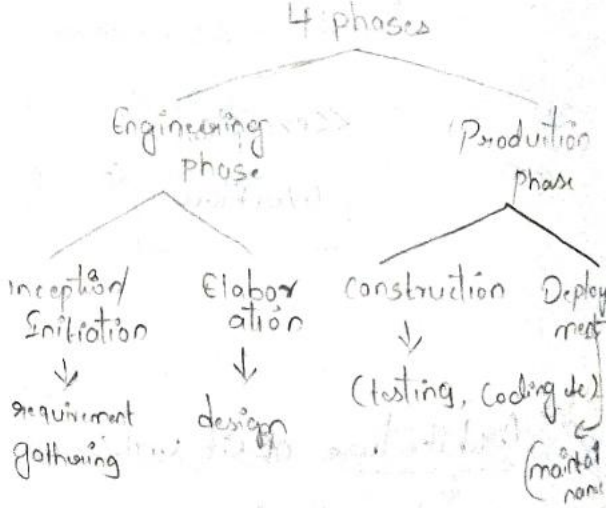


Architecture of UML

View	Stakeholder	Static Aspects	Dynamic Aspects.
Usecase (defines the functionality of uml)	endusers Analysts testers	usecase diagrams	Interaction statechart Activity
Design (defines the structure)	Endusers	class & object	ISA (→ Interaction, → State chart, → Activity)
Process (defines the flows of execution)	Programmers Configuration Managers	Class (Active Class)	ISA
Implementation (assembles all the physical components into optimized thing)	Integrators	Component	ISA
Deployment (redus)	System Engineer	Deployment	ISA

→ Usecase driven
→ Architecture centric
→ Iteration & Incremental

24/4/12 Software Development Life Cycle (SDLC):-



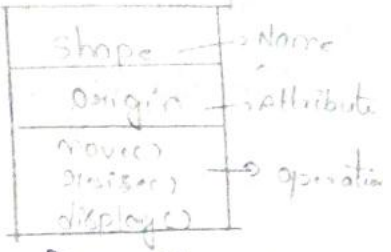
Unit-II

25/7/25

Basic Structural Modeling:-

Classes:-

Class is a description of set of objects that shares the same attributes, operations, relationships & the semantics (rules).



→ Names:

A class name must be unique, that means every class has a name that distinguishes it from the other classes.

→ A name is a textual string that name alone is known as Simple Name.

→ The path name is the class name prefixed by the name of the package in which that class lives. (mentioned with :)

Rule:-

→ A class name should be text consist of any number of letters and numbers and certain punctuation marks (!, :, ;)

→ you can capitalize the first letter of everyword in a class name as in Customer or Temperature Sensors.

Examples:-

Simple Name:-

Temperature Sensors

Customer

Wall

Path Name:-

Business Rules: fraud-Agent

Java::awt::Rectangle.

→ Attributes :- (implements the structure of the class)

→ An Attribute is a named property of a class that describes a range of values that the instance of Property may hold.

→ A class may have any number of attributes or no attributes at all.

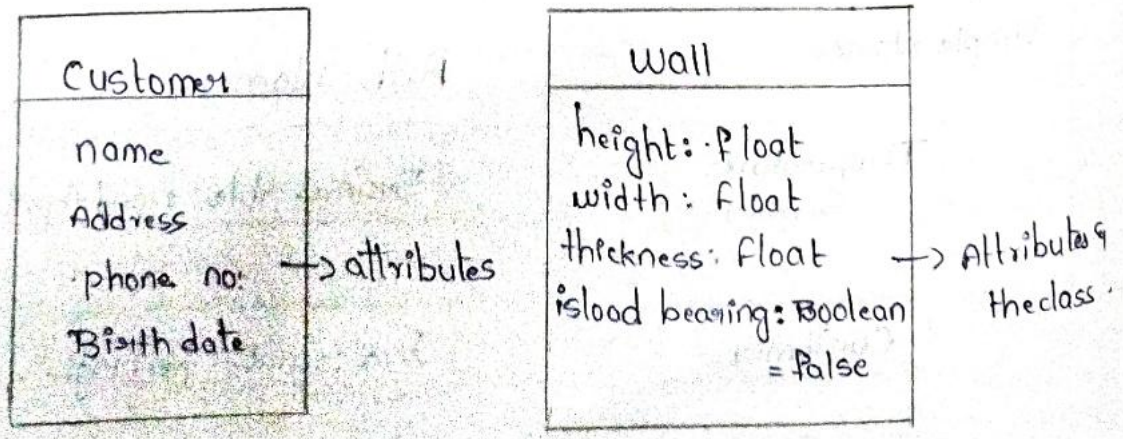
→ Graphically the attributes are listed in below the compartment just below the class name.

→ An attribute name may a yestext just like a classname.

→ you can capitalise the first letter of every word in an attribute name except the first letter as in name or load ~~vary~~ bearing. (Varying).

→ you can also Specify other features of an attribute such as marking it read only or shared by all objects of the class.

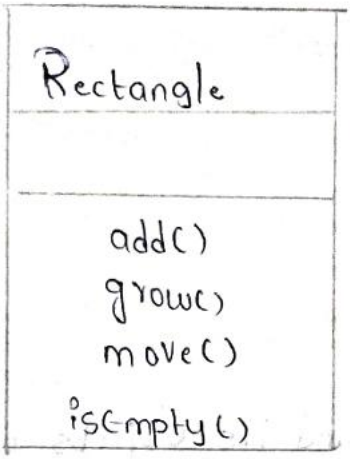
→ you can also Specify an attribute by stating its property and possibly a default initial value.

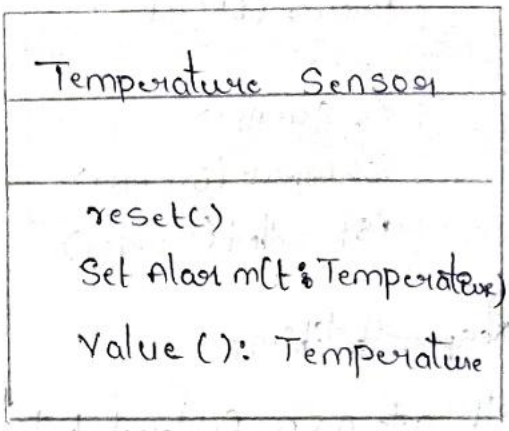


Operations:- [implements the behaviours of the class].

It is the implementation of a services that can be requested from any object of the class to affect behaviour.

→Graphically operations are listed in a compartment just below the class attributes.



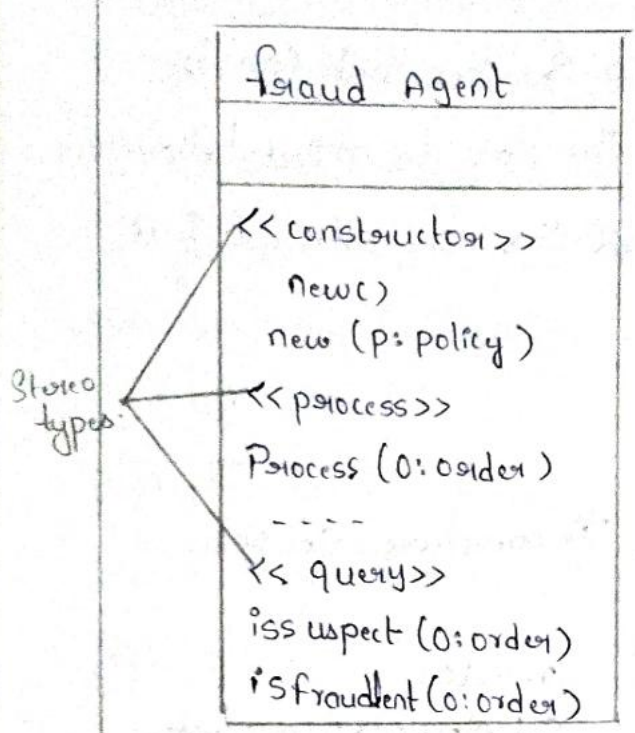


Organizing attributes & operations:-

→When drawing a class, we don't have to show every attribute & every operation at once. That means, you can choose to show only some or none of the class' attributes & operations.

→ An empty compartment doesnot necessarily means there are no attributes or operations just that you didn't choose to show them.

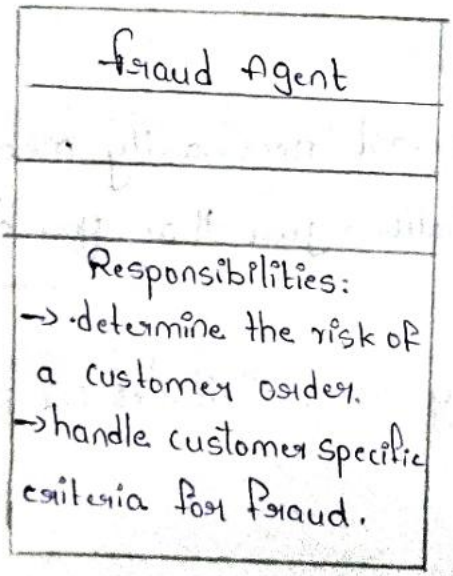
→You can explicitly specify more attributes or properties with an stereotype.



Responsibilities:-

→ It acts as a Contract between class and its objects. The responsibilities of a class represents its features & roles.

→ We can specify responsibilities as a separate compartment at the bottom of the class icon.



CRC - class responsibility collaborative tool

Common-modeling Techniques:-

1) Modelling the vocabulary of a System:

→ Identify those things that users or implementors use to describe the problem or solution use CRC cards

and use case based analysis to help use find this abstractions
→ for each abstraction identify a set of responsibilities
make sure that each class is crisply define and that there is good balance of responsibilities among all your classes.
→ It provides the attributes & operations that are needed to carry out these responsibilities for each class.

2) Modelling the distribution of responsibilities in a system:
→ Identify a set of classes that work together closley to carry out some behaviour.
→ Identify a set of responsibilities for each of this classes.
→ Look at this set of classes as a whole. Split classes that have too many responsibilities into smaller abstractions, collabs tiny classes that have trival responsibilities into larger ones & reallocate responsibilities so that each abstraction reasonably stand on its own.
→ Consider the ways in which those classes collaborate with one another and redistribute their responsibilities accordingly so that no class within a collaboration does too much or too little.

30/1/25
3) Modelling non-Software things:-
→ Model the thing your for abstracting as a class.
→ If you want to distinguish these things from the UML define, building blocks, create a new building blocks by using a new stereotypes to specify these new semantics and to give a distinct visual one.

→ If the thing you are modelling is some kind of hardware that it self contain software Consider modelling it is a kind of knode as well so that you can further expand on its structure:

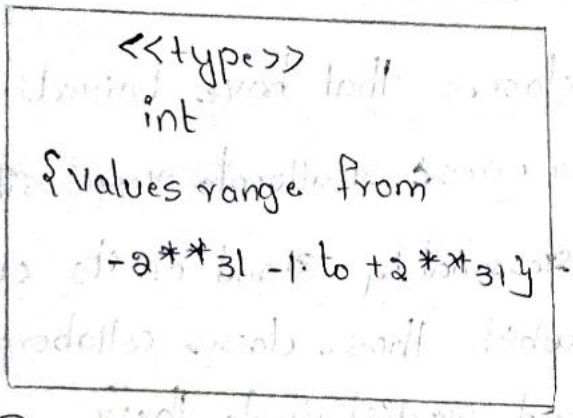
4) Modelling Primitive types:-

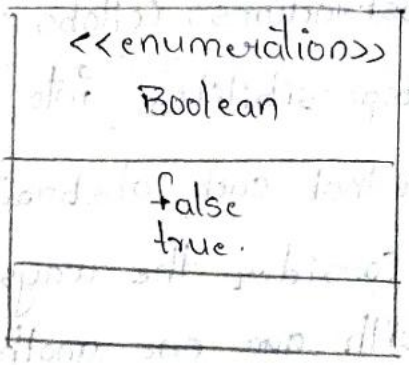
→ To model the primitive types

Model the thing you are abstracting as a type or an enumaration which is rendered using class notation with the appropriate stereotype.

→ If you need to specify the range of values associated with its types, use constraints

Example:-





Relationships:-

The things can Connect to one another either logically or physically or modelled as relationships.

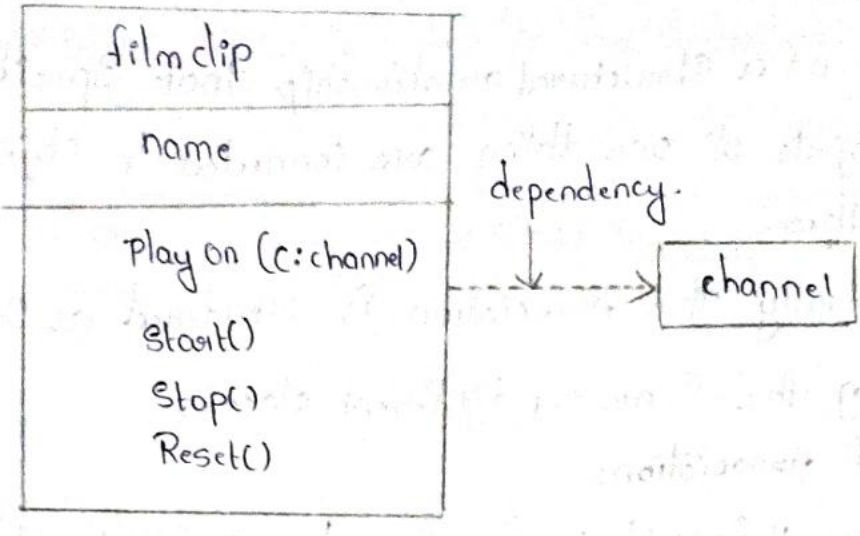
Dependency:

It acts as a using relationship that states that change in Specification of one thing may effect another thing that uses it.

Generally the dependency is rendered as dashed directed line

- - - - - ->

Ex:-



Generalization

acts as a kind of.
It is-a relationship

It acts as a is-a kind of relationship.

-> that means it is a relationship between general thing (Superclass or Parent class) and a more specific thing. [that is called as child class or subclass].

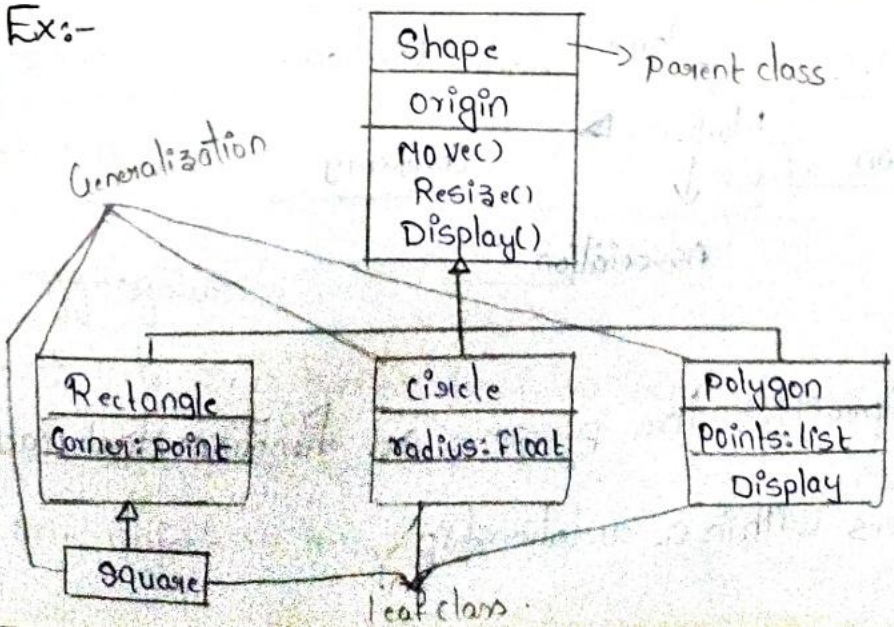
-> Generally Generalization is represented as a solid directed line with large open arrow head pointing to the parent.

-> If class has only one Parent, Such inheritance is called Single inheritance.

-> If class has one or more parents, Such inheritance is known as Multiple Inheritance.

-> A class that has no children is called leaf class.

Ex:-



31/1/25

Association:-

It acts as a structural relationship that specifies that objects of one thing are connected to objects of another.

It Generally the Association is rendered as Solid line Connecting the Same or different classes.

Types of Association:-

-> Unary - It is reflexive in nature, here Single class having a
-> Binary relationship with itself.
-> Ternary.

Binary:-

When two classes are linked. that is called binary Association

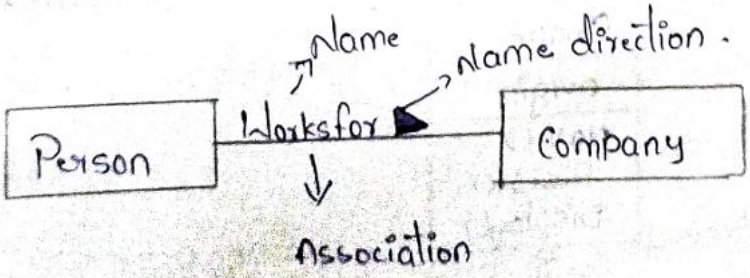
Ternary Association:-

It involves three classes that are used to represent more complex relationships where three entities are interconnected.

⇒ Name:-

It is used to describe the nature of relationship.

Ex:-

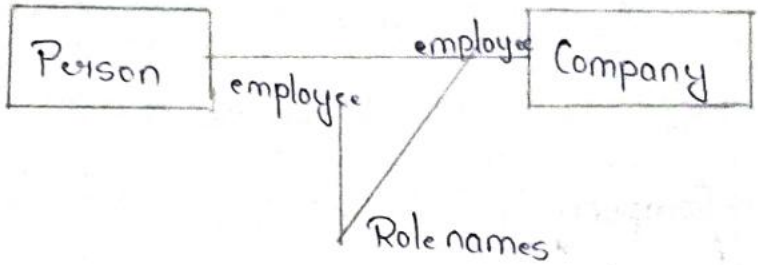


⇒ role:

It Specifies the purpose or function that each class Serves within a relationship

⇒ Role names are typically placed near the end of the association lines connected to the respected classes.

Ex:-



⇒ Multiplicity:-

It defines the number of instances of one class that can be associated with single instance of another class.

→ How many is called the multiplicity of an association role.

→ When you state a multiplicity at one end of an association you are specifying that for each object of the class at the opposite end there must be that many objects at the near end.

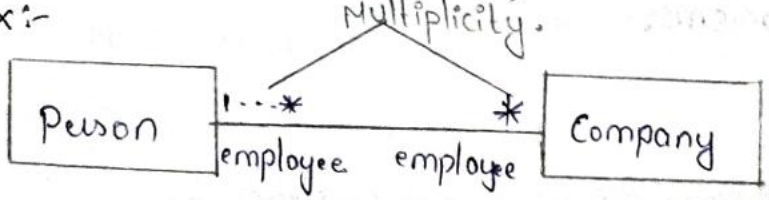
→ you can show multiplicity of exactly one (1) and

Zero (or) one (0...1)

many (0..*)

one or more (1...*)

Ex:-



Aggregation:-

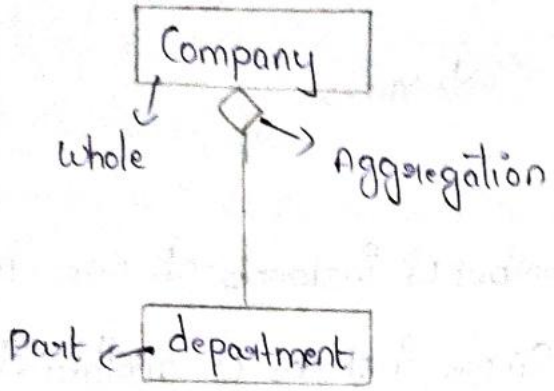
→ It acts as a has-a relationship between classes.

→ It refers to the relationships between objects where one object is known as Whole is composed of

one or more other objects is referred to as parts.

-> It is represented with the hallow diamond shape at the aggregate end of the line that connect to its Part.

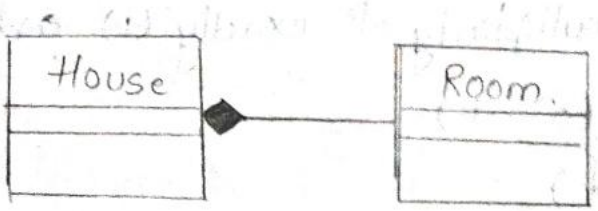
Ex:-



Composition:-

It represents a Strong form of has-a or Whole/part relationship characterised by a strong ownership. This means the parts are created when the whole is created and destroyed when the whole is destroyed.

Ex:-



Common Mechanisms:-

Stereotypes:-

It extends the vocabulary of UML So that we can create new model elements derived from the existing ones. It is represented as gullimets & placed above the name of the another element.

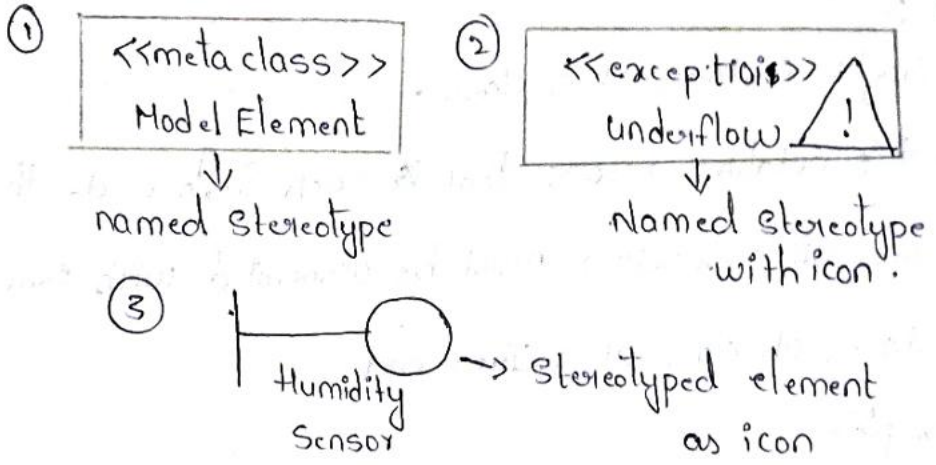
We are using three approaches for representing the Stereotype.

⇒ In 1st approach, We can represent the name of the Stereotype above the element name.

⇒ We can also represent Stereotype by using icon. That icon is placed at the right of the name.

⇒ 3rd approach:- We can use that icon as a basic Symbol for the Stereotyped item.

Ex:-



11/8/25

Tagged values:-

→ Defines the Properties for elements.

→ It allows us to extend the properties of UML building block so that we can create new information in the specification of that element.

→ It is not equal to the class attribute. and it is represented as a string enclosed by brackets and placed below the name of another element. This string includes a name {Tag} a Separator (=) and the value of tag.

you can also Specify only its value.

Multiplicity:-

It acts as a cardinality between two classes. It describes how many instances of one class can be connected to instances of another class through a given association.

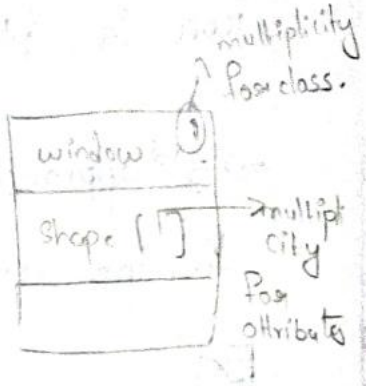
Types of Multiplicity:-

⇒ 1 —> Means exactly only one instance.

⇒ 0....1 Means zero or one instance.

⇒ 0....* Means zero or many instance.

⇒ 1....1 one or more instances.

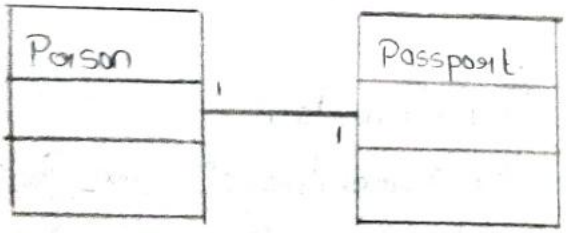


1 :-

-> single instance of a class.

-> This notation means that each instance of the class at one of association must be associated with exactly one instance of class at another end.

Example:-



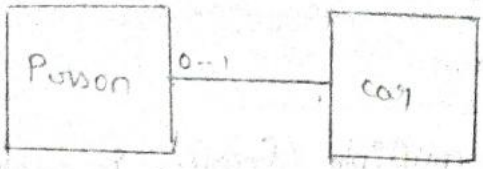
0...1 :-

-> Zero to one instance of class.

-> It is also known as optional relationship. Here instance of class at one end of association can be associated with either no instance or one instance instance of class at other end.

Ex:-

A Person may or may not own a car



-> 0...* :-

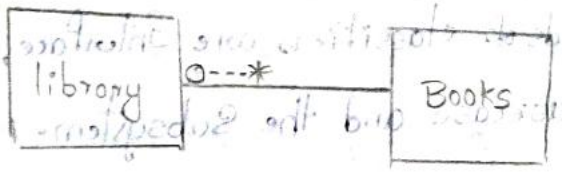
-> Zero or more instances of class.

This notation signifies that instance of class at one end of association can be associated with zero or one or many instances of class at other end.

->It denotes a flexible relationship that can vary in number of instances.

Ex:-

A library can have zero or more books.



1...1 :-

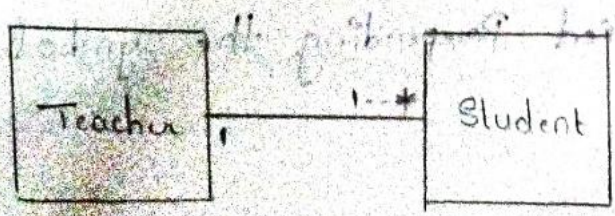
-> one or many instances of class.

->This indicates that instance of class at one end of the association must be associated with atleast one instance of class at another end, but can be associated with many.

->It ensures a mandatory relationship with no upper limit on number of instances.

Ex:-

Teacher can teach multiple students.



---:-

-> Many to many instance of class.

-> The student can enroll in multiple Courses & each course have multiple students.

5/8/25

Part-II Advanced Structural Modeling:-

⟹ **Advanced classes:-**

classifiers:-

It is a general term that describes structural & behavioural features. The most kind of classifiers in UML is class. The different kinds of classifiers are Interface, datatypes, signal, component, node, usecase and the subsystem.

6/8/25

Visibility:-

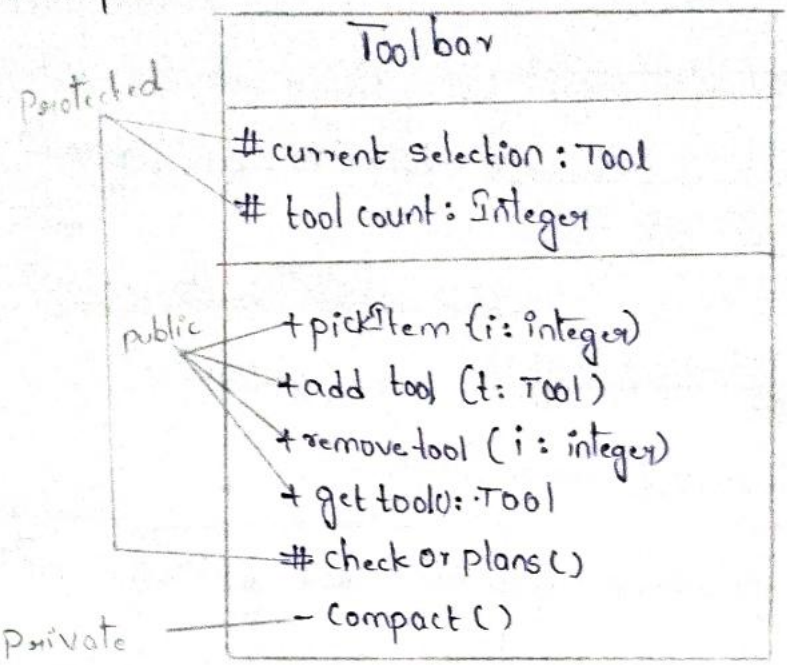
The visibility of a feature specifies whether it can be used or accessed by other classifiers.

(i) Public :- Any outside classifier with visibility to the given classifier can use the feature specified the by prepending the symbol '+'

(ii) Private :- only the classifier itself can use the features specified by prepending the symbol '-'

(iii) Protected :- Any decendent of the classifier can use the feature specified prepending the symbol '#'.

Example:-



6/8/25 Abstract class:-

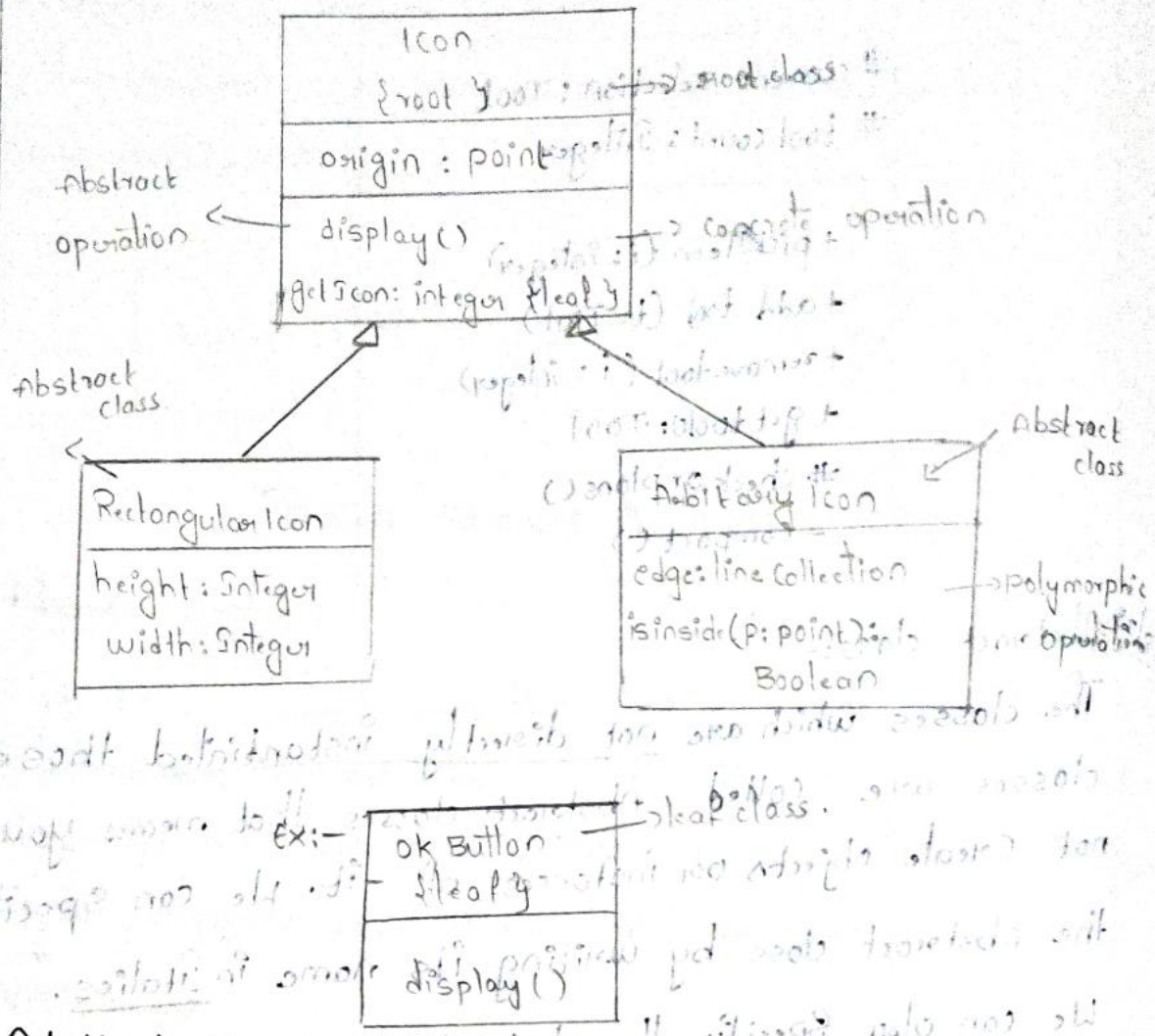
The classes which are not directly instantiated those classes are called Abstract classes that means you can not create objects or instances of it. We can specify the abstract class by writing its name in Italics. We can also specify the abstract operation by writing its name in Italics.

Leaf class & Root class:-

* A class which has no child class is known as leaf class. Such a class can be represented by writing leaf as a property under the class name.

* A class with no parents is known as root class. Such a class is represented by writing root as a property under class name.

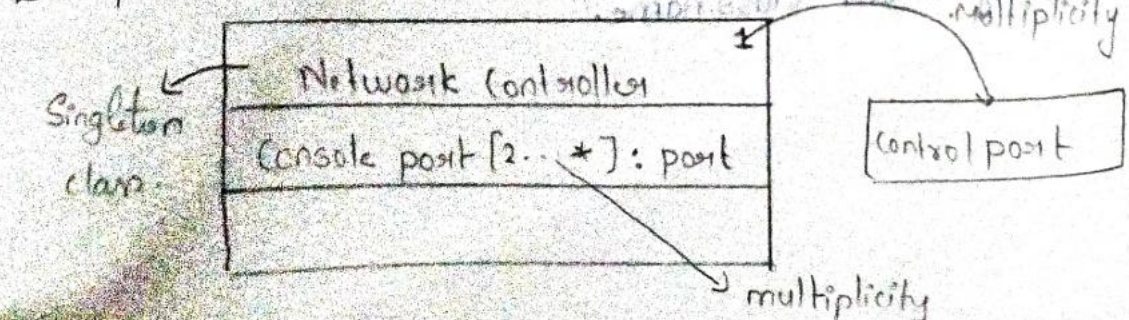
Example:-



Multiplicity:-

It is a specification of the range of allowable Cardinalityies and entity may assume. You can specify the multiplicity of a class by writing its multiplicity expression in the upper right corner of the class icon. And we can also specify the multiplicity of an attribute by writing a suitable expression in brackets [ ] just after the attribute name.

Example:-



Attributes:-

We can also specify the type initial value & the changability of each attribute.

~~The~~

Syntax of Attribute in UML:-

[visibility] name [multiplicity] [: type]

[= initial value] [{ Property-String }]

The following are the legal attribute declarations.

Origin	Name only
+ origin	Visibility & Name
origin: point	Name & type
head: *Item	Name & complex type
name [0...1]: String	Name, Multiplicity & type.
Origin: point = (0,0)	Name, type & initial value
id: integer {frozen}	Name & property

changable	There are no restrictions on modifying the attribute values.
add only	For attributes with a multiplicity greater than one, additional values may be added but once created, a value may not be removed or altered.
frozen.	The attribute values may not be changed after the object is initialized.

Operations:-

We can Specify the visibility and scope of the each operation & we can also Specify the parameters, return types & other properties of each operation.

The name of the operation + its Parameters is called as Operation Signature.

Syntax:-

[visibility] name [{ parameter-list }]
[: return type] [{ property-string }]

display	Name only.
+display	visibility & name
Set (n: Name, s: String)	Name & Parameter
getID (): integer	Name & return type
restart() {guarded}	Name & Property.

[direction] name : type [= default-value]

↓

in → An input parameter; may not be modified.

out → An output Parameter may be modified to communicate information to the other

inout → An input Parameter may be modified.

Standard Elements:-

All the UML extinsibility Mechanisms applied to classes most of an you will use Tagged values to extend class Properties [specifying the version of class] & Stereotypes to Specify new kinds of components.

It defines four standard stereotypes that applied to classes.

1) Meta class:

It Specifies a classifier whose objects are all classes

2) Power type:

It Specifies a classifier whose objects are the children of given Parent.

3) Stereotype:

It Specifies that the classifier is a Stereotype that may be applied to other elements.

4) Utility:-

It Specifies a class whose attributes & operations are all class Scope.

→ Advanced Relationships:-

Dependency:-

It is also known as using relationship that means a change in the Specification of one thing may effect another thing that uses it.

Dependency is rendered as dashed line with directed to the thing that is depended on.

- - - - ->

There UML defines a number of stereotypes that may be

applied to dependency relation ships.

-> There are 17 Stereotypes, all which can be categorized into 6 groups.

Group-1:- In first group there are 8 Stereotypes that applied to the dependency among "classes & objects" in the class diagram.

7/8/25

i) Bind:- It Specifies that the Source instantiates the target templet using the given actual Parameters.

ii) Derive :- It Specifies that the Sources may be computed from the target.

iii) friend :- It Specifies that the Source is given Special visibility into the target.

iv) instance of :- It Specifies that the Source object is an instance of target classifier.

v) instantiate :- It Specifies that the Source creates instance of the target.

vi) Powertype: It Specifies that the target is a Power type of the Source. A powertype is a classifier whose objects are all the children of given parent.

vii) Refine:- It Specifies that the Source is at a final degree of abstraction then the target.

viii) Use: It Specifies that the Semantics of the Source element depends on the Semantics of Public Part of the target.

Group-2:- In this there are two Stereotypes that applied to dependency relationships among "Packages".

i) Access:- It Specifies that the Source package is granted the right to reference the elements of the target package.

ii) import:- A kind of Access that Specifies that the Public contents of the target (access) Package enter the flat name space of the Source as if they had been declared in the Source.

Group-3:- There are two Stereotypes that applied to dependency relationship among "usecases".

i) extend:- It Specifies that the target usecase extends the behaviour of Source.

ii) include:- It Specifies that the Source usecase explicitly incorporates the behaviour of another usecase at a location Specified by the Source.

Group-4:- We have three Stereotypes when "modeling interactions" among objects.

i) Become:- It specifies that the target is the same object of the Source but a later point in time and with possibly different values states or roles.

ii) call:- It Specifies that the Source operation invokes the target operation

iii) copy:- It Specifies that the target object is an exact but independent copy of the Source.

Group-5:- One stereotype that you will encounter in the context of "state machine" is 'send'.

Send:- It specifies that the source operation sends the target event.

Group-6:- finally, one stereotype that you will encounter in the context of "organizing the elements" of your system into subsystems and models.

Trace:- It specifies that the target is an historical ancestor of the source.

Generalization:-

If you want to specify a shade of meaning the UML defines one stereotype (Implementation) and four constraints that may be applied to the generalization relationships.

Implementation:- It specifies that the child inherits the implementation of parent but doesnot make public nor support its interface there by violating substitution

⇒ There are four standard constraints that applied to generalization relationships.

* Complete
* incomplete
* disjoint
* overlapping

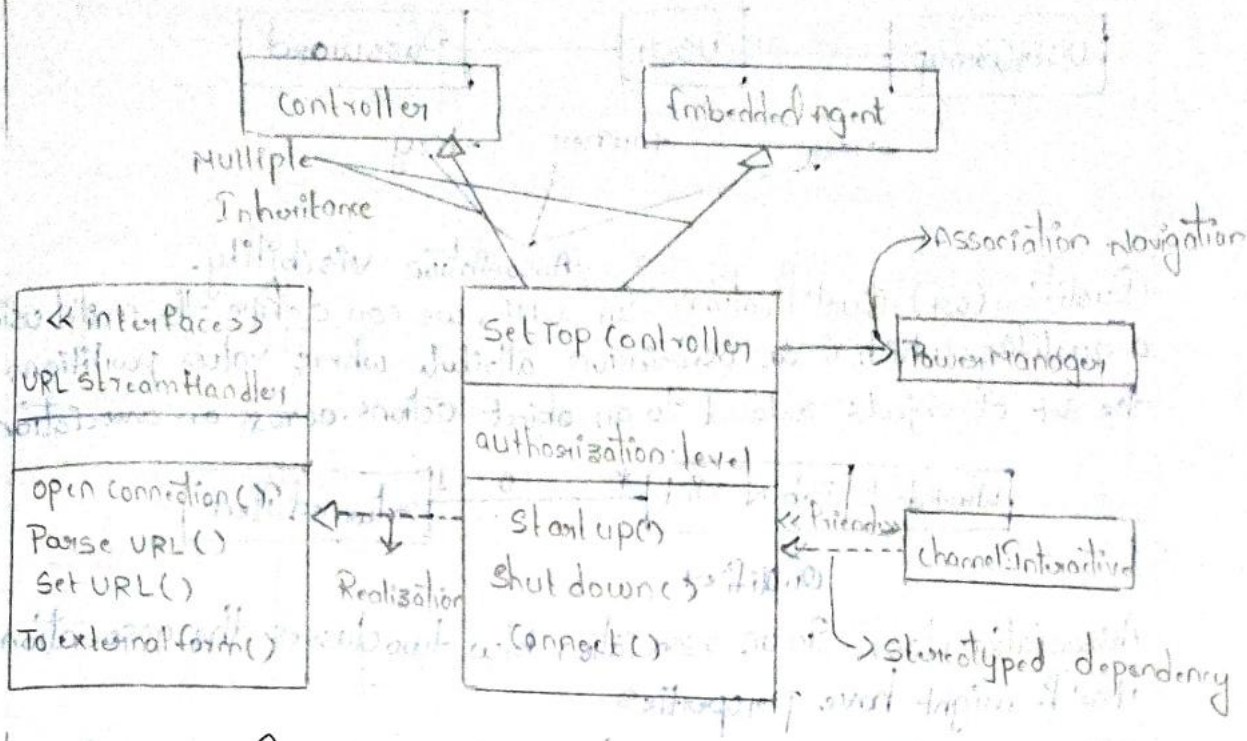
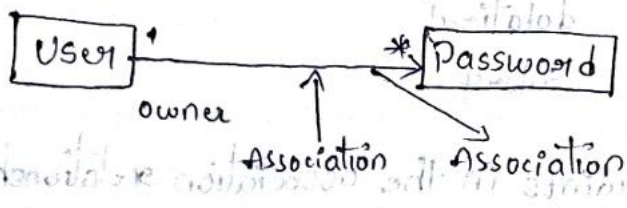


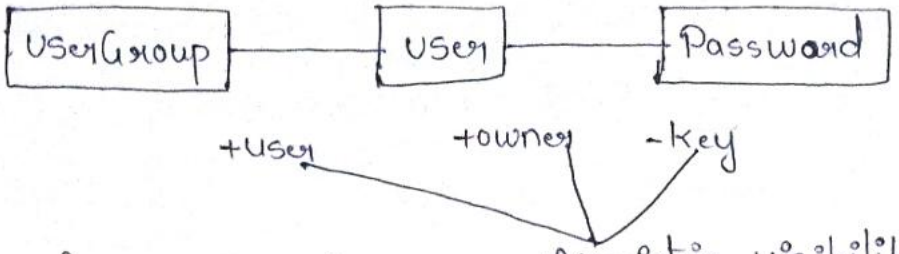
fig:- Advanced Relationships.

Association: It is a structural relationship specifying that objects of one thing are connected to the object of another. Graphically, an association is rendered as a solid line connecting the same or different classes.

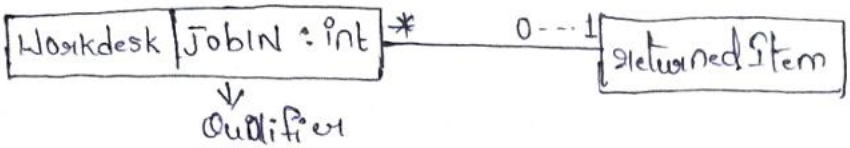
Navigation:- Navigation in association is Bi-directional, you can explicitly represent direction of navigation by adorning an association with an arrow head pointing to the direction of the traversal.



Visibility: we can specify 3-levels of visibility for an association and by appending a visibility symbol to a role name. The Public visibility indicates the object are accesible to any object outside of a association. The private visibility indicates that objects at the end are not accesible to any objects outside the association. The protected visibility indicates the objects at that end are not accesible to any objects outside the association except the children of other end.



Qualifier (or) Qualification :- In UML, we can define the model using a qualifier which is an association attribute where values partitions the set of objects related to an object actions. across an association

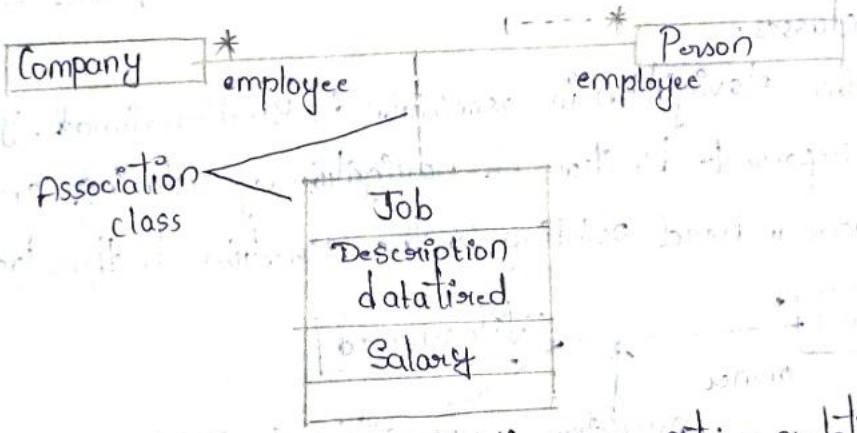


Association class:- In an association blw two classes the association itself might have properties.

→ An association class can be seen as an association that also has class properties.

→ you can rendered an association class as a class symbol attached by a dashed line to an association.

Ex:-



→ there are 6 Constraints in the association relationships.

1) Implicit 2) ordered 3) changable
4) addonly 5) frozen 6) XOR

Realization:- It is a Semantic relationships blw classifiers in which one classifier specifies a contracts that another classifier guarantees to carry out.

→ Graphically the realization is rendered as a dashed directed line with a large open arrow head pointing to the classifier that specifies the contract.

→ Realization Notation is a combination of notations for dependency and weak generalization.

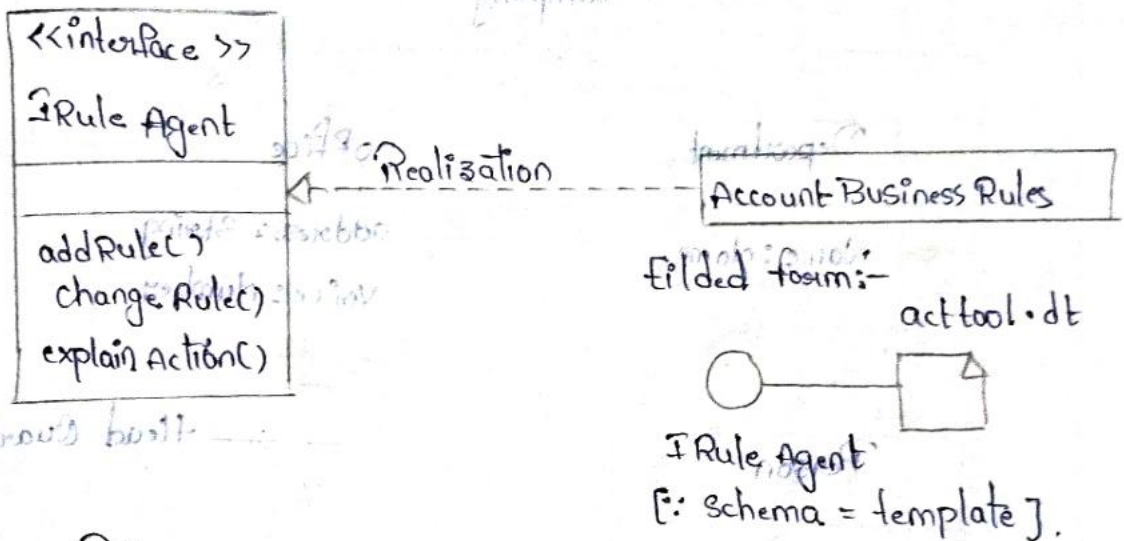
→ you will also use realization in two circumstances i.e., in the context of interfaces & the context of collaborations.

→ you can represent realization in two ways:-

i) dashed hallow arrow head -----▷

ii) Elided format i.e., using the interface lollipop notation

Example:-



class Diagram:-

→ It is a collection of set of classes, interfaces, collaborations and their relationships.

→ Graphically the class is represented as a Rectangle (width three compartments.

→ Like all other diagrams the class diagram may contains nodes and constraints.

→ And we use class diagram to model the static design view of the system. and this view primarily supports functional requirements of the system and the services provided by the system to its endusers.

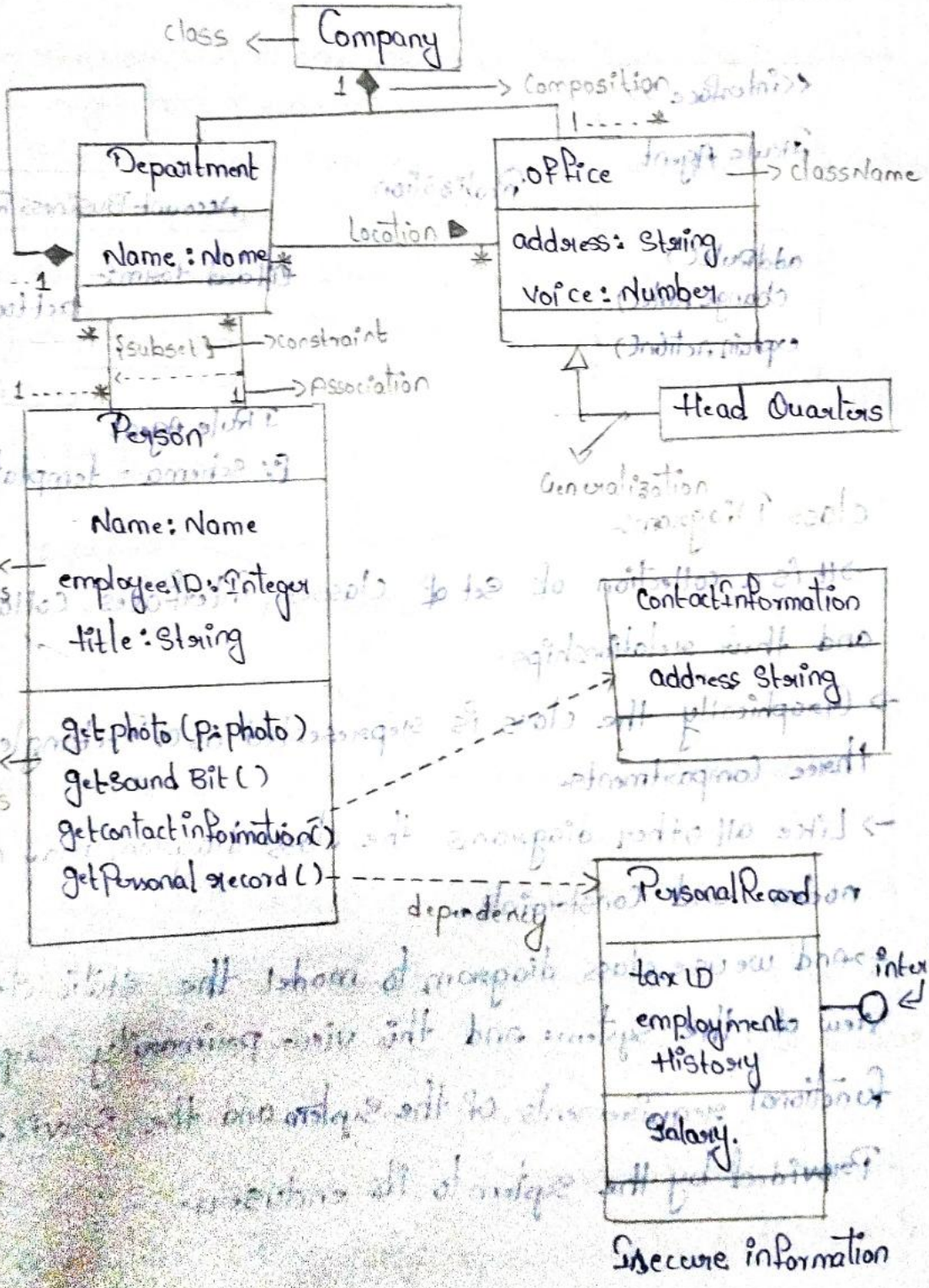
→ we will use the class diagram in one of three ways

[i] To model the vocabulary of the system.

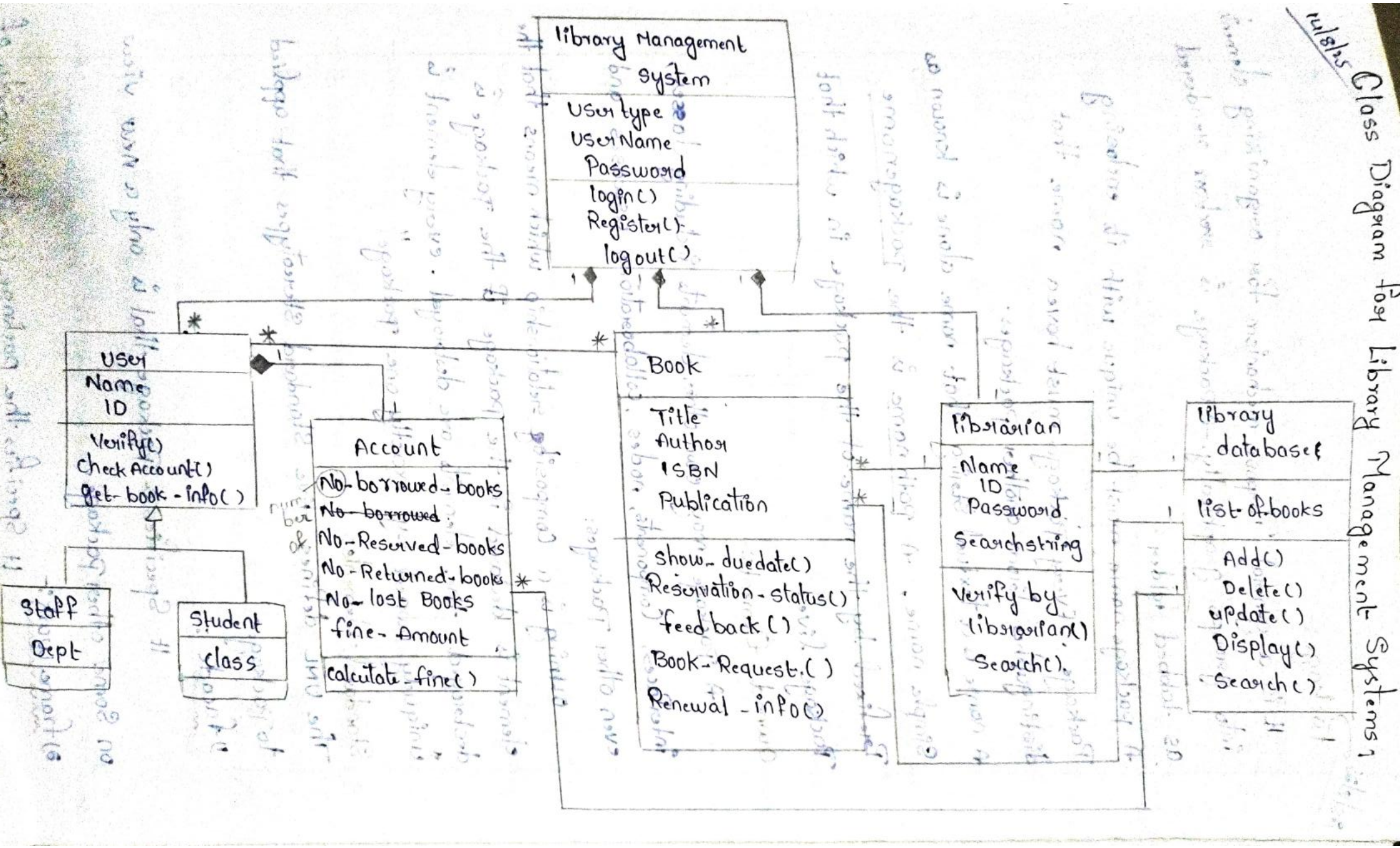
[ii] To model simple collaborations → working together.

[iii] To model a logical database schema.

Example :-



Class Diagram for Library Management System



28/8/25

Package:-

It is a general Purpose mechanism for organising elements into groups graphically. A Package is rectangle rendered as tabbed folder.

A package name must be unique with its enclosing Package. Every Package must have a name that distinguishes from another Packages.

A name is a textual String that name alone is known as Simple name, A Path name is the Packagename Prefixed by the name of the package in which that Package lives.

Owned Elements:-

A Package may owa other elements including classes, interfaces, Components, nodes, collaborations, usecases and even other Packages.

OWning is a Composite relationship which means that the element is declared in the Package if the package is destroyed the elements are destroyed. every element is uniquely owned by exactly one package.

Standard Elements:-

The UML defines five Standard Stereotypes that applied to Packages.

1) facade :-

It Specifies a Package that is only a new view on Some other Package.

2) framework :-

It Specifies the package Consisting mainly of

Patterns.

3) Stub :-

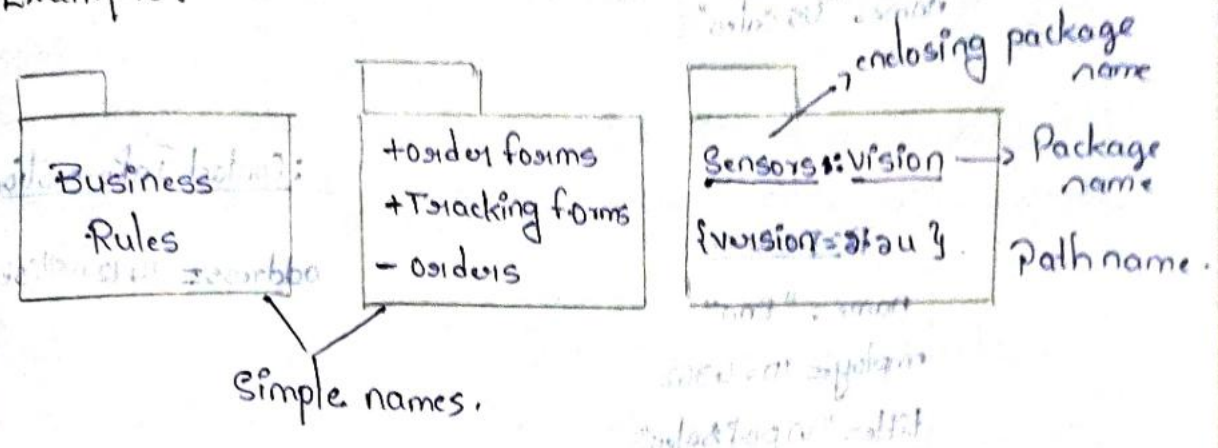
// Specifies the Package that Serves as a Proxy for the Public Contents of another Package.

4) Sub system:-

// Specifies a Package representing an independent Part of entire system being model.

5) System:- // Specifies the Package representing the entire System being modeled.

Example :-



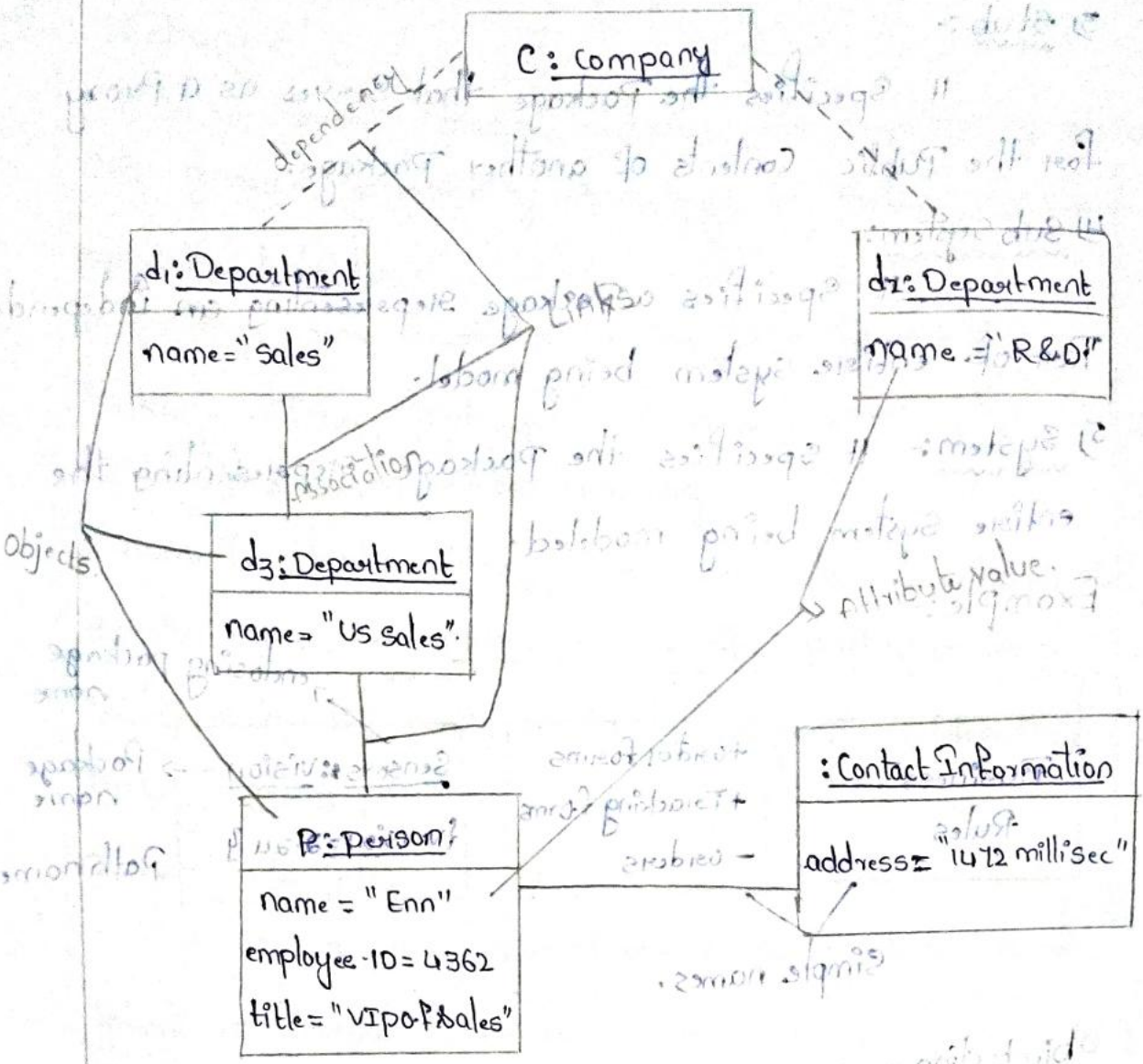
Object diagram:-

→ It is a diagram that shows a set of objects and their relationships at a point in time and an object diagram is a collection of vertices and arcs.

→ object diagram Commonly contains objects & links.

→ It models the static design view or static Process view of the system just like a class diagrams. This ~~New~~ view Primarily supports the functional requirements of the system i.e., the services the system should Provide to endusers.

⇒ Object diagrams let you model the static data structures


C: Company
dependency
d1: Department
name = "Sales"
d2: Department
name = "R&D"
Association
d3: Department
name = "US sales"
objects
P: Person
name = "Enn"
employee-ID = 4362
title = "VIP of Sales"
: Contact Information
address = "1472 millisec"

19/25

Unit - III

Part - I : Basic Behavioural Modelling.

Interaction, Interaction Diagrams:

Interaction :- It is a behaviour that comprises a set of Messages exchanged among a set of objects within a context to accomplish a purpose.

Objects & roles: The objects that participate in an interaction are either Concrete things (or) Prototypical things. As a Concrete thing an object represents something in the real world.

In collaboration the objects you find are Prototypical things that play particular roles not specific objects in the real world.

Links :- It is a Semantic Connection among objects.

→ A link is an instance of an Association.

→ Wherever there is a link between two objects, one object can send a message to the other object.

→ A link specifies a path along which one object can dispatch a Message to another object.

→ If you need to be more Precise about how that path exists you can adorn the appropriate end of the link with any of the following standard stereotypes.

1) Association: Specifies that the corresponding object is visible by Association.

2) Self :- Specifies that the corresponding object is visible because it is the dispatcher of the operation.

3) Global: Specifies that the corresponding object is

visible because it is an enclosing scope.

4) Local: Specifies that the corresponding object is visible because it is in the local scope.

5) Parameter: Specifies that the corresponding object is visible because it is a parameter.

Messages:- It is a specification of communication among objects that convey information with the expectation that activity will ensure when you pass a message, the action that results an executable statement that forms on abstraction of a computational procedure. An action may result in a change in state.

In UML you can model several kinds of actions.

① call:- Invokes an operation on an object may send a message to itself Resulting in local invocation of an operation.

② Return:- Returns a value to the caller.

③ Send: Sends a signal to an object.

④ Create: Create an object

⑤ Destroy: Destroys an object an object may commit Suicide by destroying itself.

Interaction Diagrams:-

It commonly contains

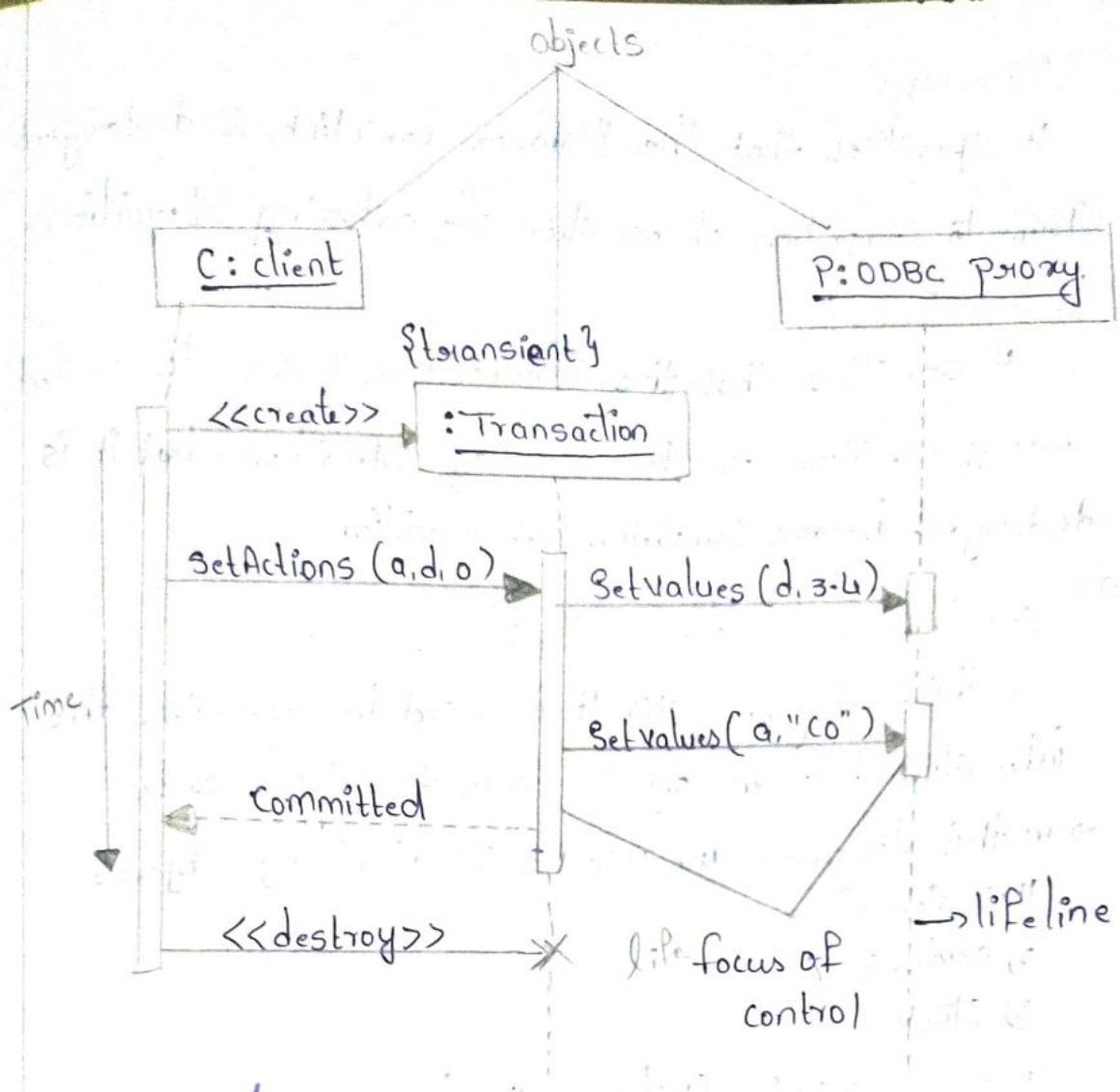
→ Objects
→ links
→ Messages

Sequence Diagram:- It objects can be represented in

- simple object name
- Anonymous object name
- object names its class

It emphasizes time ordering of messages.

All the messages will be ordered sended in a sequence manner.



4/9/25 Creation, Modification and Destruction :-

Most of the time the objects you show participating in an interaction exist for the entire duration of the interaction, however in some interactions, the objects may be created (specified by a create message) and destroy (specified by a destroy message) In order to specify if an object or link enters and or leaves during an interaction you can attach of one of the following constraints to the element.

1. New:- Specifies that the instance (or) and links is created during execution of the enclosing interaction.

2. Destroy:

It specifies that the instance (or) links is destroyed prior to completion of execution of enclosing interaction.

3. Transient:

It specifies that the instance (or) links is created during execution of the enclosing interaction. but it is destroyed before completion of execution.

Slab 8 Sequential diagram for online placing order

In this diagram, It is used to visualise the interaction between objects in a sequential order.

→ In this diagram we use 3 participating objects.

1) Customer
2) Order &
3) Stock

step-1 & step-2:- Customer creates an order.

Step-3: Customer add items to the order.

step-4 & step-5: Each item is checked for availability in inventory.

Step-6, 7 & 8:- If Product is available it is added to the order.

Step-9: return [received to the Customer].

Step-10 & 11: Save order & destroy order.

Notations:-

Synchronous - ⟶

Asynchronous - ⟶

Iteration - ↻

Response - ⟵ - - -

Create object - ⟶ «create»

destroy object - ⟶ X

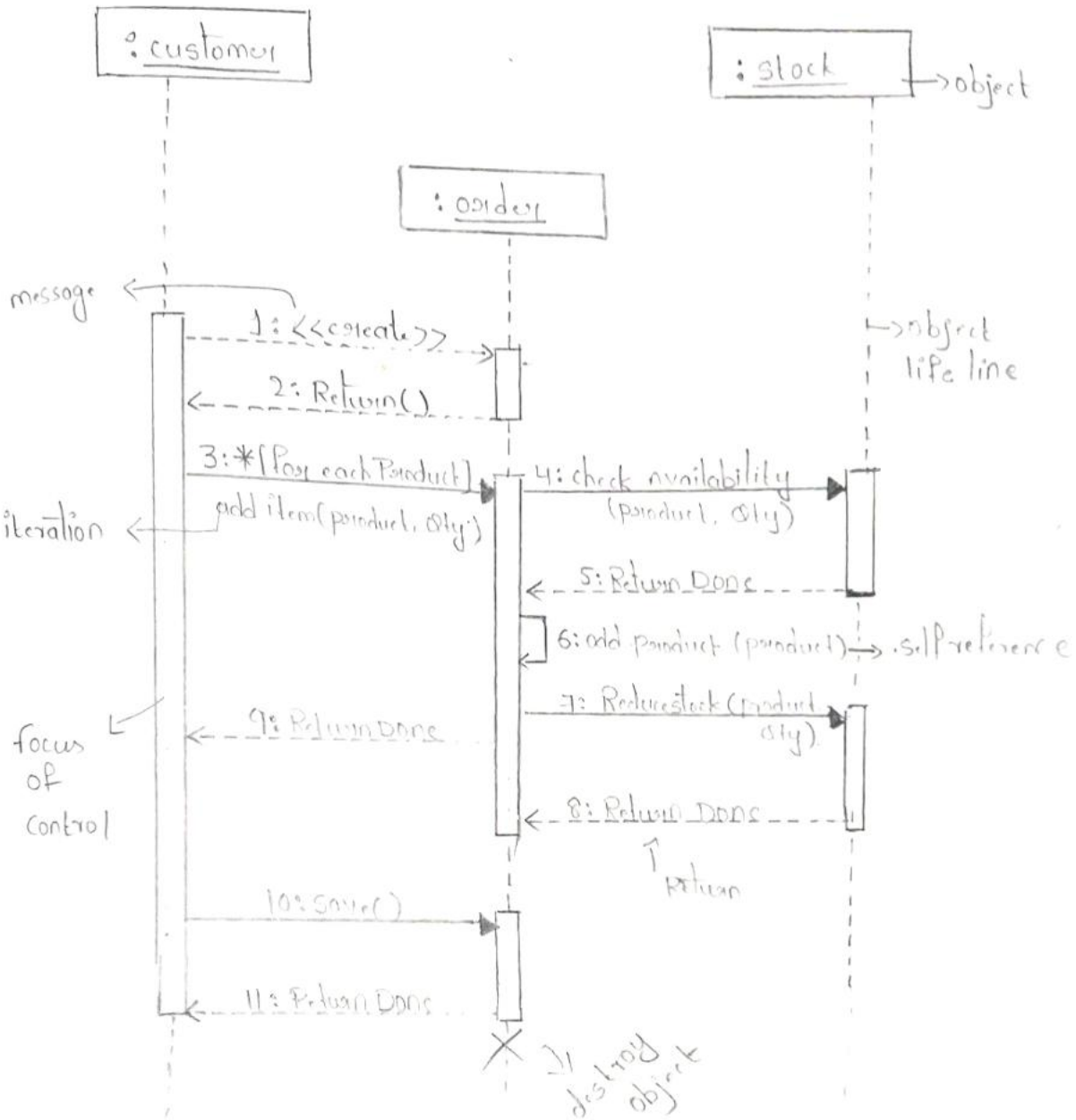
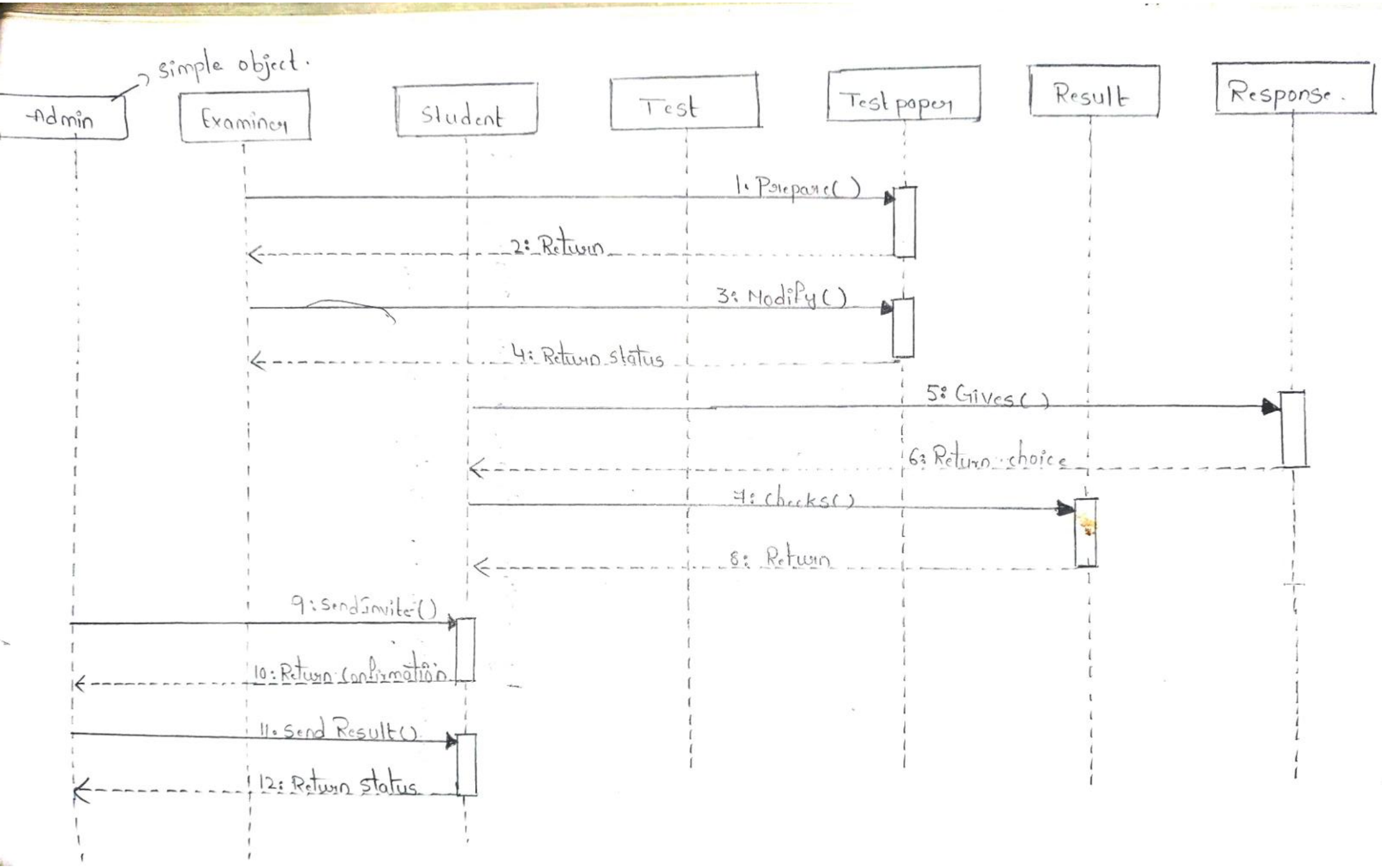


fig: Sequence diagram for online placing order

Sequence diagram for online exam system:


simple object.
Admin
Examiner
Student
Test
Test paper
Result
Response.
1: Prepare()
2: Return
3: Modify()
4: Return status
5: Gives()
6: Return choice
7: checks()
8: Return
9: sendInvite()
10: Return confirmation
11: Send Result()
12: Return status

19/25 Usecase diagram:-

It defines the interaction between system and its endusers.

Notations:-

Actors: It acts as a external entities that interact with the system. these includes users other systems & hardware devices and these act as initiate usecases and relieve the outcomes.

→ Actors are rended as "stick figures"

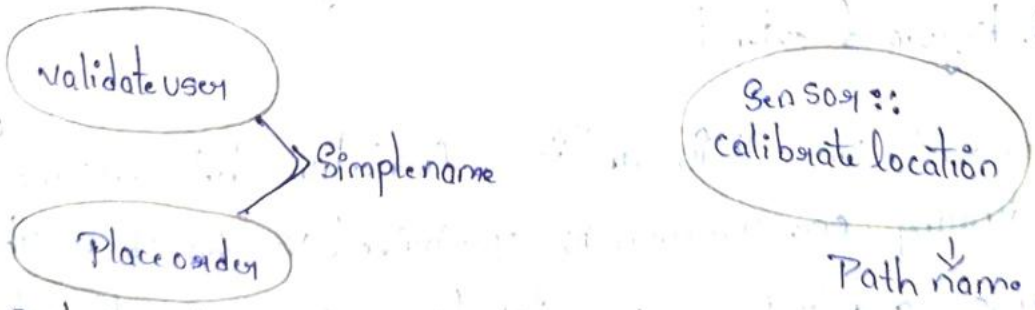
Usecases: It represents specific things the system can do

→ The usecases are represented by "solid ellipse local" ⬭

→ the use case name (as the) must be unique within its enclosing Package. every usecase must have a name that ~~dinguishes~~ distinguishes it from other usecases. A Name is a textual string.

we can represent the name of a usecase in 2 ways.
ie:- Simple name &
Path name.

The name alone is known as Simple name and the path name is the usecase name. Prefixed by the name of the Package in which that the use case lives.



System boundary: It is typically represented by a rectangular box that surrounds all the usecases of the system.

→ It is the visual representation of the scope or limits of the system you are modeling. It defines what is inside the system and what is outside and

→ The main purpose here is to clearly outline the boundaries of the system indicating which components are internal to the system and which are external actors or entities interacting with the system.

Relationships:-

→ Association:- It represents an interaction / communication between actors & the use case.

It is rendered by a solid line connecting the actor to the particular usecase "_____".

or uses relationship

→ Include: It includes

here the usecases includes the functionalities of another usecases. & it is denoted by "dashed line arrow pointing from the including use case to the included use case"

<<include>> ---------->

→ Extend: The use case can be extended by another usecase under specific conditions. it is represented by dashed arrow with keyword extend.

<<extend>> ---------->

→ Generalization:- It is an "is-a connection" between two use cases indicating one usecase is specialized version of another usecase and it is represented by an arrow pointing from the specialized usecase to General usecase

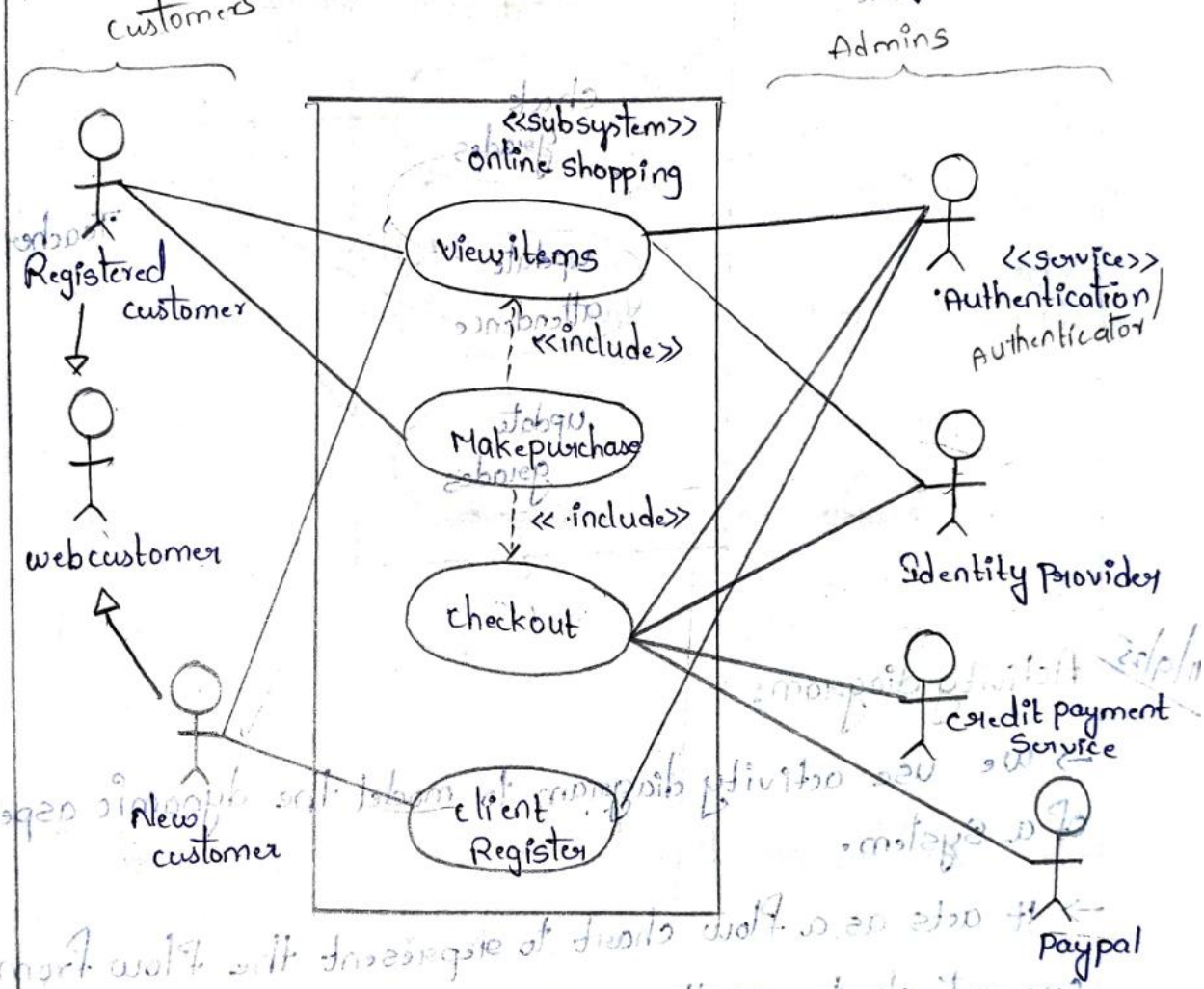
child class ——▷ Parent class

Usecase diagram for online shopping system:

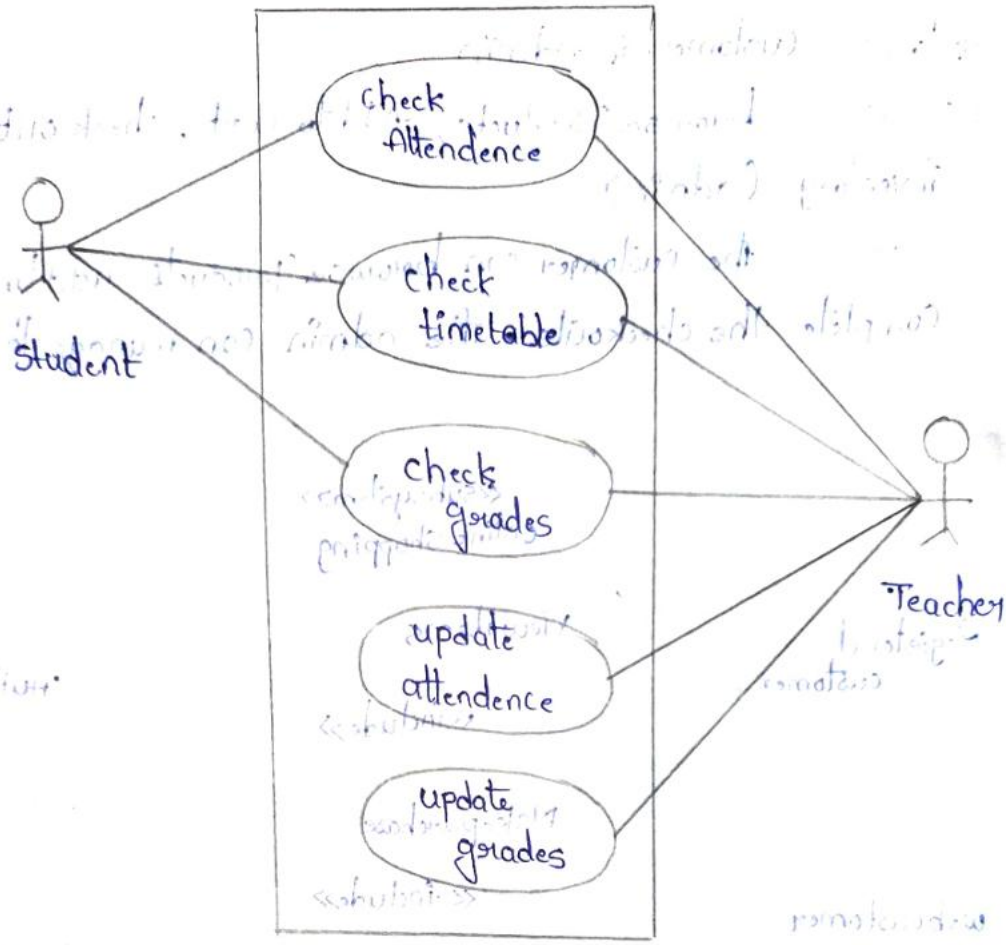
Actors:- customers & Admin

Usecases:- browse Products, Add to cart, check out, Manage inventory (admin)

relations:- the customer can browse Products, add to cart & complete the checkouts. the admin can manage the inventory



Usecase diagram for Student Management system



n/qns Activity diagram:

→ We use activity diagram to model the dynamic aspects of a system.

→ It acts as a flow chart to represent the flow from one activity to another activity.

→ The activity can be described as an operation of the system.

→ It is similar like flowchart with more specific symbols and notations

Purpose:

→ It draws the activity flow of system.

→ It describes the sequence from one activity to another activity. &

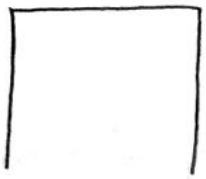
→ It describes parallel branch & Concurrent flow of the

system.

→ Graphically, Activity diagram is a collection of vertices and arcs.

→ The activity diagram mainly contains, Action states & Activity states, transitions & Swimlanes

Activity diagram Notations :-

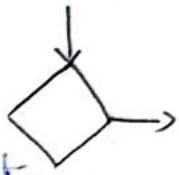


Swimlane

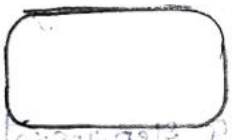
initial state

end state

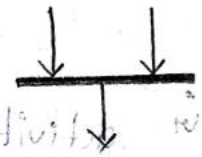
Control flow



decision Node



Action state & Activity state



fork

Join

initial state :- It is represented with black filled circle which acts as an entry point.

End state :- It acts a exit point and it is represented with Solid ball inside a circle.

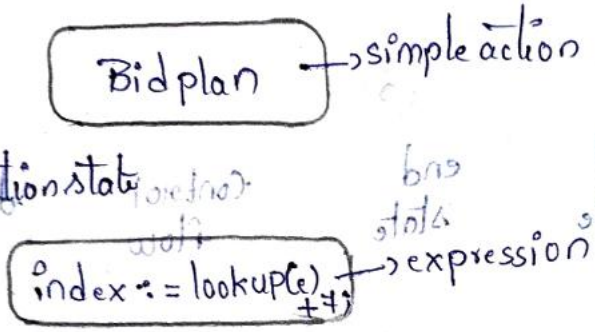
Action states & Activity states :-

Action state :- It represents execution of an action on objects or by objects. That means, you might call an operation on an object, send a signal to an object, create or destroy an object. These are all called as action states because each representing the execution of an action.

→ We represent action state using a rectangle with rounded corners.

→ Action states are Atomic in nature that means, events may occur but the work of an action state is not interrupted.

Example:-



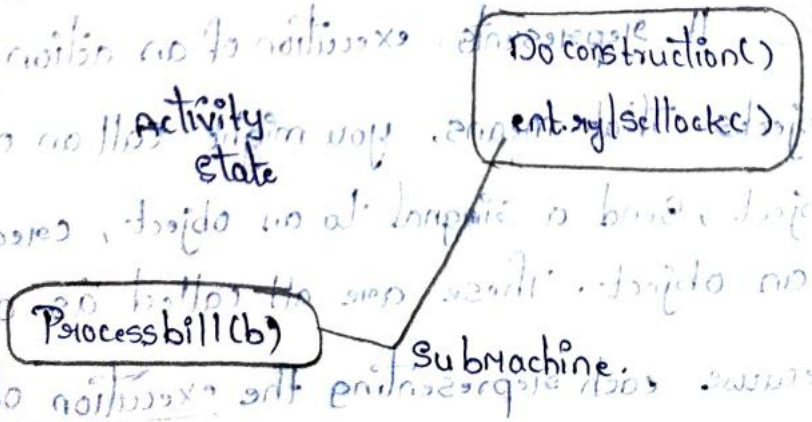
Activity State:-

→ The Activity state, their activity being represented by another activity diagrams.

→ Generally the activity states are not atomic that means they may be interrupted & in general they are considered to take some duration to complete.

→ There is a notational distinction between action state & activity state except the activity state may have additional parts such as entry & exit actions.

Example:-



Transition:

→ control flow:-

when action or activity of a state completes the flow of control passes immediately to the next action or activity state for that purpose we use transitions to show the path from one action or activity state to the next action or activity state.

You can represent the transition as simple directed line. ——→

→ branching:

We can include a branch here which specifies alternative paths taken based on some boolean expression. Here you represent a branch as diamond, a branch may have one incoming transition and two or more outgoing transitions

→ forking joining

⇒ fork:- Here we use synchronization bar which is represented as thick horizontal or vertical line.

→ The fork represents the splitting of single flow of control into two or more concurrent flows of controls

→ A fork may have one incoming transition & two or more outgoing transitions

⇒ join:- It represents the synchronization of two or more concurrent flows of control.

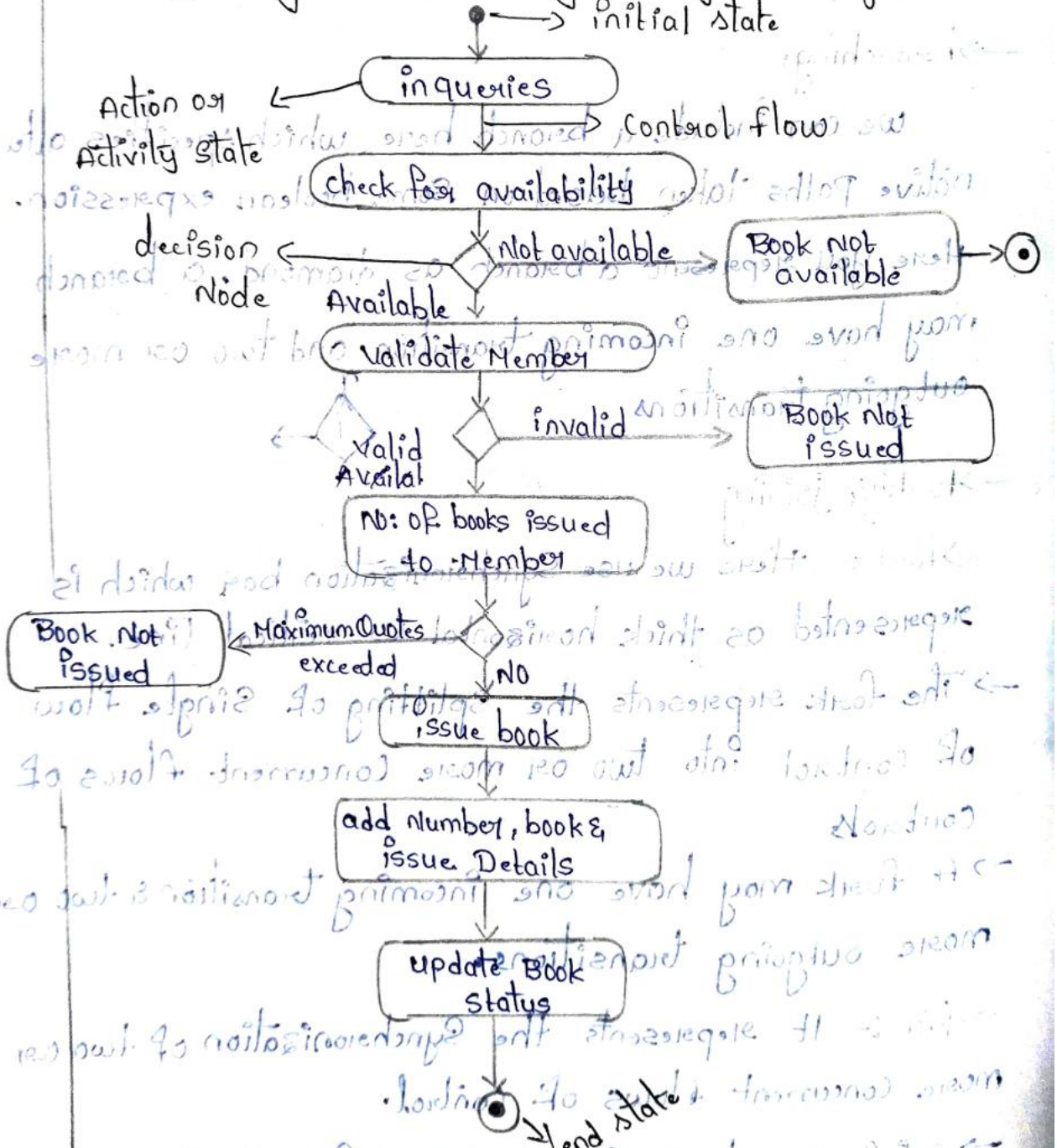
→ A join may have two or more incoming transitions and one outgoing transition

→ Swimlanes:- It is especially useful when you are modelling workflows of business processes to partition the activity states on an activity diagram into groups.

→ Each group represents the business organisation responsible for those activities.

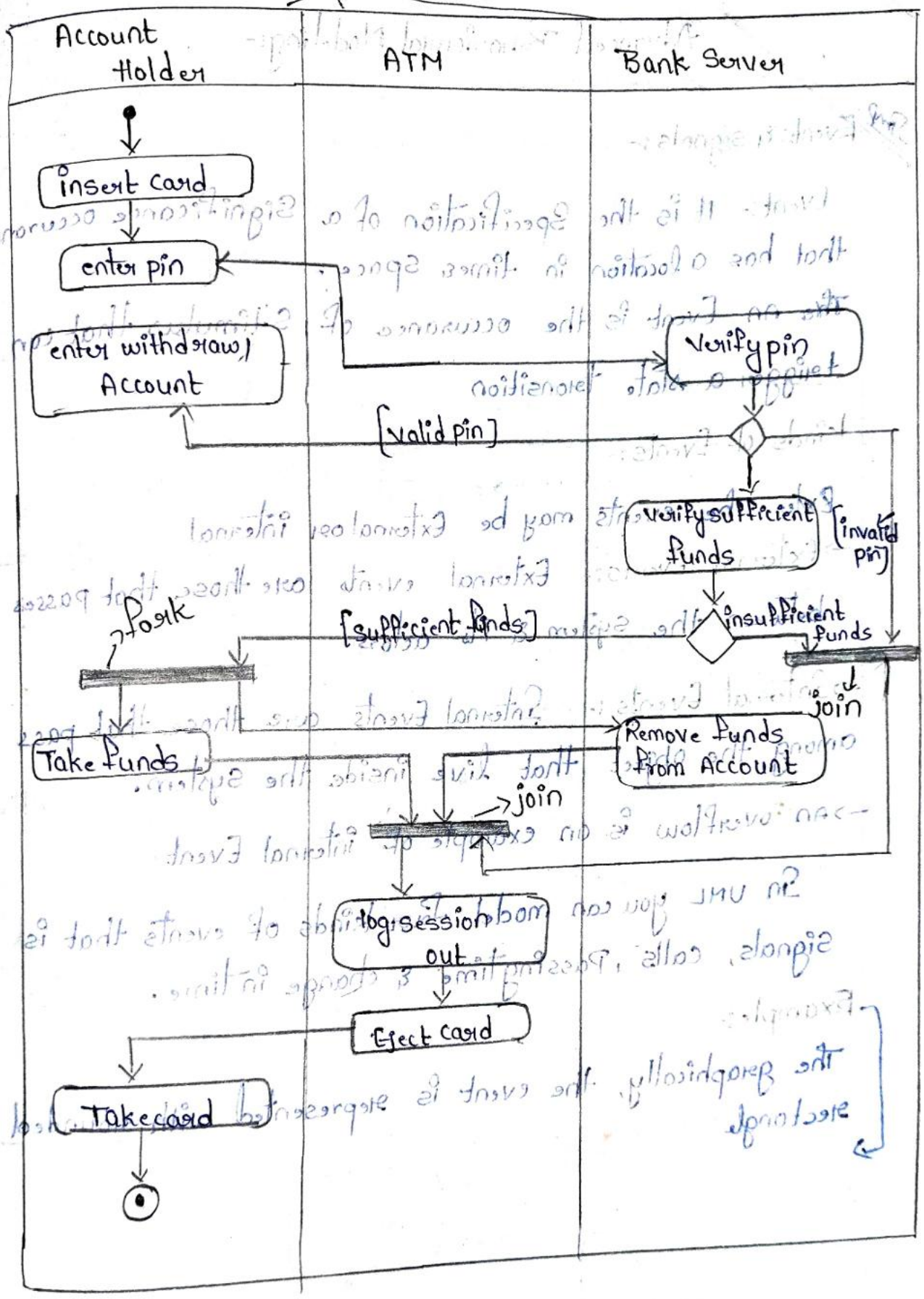
→ It is represented by a vertical solid line.

Activity diagram for library Management system:



Library Management System

Activity diagram for ATM:



12/9/25

Unit-4

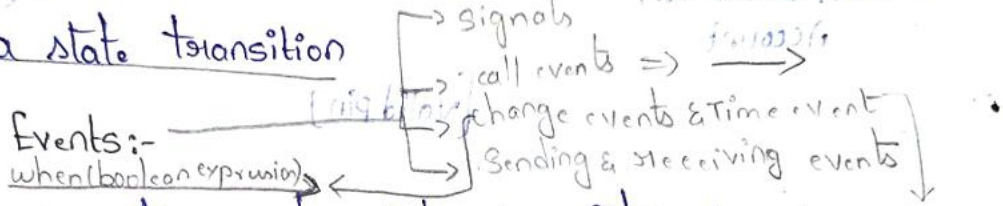
Advanced Behavioural Modeling:-

Events & signals:-

Event:- It is the Specification of a Significance occurance that has a location in time & space.

An Event is the occurance of stimulus that can trigger a state transition

Kinds of Events:-



The events may be External or internal

→ External Events:- External events are those that passes between the system & its actors

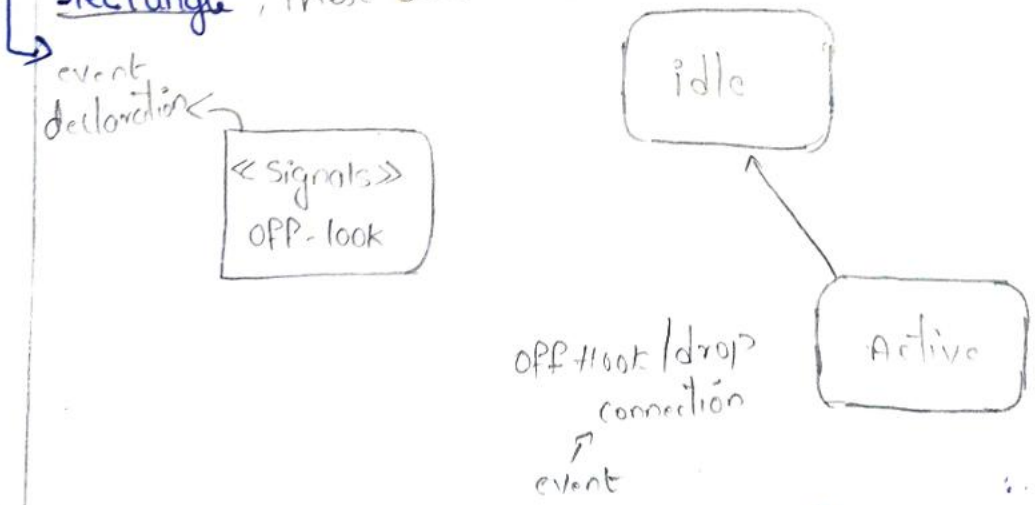
→ Internal Events:- Internal Events are those that pass among the object that live inside the system. (ex: overflow)

→ An overflow is an example of internal Event.

In UML you can model four kinds of events that is signals, calls, Passing time & change in time.

Example:-

The graphically, the event is represented with rounded rectangle, These are used in the State machines.



Signal:-

→ A signal is a kind of event that represents the specification of Asynchronous stimulus Communication blw instances.

→ It represents a named object i.e. the dispached asynchronously by one object and then received by another objects.

<<send>> - - - - - - - ->

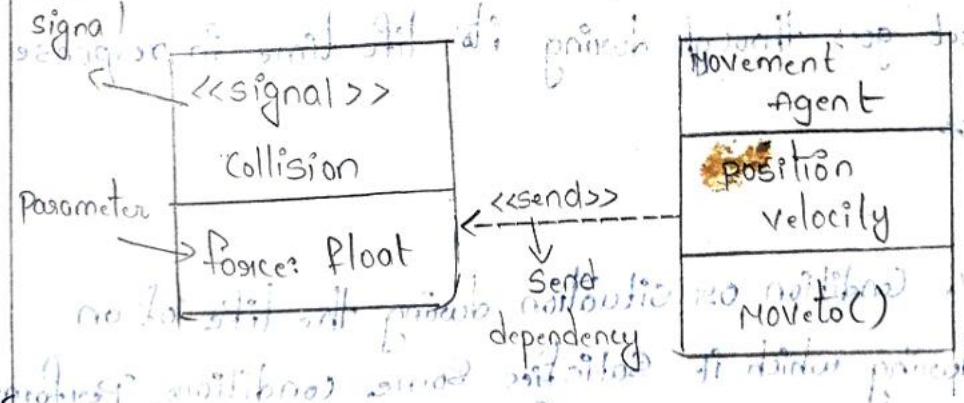
→ The signals may also involved in Generalization relationships

→ A signal may be sent as a action of state transition in a state machine or the sending of message in an interaction.

→ The execution of an operation can also send signals in UML you model the relationship blw an operation and the events that it can be send by using a dependency relationship stereotyped as Send.

→ In UML you model signals as stereotyped classes and you can use a dependency stereotype as Send to indicate that an operation sends a particular signal.

Examples:-



Call Events:

The call events represents the dispatch of an operation. It is Synchronous in nature

→ This means, when an object invokes an operation

on object that has a state machine. the control passes from sender to receiver. the transition is triggered by the event. the operation is completed; the receiver transitions to a new state & the control returns to the sender.

Example:-

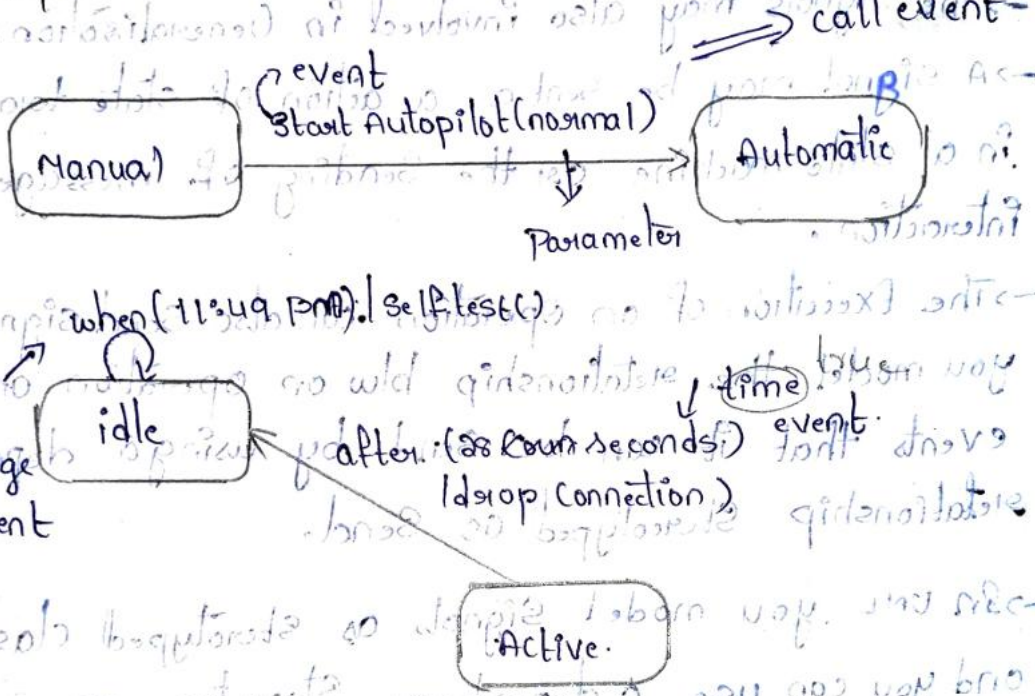


fig: time & change events.

17/9/25

State Machine:-

It is a behaviour that specifies the sequence of states and object goes through during its life time in response to events.

->State:-

It is a condition or situation during the life of an object, during which it satisfies some condition, Performs some activity or wait for some event.

Graphically a state is rendered as a rectangle with rounded corners.

Transition:-

It is a relationship blw two states indicating that an object in first state will perform certain actions & enter the second state when a specified event occurs & specified conditions are satisfied.

The Transition is rendered as a solid directed line ⟶

Imp Statemachine diagram for online order system:-

→ on the event of an order being received (with we transition from our initial state to unprocessed order state.

That unprocessed order is then checked.

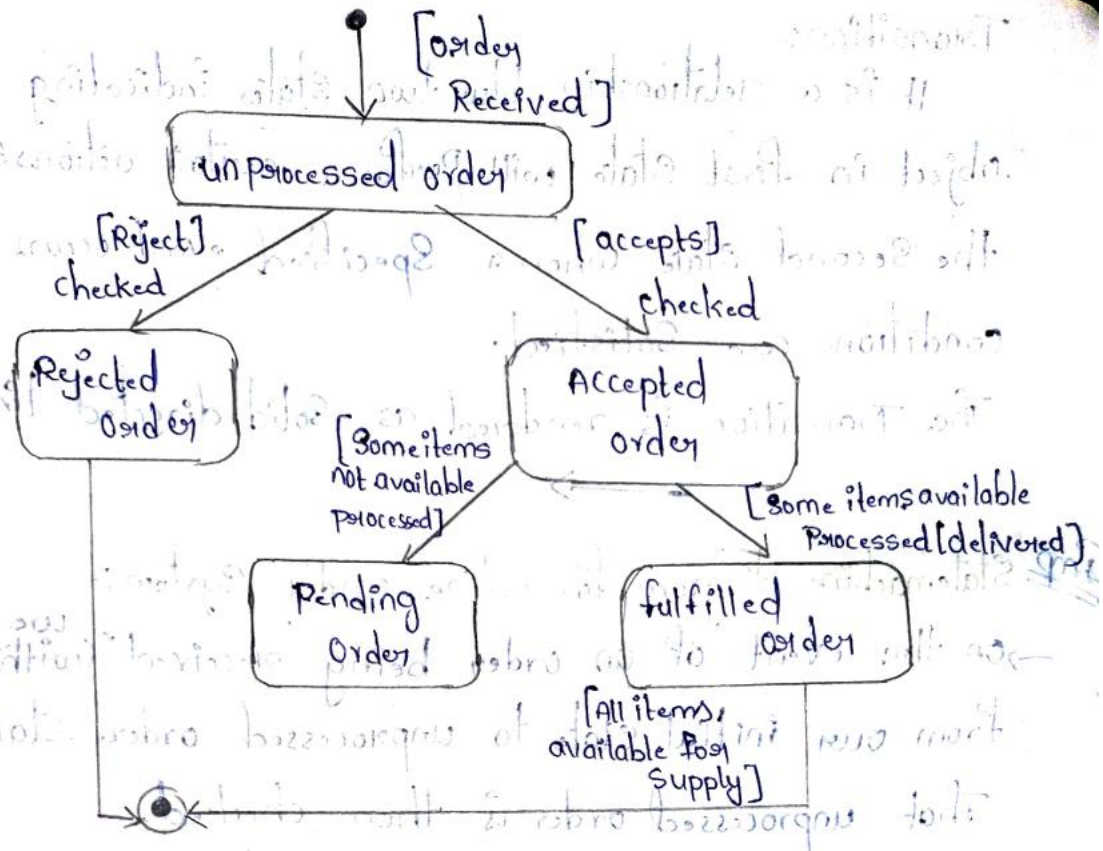
→ If the order is rejected, we transition to the rejected order state.

→ If the order is accepted & we have items available we transition to the fulfilled order state.

→ However if the items are not available, we transition to the pending order state.

→ After the order is fulfilled, we transition to the final state.

In this example, we merge two states i.e., fulfilled orders and rejected order into one final state.



Online Order System

A state has several parts the state

1) Name - A textual string that distinguishes from other entry& state. A state may anonymus meaning it has no name.

2) Entry & Exit actions - Actions executed on entering & exiting the state respectively.

3) Internal Transitions - Transitions that are handled without causing a change in state.

4) Substates - The nested structure of a state involving disjoint or concurrent ~~substent~~ substates.

5) Deferred Events - A list of events that are not handled in that state but rather are postponed or queued for handling by the object in another state.

Processes & Threads:-

Process: It is a heavy weight flow that can execute concurrently with other processes.

Thread: It is a light weight flow that can execute concurrently with other threads within the same process.

Active class: Active class is a class whose instances are active objects; an active object is an object that owns a process or thread & can initiate control activity.

→ Graphically, the active class is rendered as rectangle with thick lines.

→ Processes & threads are stereotyped active classes.

Flow of Control:-

These are of two types, i.e., Sequential System & Concurrent system.

→ Sequential system: There is a single flow of control that is one thing & only one takes place at a time.

→ Concurrent system: There is multiple simultaneous flow of control more than one thing can takes place at a time.

Active Classes & Events:-

→ Active classes are just like class which represents an independent flow of control

→ when an active object is created the associated flow of control is started. when the active object is destroyed the associated flow of control is destroyed.

→ Active classes shares the same properties as like other classes.

Standard Elements:

All the UML Extensibility Mechanisms applied to the active classes. It defines two standard stereotypes that apply to active classes.

i) Process: It specifies a heavy weight flow that can execute concurrently with other processes.

ii) Thread: It specifies a light weight flow that can execute concurrently with other threads within the process.

Communication:- when an objects collaborate with one another they interact by passing messages from one to another in a system with both active & passive objects.

There are 4 possible combinations of interactions that must consider.

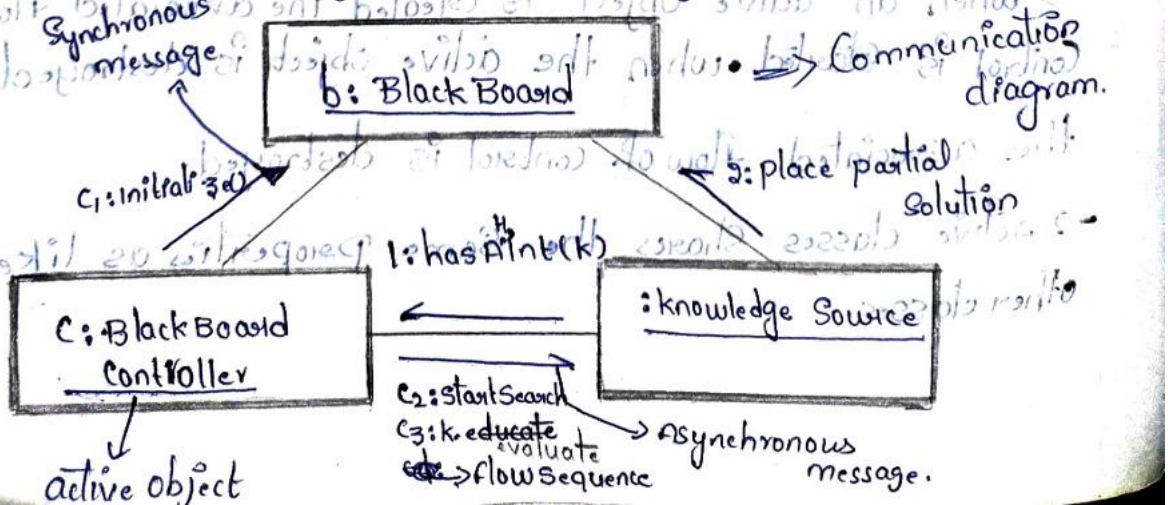
1st message - It may be passed from one object to another

In UML, Synchronous message is rendered as a full arrow and Asynchronous message is rendered as half arrow.

2nd message - It may be passed from one active object to another

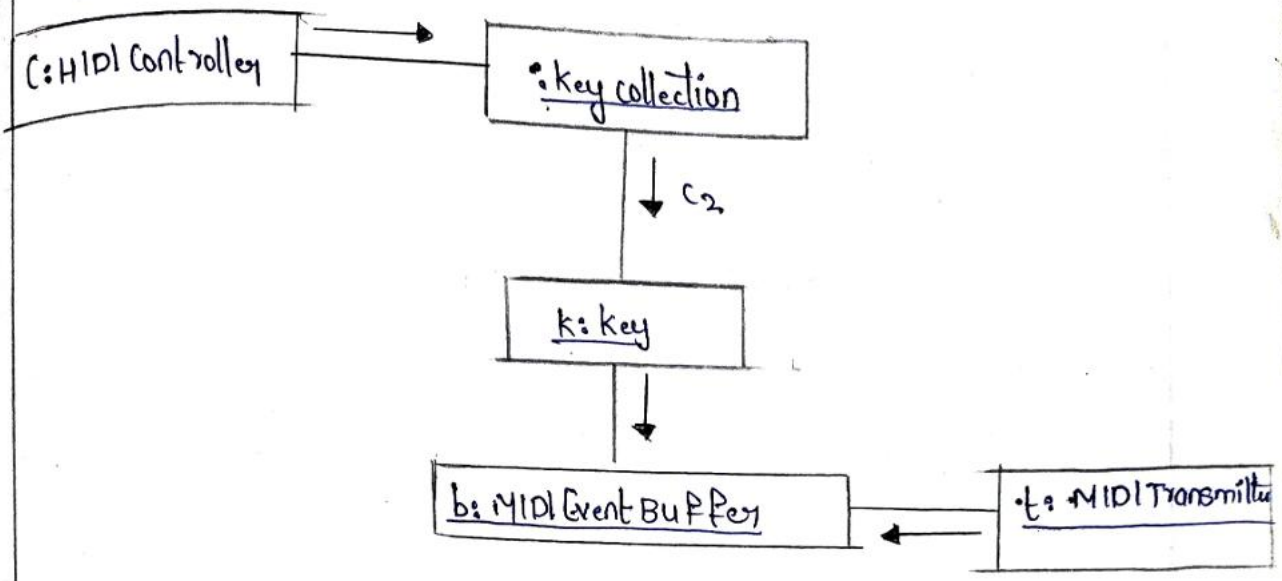
3rd message - It may be passed from one active object to passive object.

4th message - A message may be passed from active passive object to active object.

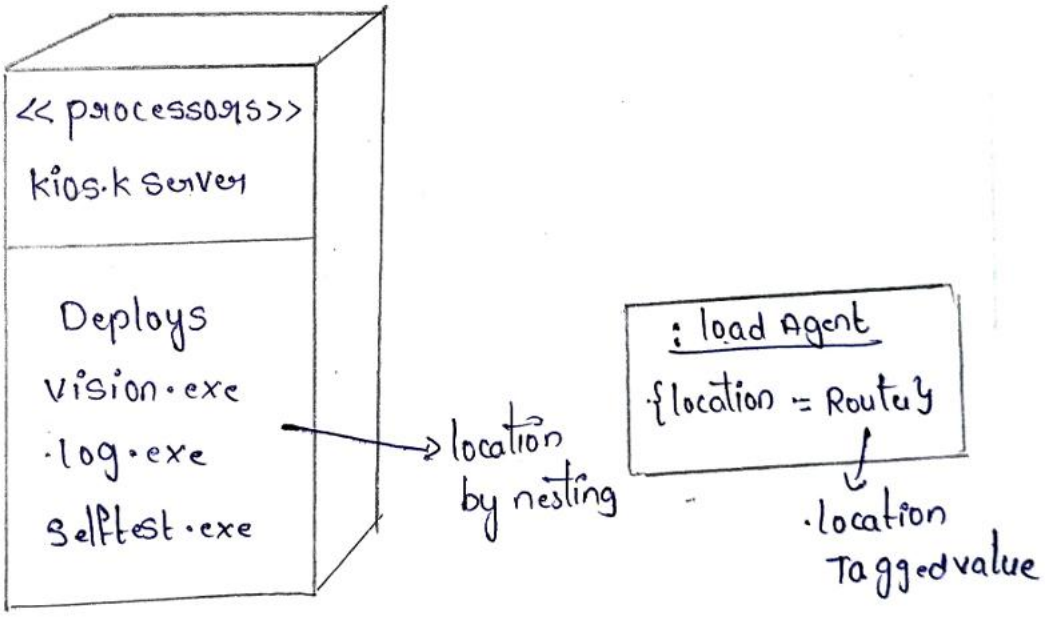


Time & Space:-

Time:-



location



Component:-

It is the physical & replacable part of the system that confirms to and provides the realization set of interfaces. Graphically a component is rendered as a Tabbed rectangle.

→ Names:- A component name must be unique in nature with its enclosing package.

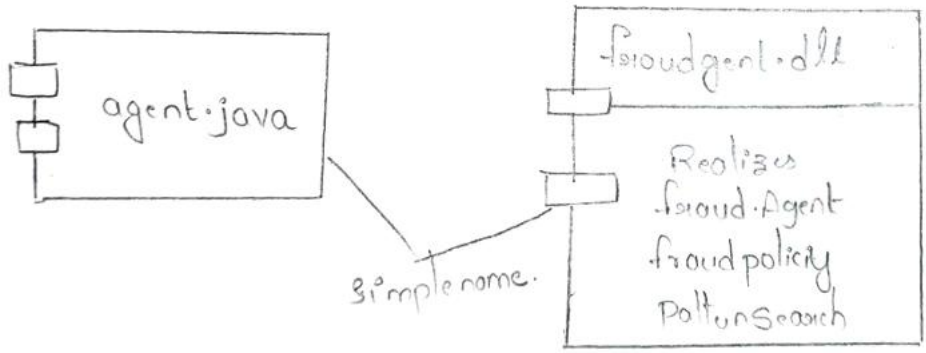
→ Every Component must have a name that distinguishes it from other Components.

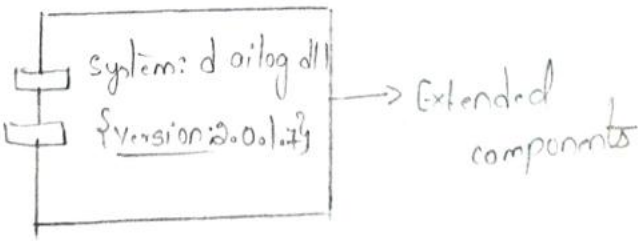
→ A Name is a textual string. the name alone is known as simple name

→ A path name is the Component name prefixed by the name of the package in which the component lives.

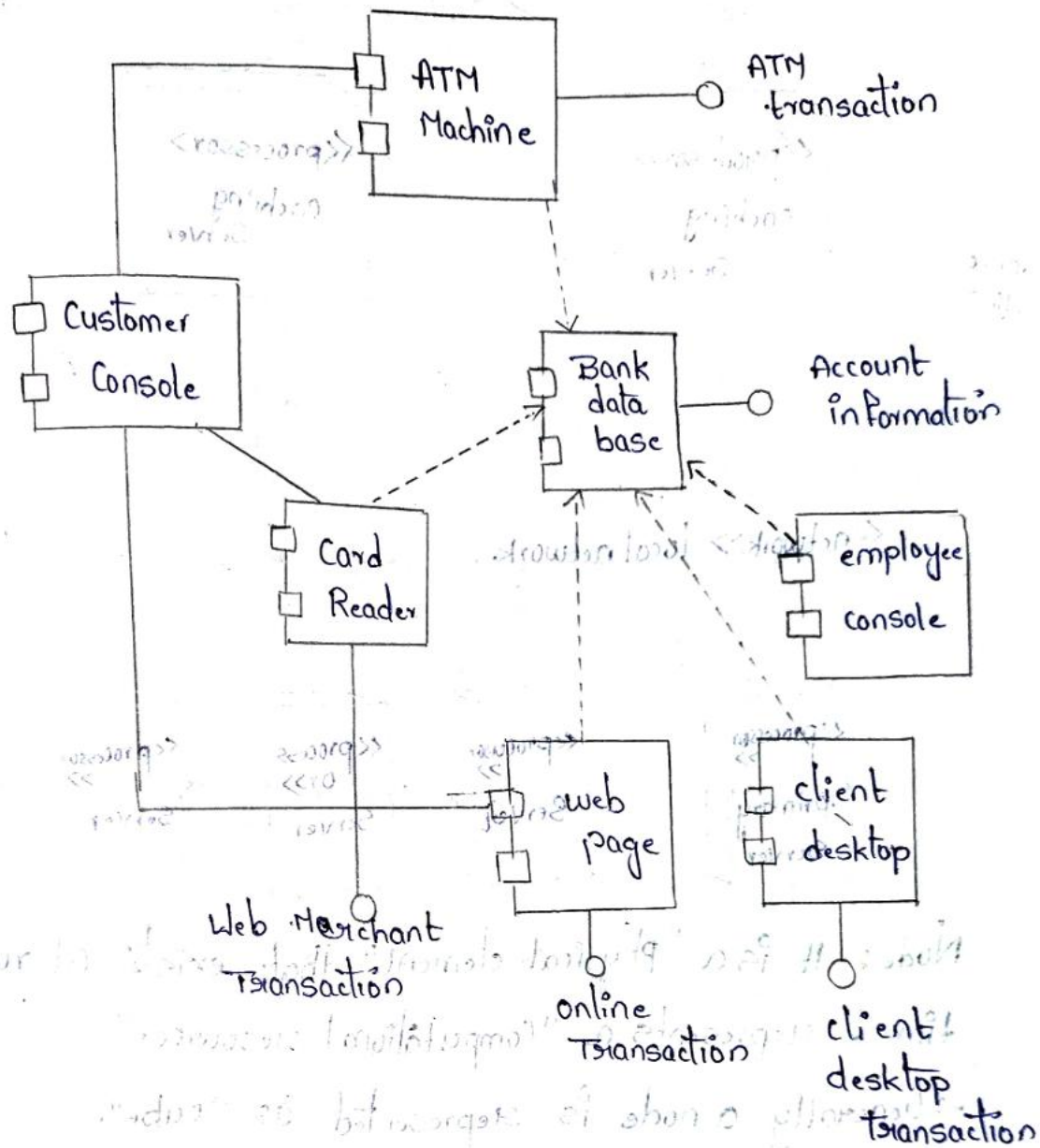
→ You may draw Components adorned with tagged values or with additional Compartments to expose their details.

Examples of Components:-

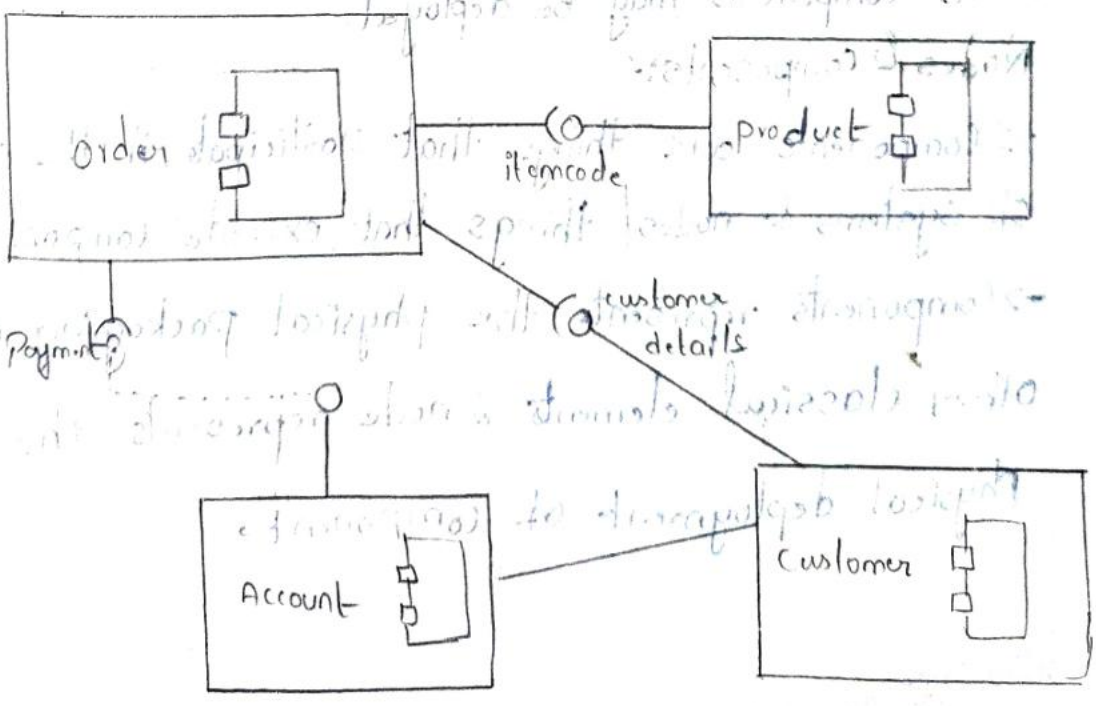




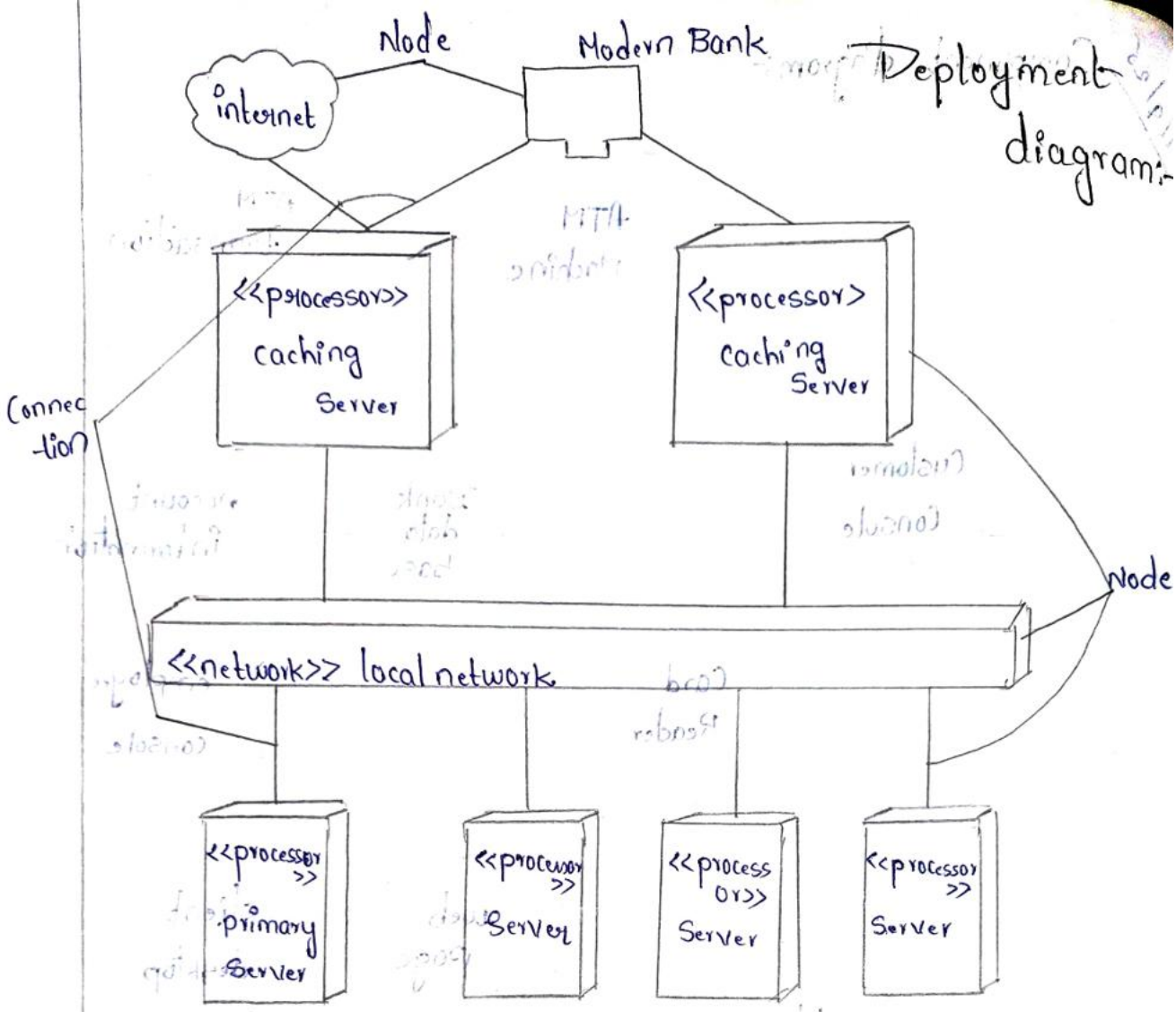
Component diagram:- for Bank transaction



Online shopping system:-



Deployment diagram:-



Node: It is a "physical element" that exists at run time & represents a "computational resource."

→ Generally a node is represented as "cube".

→ A node typically represents a processor / device on which components may be deployed.

Nodes & components:-

→ Components are things that participate in the executio of systems & nodes / things that execute components.

→ Components represents the physical packaging of other classical elements & node represents the physical deployment of component.

Deployment diagram:-

A deployment diagram is a diagram that shows the configuration of runtime processing nodes & the components that live on them.

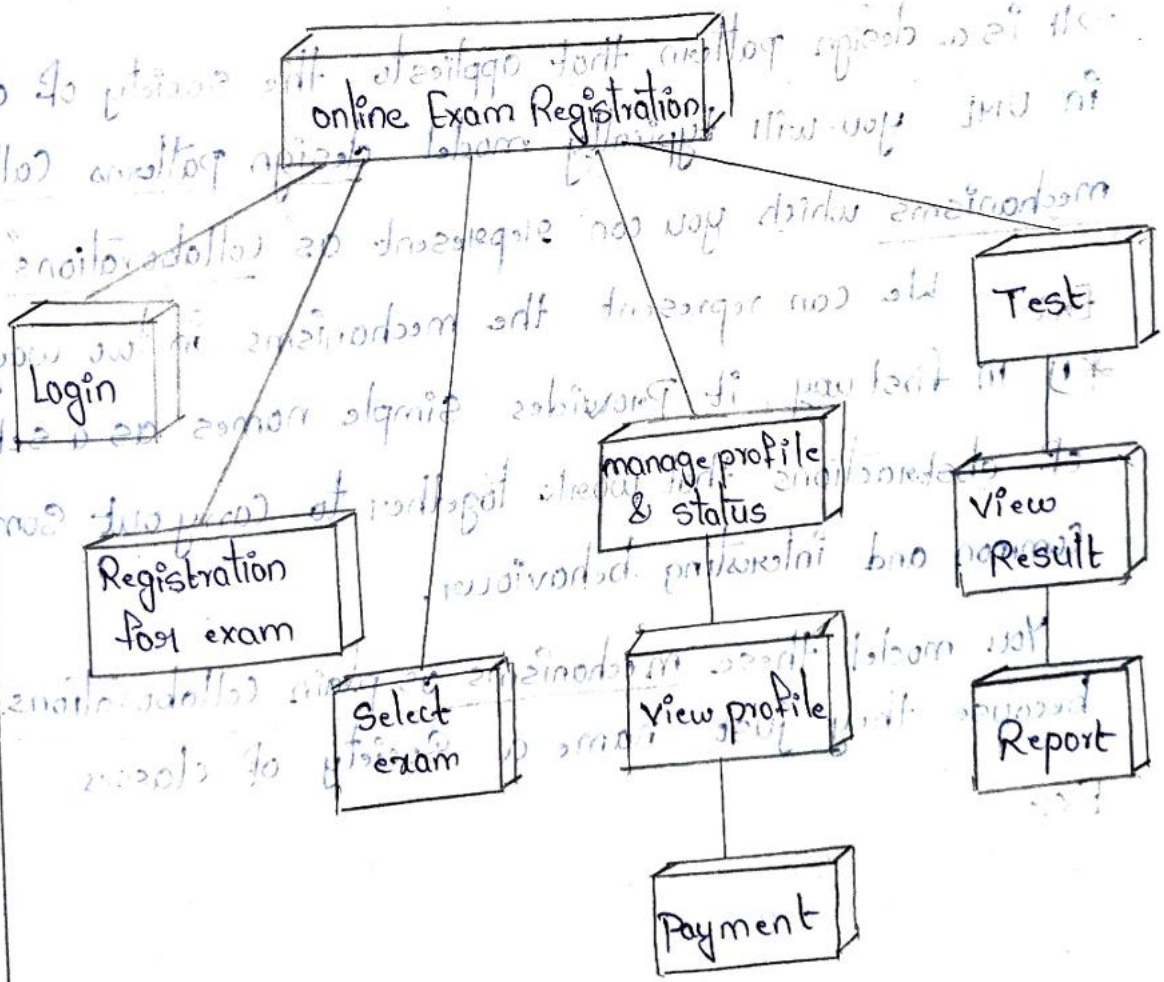
Purpose:- The purpose of deployment diagram is to visualize the "hardware topology" of a system.

-> Describes the hardware components used to deploy the software components.

-> Describes the runtime processing nodes.

-> It consists of nodes, dependency & association relationships.

Deployment diagram for online exam registration :-



24/9/25

Unit - V.

Patterns & frame works :-

- also called as Mechanism
- acts as design pattern
- architectural patterns
- It acts as a reusable component

Patterns :-

It acts as a reusable solution to a commonly occurring problem within a given context.

[or]

→ It Provides a common solution to a common problem in a given context.

→ A well structured systems are full of patterns.

→ We use patterns to specify the structure & behaviour of society of classes.

Mechanisms :-

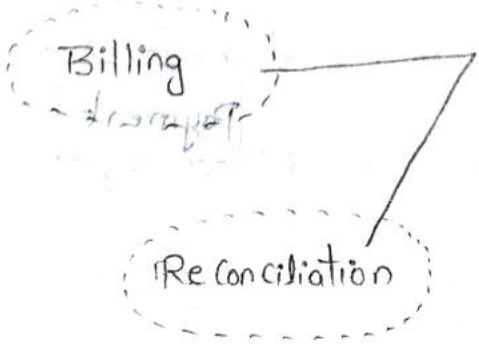
→ It is a design pattern that applies to the society of classes in UML you will typically model design patterns called mechanisms which you can represent as "collaborations".

~~**~~ We can represent the mechanisms in two ways.

*1) In first way, it Provides simple names as a set of abstractions that work together to carry out some common and interesting behaviour.

You model these mechanisms as plain collaborations because they just name a society of classes

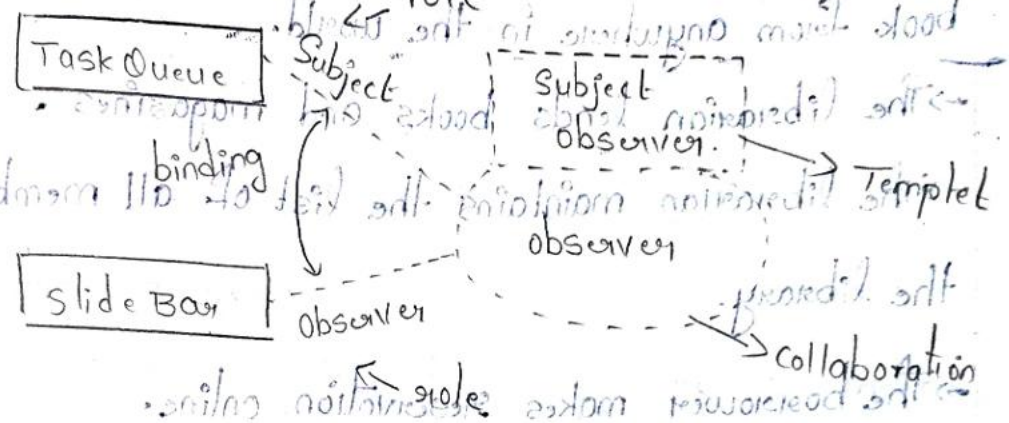
Ex:-



2) In second way, it names a templet for a set of abstractions that work together to carry out some common & interesting behaviour.

You model these mechanisms as parameterise collaboration which are rendered in the UML.

Ex:-



Frame Works:-

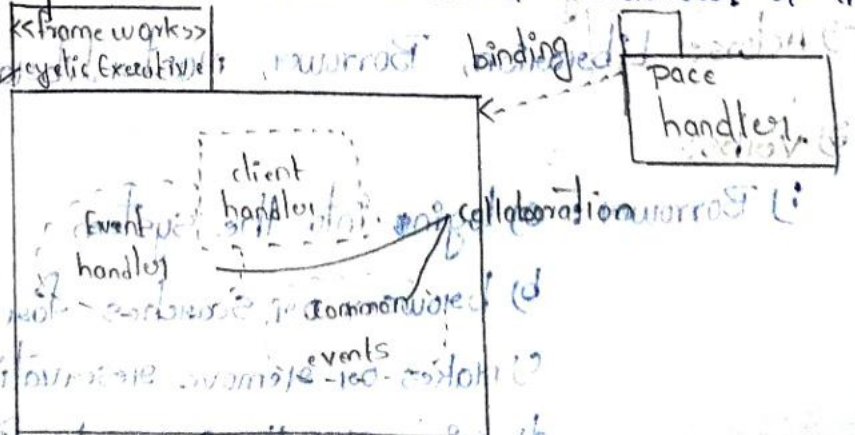
It is typically an architectural pattern that provides an extensible template for applications within a domain.

→ It is bigger than the mechanism & it acts as a collection of classes, components & patterns that provide reusable architectures.

→ similarly, you will typically model architecture patterns as frame works which you can represent as "Stereotyped packages".

Ex:4 → It specifies the structure & behaviour of the entire system.

Ex:-



*Imp

Case Study on Unified library Applications:-

→ The unified library Application emphasizes the online reservation issues and return of books.

→ This system globalizes the present library system by using this application, the members can reserve any book from anywhere in the world.

→ The librarian lends books and magazines.

→ The librarian maintains the list of all members of the library

→ The borrower makes reservation online.

→ The borrower can remove reservation online.

→ librarian issues books to the borrower.

→ librarian calculates dues to be paid by the borrower.

→ Borrower issues or returns books & magazines.

→ librarian places order about the requirements to the master librarian.

→ The librarian updates the system..

→ The master librarian maintains librarians.

Textual Analysis:-

(nouns) 1) Actors: Librarian, Borrower, Master librarian, catalog

2) Verbs:-

i) Borrower: a) logins into the system

b) browser or searches for book or magazine

c) Makes or remove reservation.

d) View results & reports from the

Unified library application system.

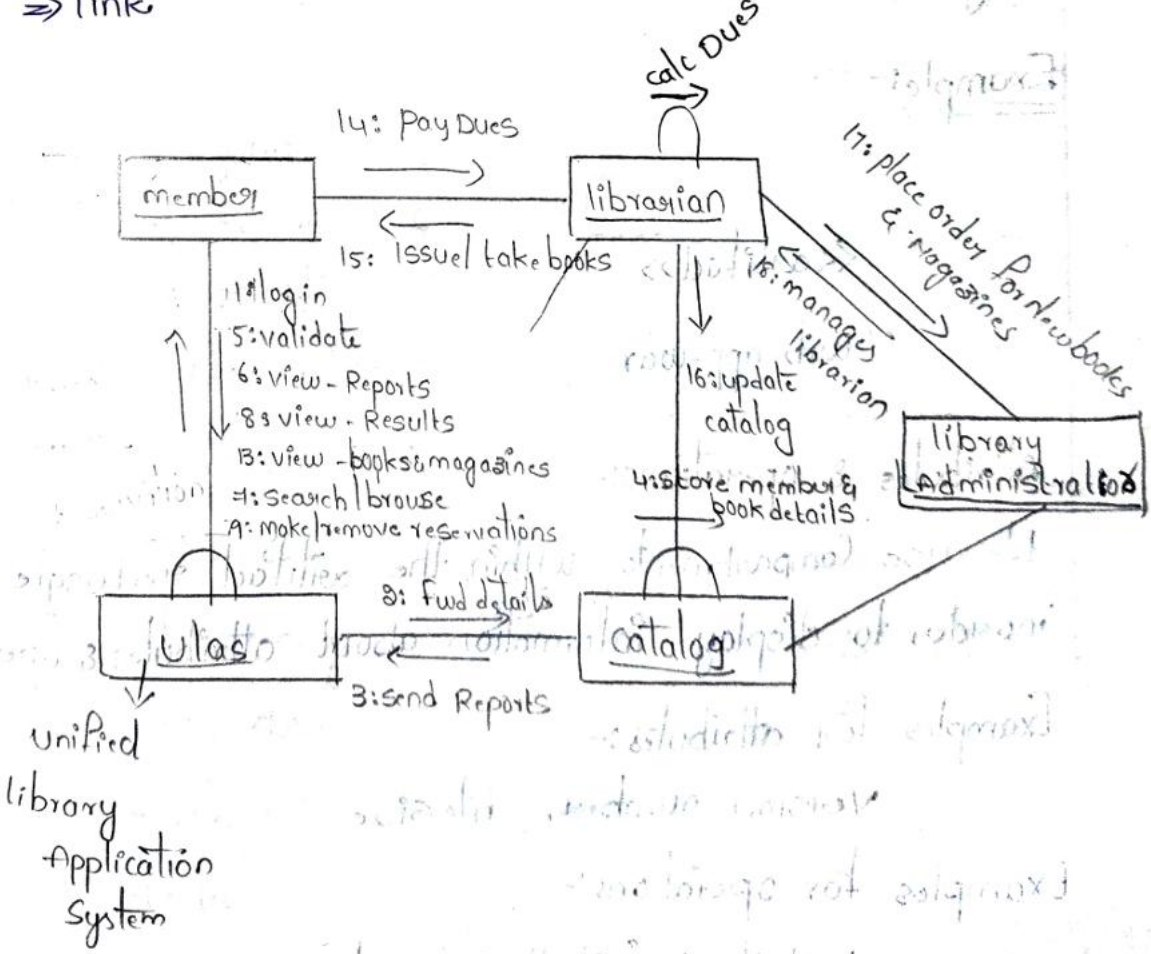
ii) Librarian: a) Manages & validates members.
b) View reports from the system.
c) Issues books
d) Takes books
e) calculates due
f) places order to master librarian.
g) Maintain links of books & magazines.

iii) Master Librarian:-
a) Maintains all other librarians.

Collaboration diagram:-

It displays object interactions, organised around objects and their links to another object.

→ The components are
⇒ Actor
⇒ object &
⇒ link



25/9/25 Artifacts Diagrams:-

Artifact :-

It is a classifier that represents some physical entity that means a piece of information that is used or produced by a software development process or by deployment & operation of the system.

Examples for the artifacts are executable files, libraries, Software components, documents & databases.

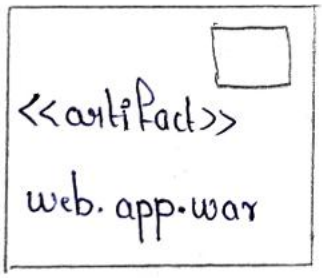
Node :-

It acts as a computational resource.

Examples:- Server or device where artifacts are deployed.

→ An artifact is typically depected as rectangle with the stereotype <<artifact>> inside. & it may also include an icon resembling a document in the upper right corner.

Example:-



Attributes & operations:-

We use compartments within the artifact rectangle inorder to display information about attributes & operations

Examples for attributes:-

Version number, file size.

Examples for operations:-

Install, uninstall, upgrade.

Naming:-

Artifact has a unique name that describes the physical entity it represents.

→ Artifacts are mainly used in the deployment diagrams to show how Software Components (Artifacts) are deployed onto physical (hardware Software execution) nodes.

→ Artifacts are deployed onto nodes indicating which physical files are data resides on which hardware or Software environment.

→ A manifestation relationship represented by a dashed line with an open arrow and the stereotype <<manifest>> links components or classes to the artifacts that implement them.

Relationships:-

Artifacts can have Association or dependencies with other artifacts or model elements with in a diagram.

Artifact diagram:-

→ It is an structural diagram used to model the physical implementation of Software Components.

→ It shows how artifacts [like Source code, files, binary files, executable files, documents etc]. are physically deployed on nodes [Servers, devices, execution environment)

→ It is often using deployment & implementation phases of Software development.

The Purpose of Artifact diagrams:-

→ It illustrates the deployment of Software Components.

→ It visualizes the structure of software artifacts in a system.

→ It shows the relationship blw nodes & artifacts.