Model- Unit-III UML

A model captures aspects important for some application while abstracting (omitting) the rest.

→ A model in the context of software development can be graphical, textual, mathematical or program code based.

→ Models are useful in documenting the designs and analysis of the result.

→ Graphical models are very popular because they are easy to understand and construct.

→ UML is the primary graphical modeling tool.

Need for a model:-

→ model helps to manage complexity

→ It is useful in the following

* Analysis
* specification
* Code generation
* Design

→ It is used to visualize and understand the working of a system.

Unified modeling language (UML):-

→ UML is used to visualize, specify, construct and document the artifacts of a software system.

→ It provides set of notations (rectangles, lines ellipses etc) to create visual model of the system.

→ UML has its own syntax (symbols & rules) and semantic (meaning of symbols & sentences)

UML diagrams:-

→ UML can be used to construct nine different types of diagrams to capture five different views of a system.

→ Nine different types of UML diagrams are divided into two types

1. Static Diagram
2. Dynamic Diagram.

Static diagram:-

1) Usecase diagrams → User view
2) Class diagrams → Structural view/Design view
3) Object diagrams
4) Component diagrams → Implementation view
5) Deployment diagrams → Environmental view

Dynamic diagrams:-

6) Sequence diagrams
7) Collaboration diagrams } Behavioral/process view
8) Statechart diagrams
9) Activity diagrams

UML diagrams can capture the following five views of a system

1. User's view
2. Structural view or Design view.
3. Behavioral view or process view.
4. Implementation view
5. Environmental view.

User's View:-

→ This view defines the functionalities made available by the system to its user.

→ It captures the external users view of the system in terms of functionalities offered by the system.

→ The user view is a black-box view of the system where the internal structure, dynamic behaviour and implementation, are not visible.

→ The user's view can be considered as the central view and all other views are expected to confirm to this view.

It defines the kinds of objects (classes) important to the understanding of the working of a system and to its implementation.

→ It also captures the relationship among the classes (objects) It is also called the static model, since the structure of a system does not change with time

Behavioral view:-

The behavioral view captures how objects interact with each other to realize the system behavior. It is also called dynamic model. Since the behavior of the system will change with time.

Implementation view:-

This view captures the important components of the system and their dependencies

Environmental view:-

Environmental view models how the different components are implemented on different pieces of hardware.

Usecase Model :-

The usecase model for any system consist of a set of "use cases".

→ Usecases represent the different ways in which a system can be used by the users.

→ Usecases partition the system behaviour into transactions, such that each transaction performs some useful action from the user's point of view.

→ Each transaction, to complete, may involve multiple message exchanges between the user and the system.

→ A use case consists of one main line sequence & several alternate sequences.

→ The "mainline sequence" is the most frequently occuring sequence of interaction.

→ The variations to the mainline sequence occurs when some specific conditions hold.

→ Such type of sequences after variations are called "alternate sequences"

→ The mainline sequence & each of the variations are called "scenarios" of the usecase.

Representation of Usecases:-

→ Usecase is represented by an ellipse with the name inside the ellipse. (LIS)

→ All the usecases of system are enclosed within a rectangle which represents the system boundary

→ The name of the system appears inside the rectangle 

→ Users and external systems can be represented by stick person icons. ()

→ Which When a stick person icon represents an external system, it is annotated by the stereotype ≪external system≫.

→ A line connecting an actor & usecase is called Communication relationship.

Text description:-

→ Each ellipse in a usecase diagram, by itself conveys very little information, but not the clear idea about the usecase.

Therefore usecase diagram should be accompanied by a text description.

The following are some of the information which may be included in a usecase text description in addition to mainline sequence and the alternate scenarios.

* Contact persons:-
This section lists of personnel

* Actors:- It gives some information about actors using a usecase which may help the implementation of the UseCase.

Precondition:- The preconditions would describe the State of the system before the usecase execution Starts.

post conditions:- This captures the state of the system after the use case has successfully completed.

Non-functional requirements:-
This Contains the important Constraints for the design and implementation, such as platform and environment conditions, qualitative statements etc.

Exceptions, error situations -

This contains only the errors such as lack of user's access rights, invalid entry etc.

Sample dialog:- It illustrates the usecase.

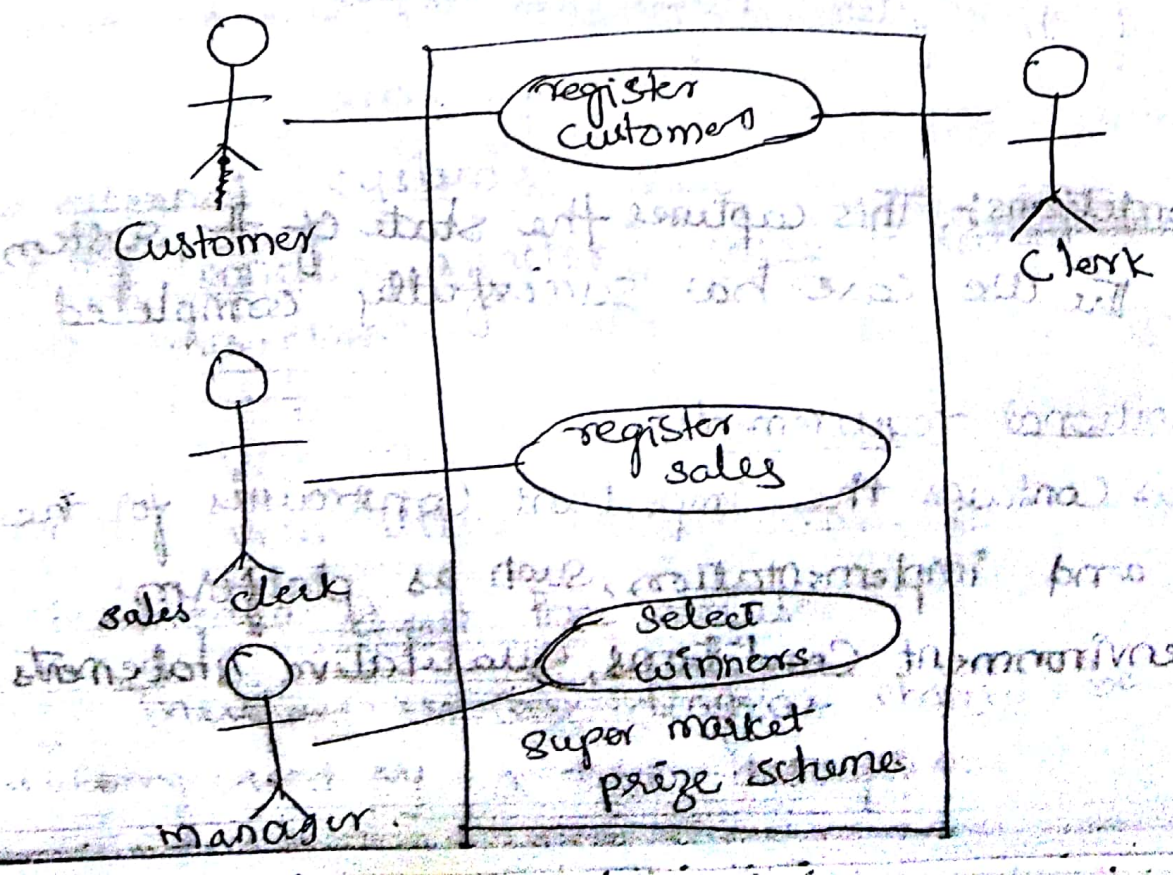
Specific user interface requirements:-

It contain specific requirements for the user interface of the usecase. For ex:- It contains forms, screenshots, interaction style etc.

Document references:-

It contains references to documents which may be useful to understand the system operation.

Usecase model for supermarket prize scheme:-



Factoring Usecases:-

Factoring usecases required under two situations

i) Complex usecases need to be factored into Simplex use cases

ii) Usecases need to be factored whenever there is Common behavior across different usecases

UML offers three factoring mechanisms

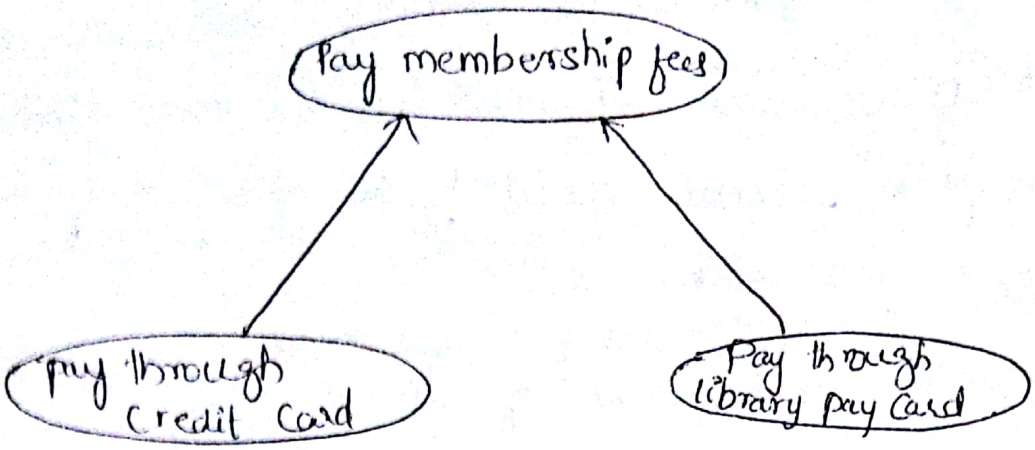
1) Generalisation
2) Includes
3) Extends

Generalisation:-

→ Generalisation can be used when there is a one usecase that is similar to another but does something slightly different.

→ The child usecase inherits the behaviour and meaning of the present usecase.

→ The base & derived usecases are separate use cases and should have separate text descriptions



Representation of usecase generalization.

Includes :-

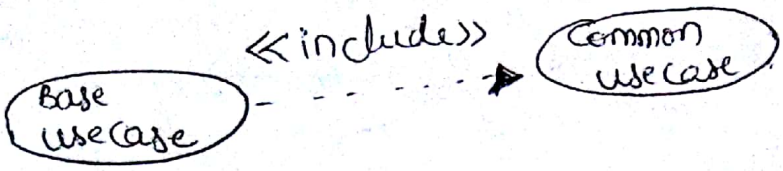
→ The include relationship implies one usecase includes the behaviour of another use case in its sequence of events and actions.

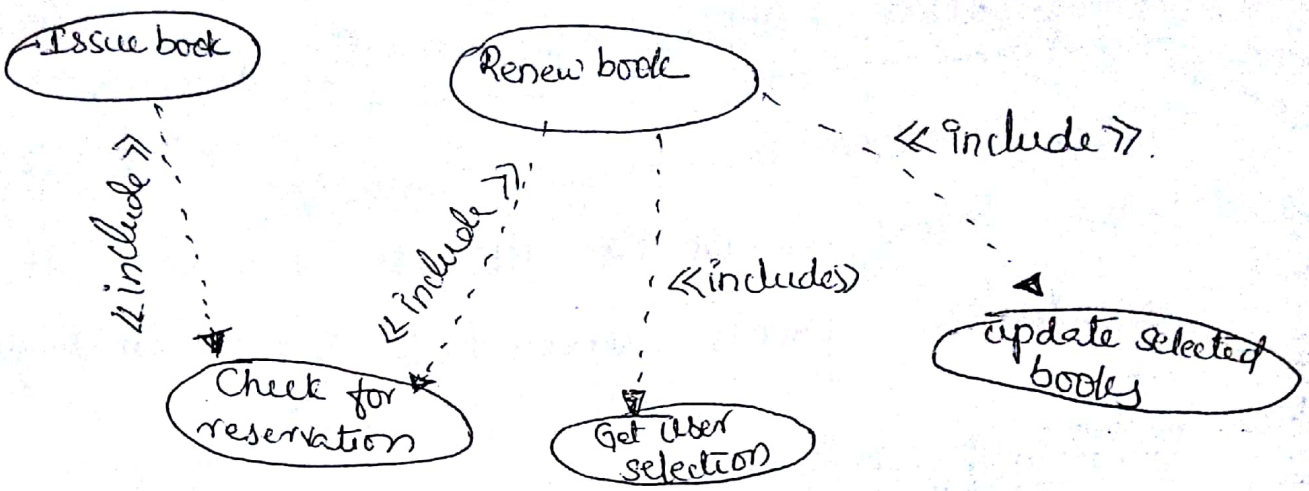
→ It is appropriate when a chunk of behavior that is similar across a no. of usecases.

→ It will help in not repeating the specification & implementation across different usecases.

→ The include relationship explores the issue of reuse.

→ It is represented using a predefined stereotype ≪ include ≫.



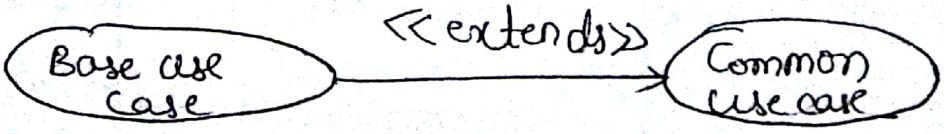


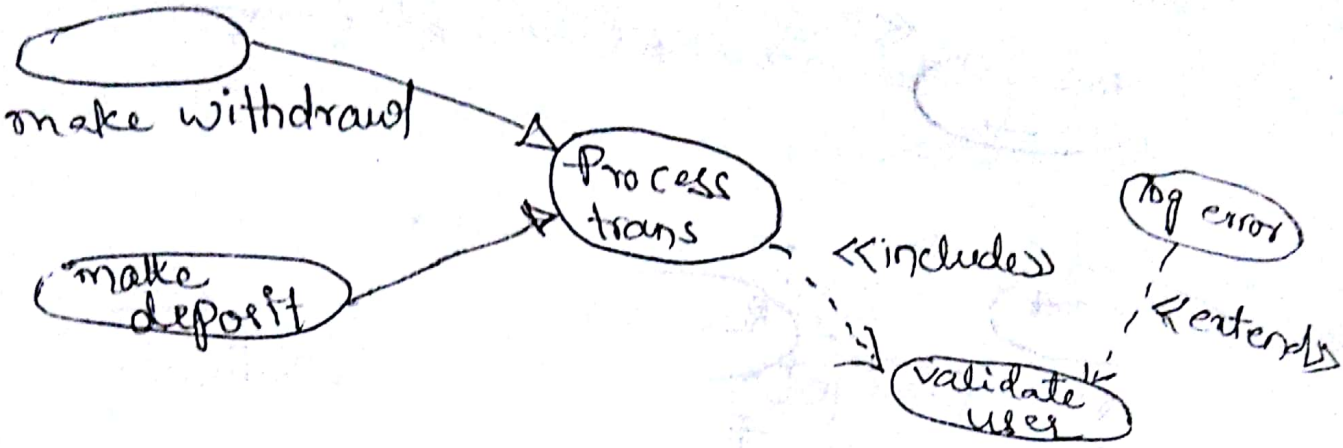
Extends:-

→ The main idea behind the extends relationship among the usecases is that it allows to show optional system behavior.

→ The extends relationship is similar to generalization

→ It is used to capture alternate paths or scenarios



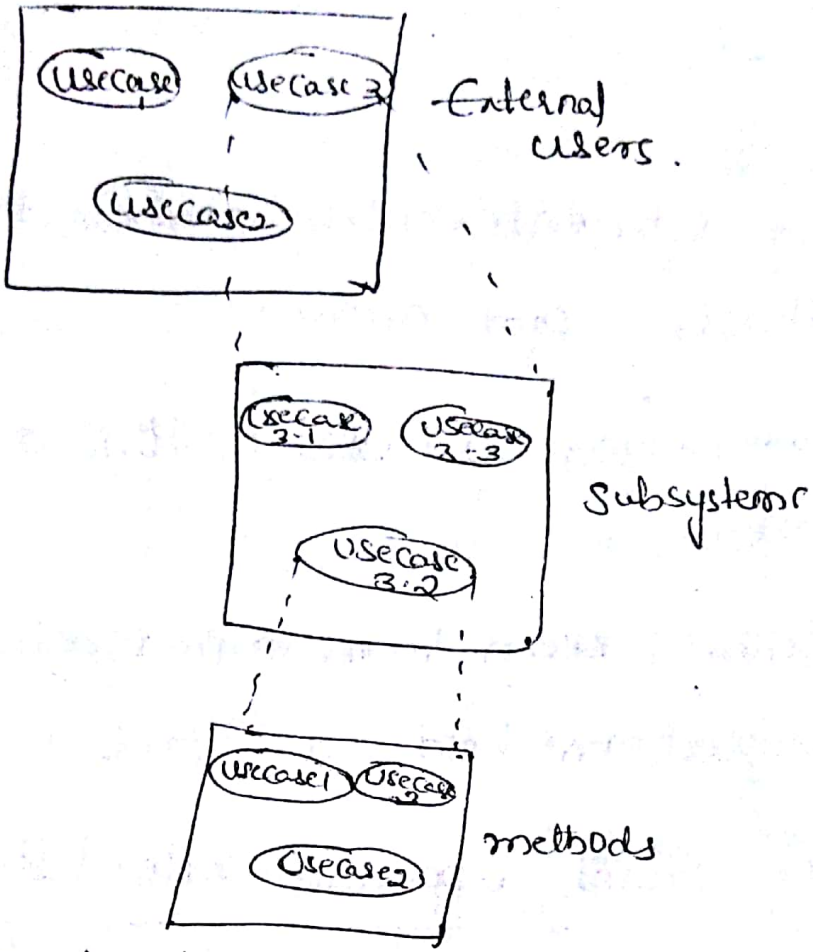


Organization of Usecases:-

→ When the use cases are factored, they are organized hierarchically

→ Top level usecases are super-ordinate to the refined usecases. The refined use cases are sub-ordinate to the top-level usecases.

→ Complex usecases should be decomposed and organized in a hierarchy. its not necessary to decompose simple use cases.



Hierarchical organization of usecases.

Class diagrams:-

→ A Class diagram describes the structure of a system. It shows how a system is structured rather than how it behaves.

→ The static structure of a System comprises of a no of Class diagram and their dependencies.

→ The main Contents of a class diagram are classes and their relationships as generalization, aggregation, association and various kinds of dependencies.

Classes:-

→ The classes represent entities with common features i-e attributes and operations

→ Classes are represented as solid outline rectangles with compartments

→ Name should be written in the center of the upper compartment in bold face.

→ It Name starts with uppercase letter and.

→ The classes have optional attributes & operations Compartments.

Attributes:-

→ An attribute is a named property of a class. It represents the kind of data that an object might contain.

→ Attributes are listed with their names, and may optionally contain specification of their type, an initial value, and constraints.

→ The first letter of a class name is a small letter.

bookName : String.
↓ attribute Name ↓ type.

Visibility :-

To specify the visibility of a class member the following notations must be placed before the member's name.

+	public
-	private
#	protected
/	derived
~	package

Ex:- → balance : double : 110.00

Operations :-

The Operation names are always begin with a lower case letter

→ The operation may have a "return type" Consisting a single return type expression.

Ex:- issueBook (in bookName) : Boolean.

Sig:- Visibility name (parameters) : returntype).

Comments :-

* Represented as a folded note, attached to the appropriate class/method by a dashed line

Eg:

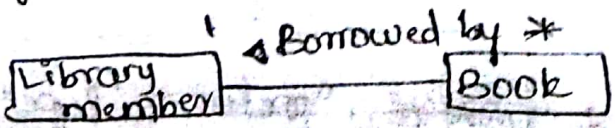
Association :-

→ Associations are needed to enable objects to Communicate with each other.

→ An association relation between two objects is called object connection or link.

→ Association btw two classes is represented by drawing a straight line btw the concerned classes.

→



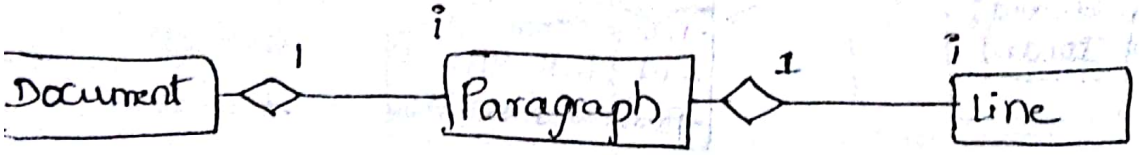
The arrowhead may be placed to indicate the reading direction of the association.

Aggregation :-

→ Aggregation is a special type of association where the involved classes represent a whole-part relationship.

→ When an instance of one object contains instances of some other objects, then aggregation relationship exists between Composite object and Component object.

→ Aggregation is represented by the diamond symbol at the Composite end of a relationship



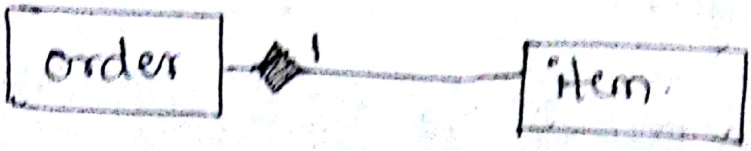
Composition :-

→ Composition ~~is a~~ shows strong ownership stricter ~~form~~ of aggregation, in which the parts are existence-dependent on the whole.

→ The life of the parts closely ties to the life of the whole.

→ When the whole is created, the parts are created and when the whole is destroyed, parts are destroyed.

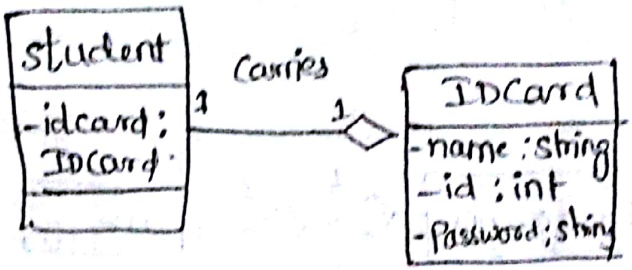
→ This relationship is represented as a filled diamond drawn at the Composite-end.



Multiplicity of Associations:

1) One-to-one :-

Each student must carry exactly one ID card



* ⇒ 0, 1, or more

1 = exactly 1

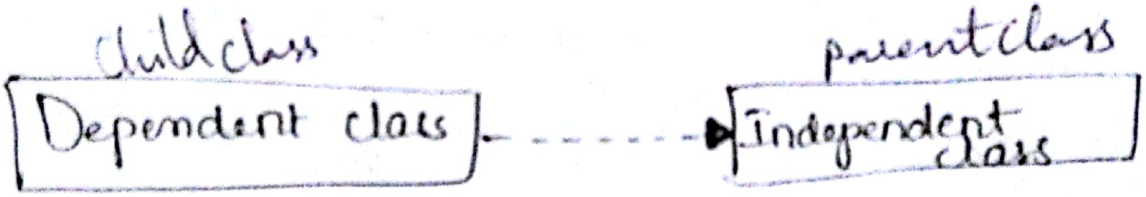
2..4 ⇒ between 2 and 4

3.... * ⇒ 3 or more.

Dependency:-

Dependency is a weaker form of bond that indicates that one class depends on another because it uses it at some point in time

→ A dependency relationship is shown as a dotted arrow that is drawn from the dependent class to the independent class.



eg:- Employee class → Department class

Generalization:- Square class ⇢ Shape

Generalization/inheritance involve the inheriting the properties of superclass by a subclass.

→ Line/arrow style differ based on type of parent class.

→ If the parent class is a normal class, then the arrow line should be a solid line with black arrow head

→ If the parent class is a abstract class then the line should be a solid line with white arrow head.

→ If the parent class is an interface then the line should be a dashed line with ~~with~~ white arrow head.

Interaction Diagrams:-

* Interaction diagrams illustrate how objects interact via messages in order to fulfill certain tasks.

* There are two kinds of interaction diagrams.
  - → Collaboration diagrams
  - → Sequence diagrams.

When & how to use interaction diagrams:-

Identify the system events that are implied by the usecases

→ make atleast one interaction diagram for each system event

Notations of interaction diagram:-

→ class → Person

→ Instance of class (object w/o name) : Person

→ Named instance of a class michael : Person

→ Named object only → michael

Collaboration diagrams:-

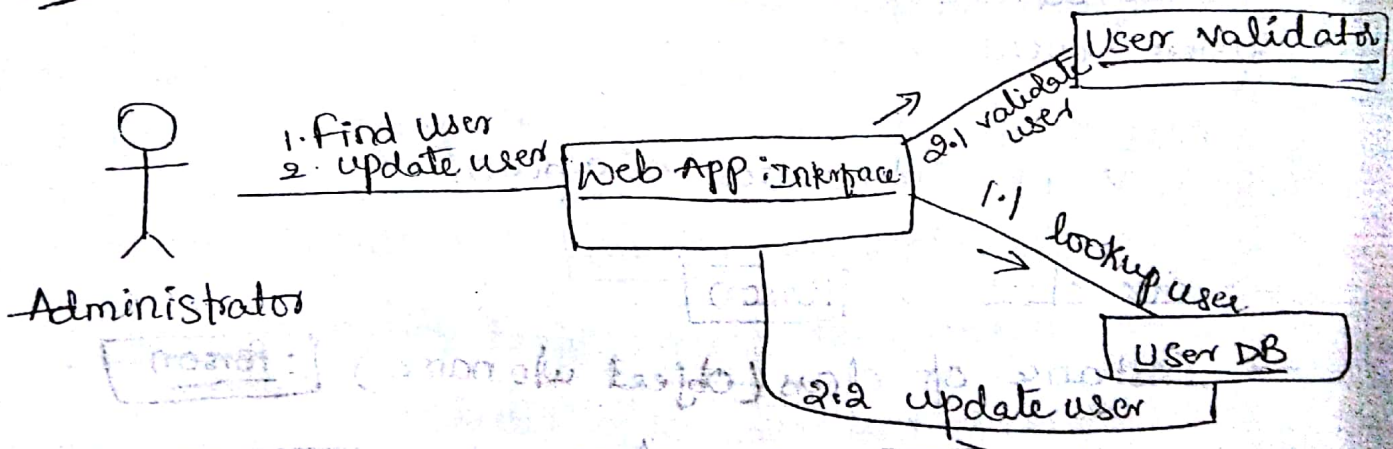
→ A collaboration diagram shows the relationship between objects and the order of messages passed between them.

→ It shows both structural and behavioural aspects.

→ The objects are listed as icon and arrows indicate the messages being passed b/w them.

→ The numbers next to the messages are called Sequence numbers and as the name suggests they show the sequence of messages as they are passed between the objects.

Ex:-



Strength of Collaboration diagram:-

These are simple and easy to understand

Weaknesses:-

→ They are good at describing behavior, but they do not define it

→ They do not show the iterations.

Sequence diagrams:-

→ A sequence diagram shows relation between them objects.

→ It shows object interactions arranged in time sequence.

→ It should be read from left to right and from top to bottom.

→ At the top of the diagram are names of objects that interact with each other.

→ Each object has a lifeline. Time flows from top to bottom

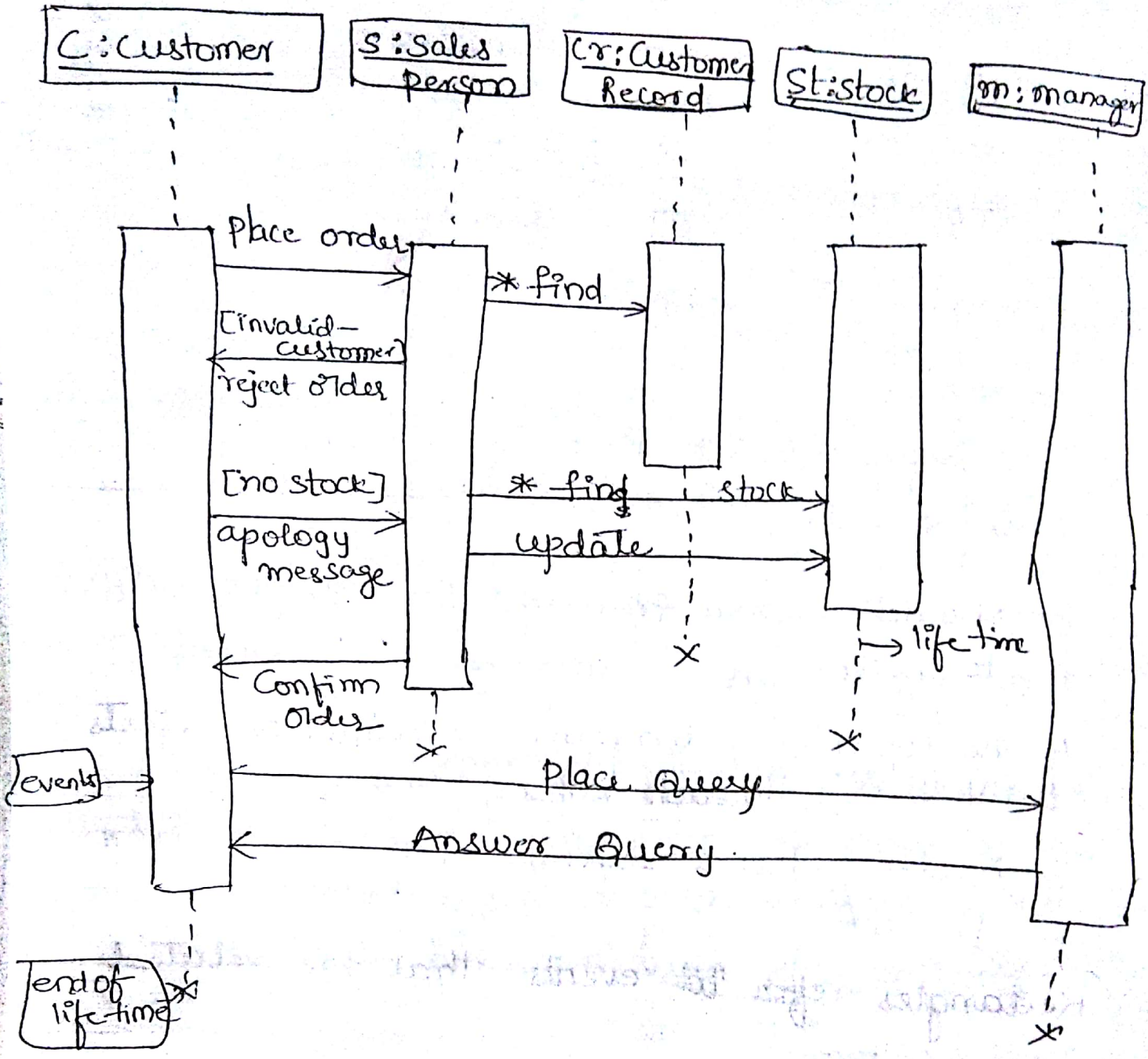
→ Rectangles refer to events that are related to each other.

→ Arrows indicate messages that are sent from one object to another object.

→ Feedback arrow shows that a value is returned

→ X at the end of lifeline shows that the object to exit.

Exp:-



Advantages :-

→ Simple and easy.

→ It shows time ordering of messages.

Disadvantages :-

→ forced to extend right when adding new objects, Consumes a lot of space at todo

Types of rel"ships in UML:-

① Dependencies
② Generalization
③ Associations → Aggregation

Dependencies:-

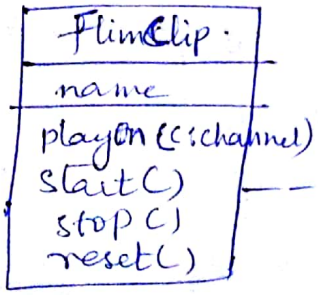
→ It's a rel"ship that states that a change in specification of one class may affect another class; but reverse is not true.

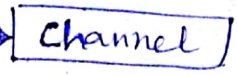
→ It is a weaker form of bond that indicates that one class depends on another because it uses it at any pt of time.

→ It is represented as a dotted arrow line (dashed directed line).

Syntax:-

Eg①





Eg②:-

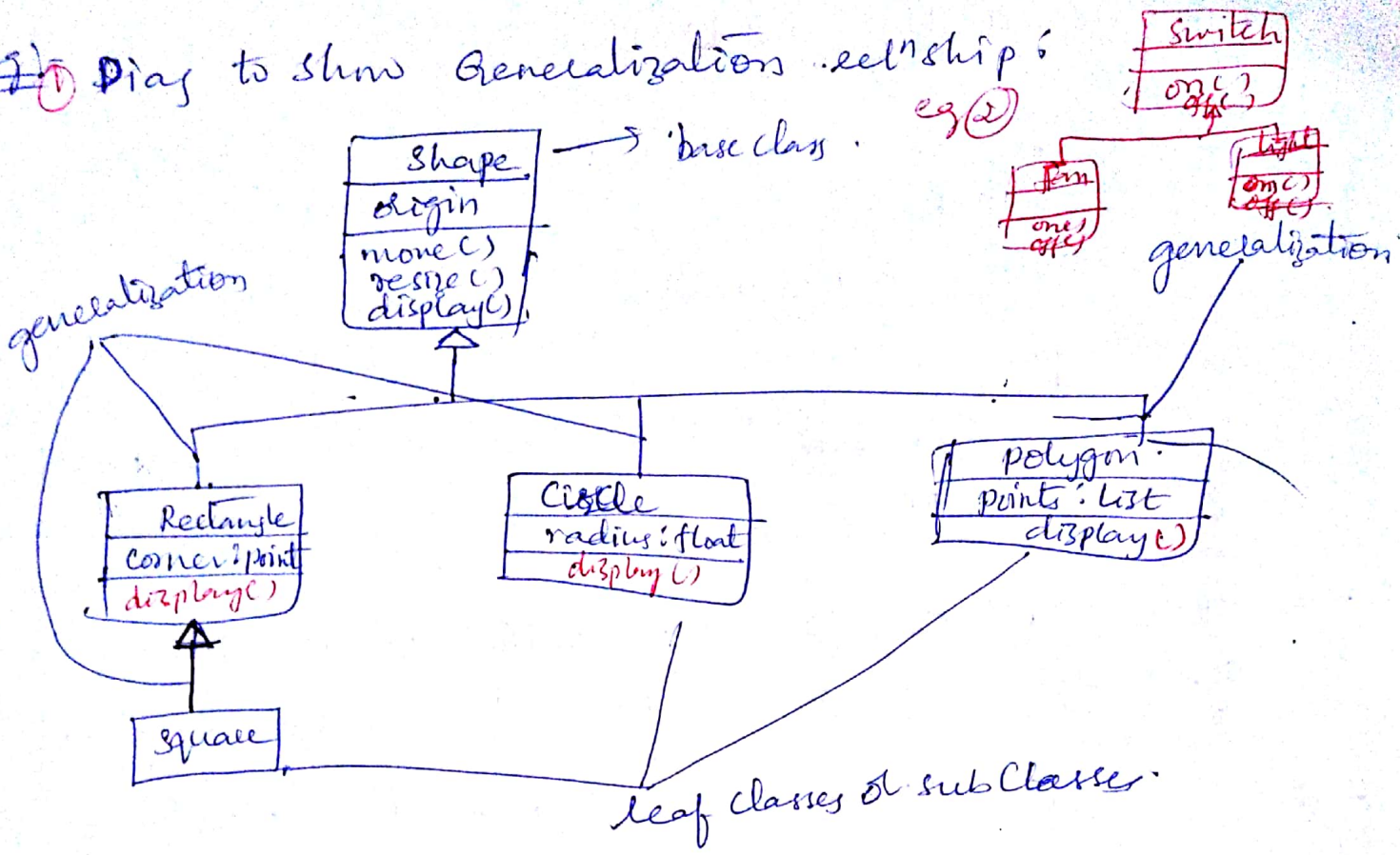
Generalization (or) Inheritance:

→ It involves inheriting the properties of 'Super class (parent class) by a subclass (child class) especially their attributes & opers.

→ It is sometimes called as "is-a-kind-of" rel"ship.
• ie; one thing is-a-kind-of a more general thing.

→ we use generalization rel"ship among classes & interfaces to show inheritance relships. we can also create generalizn among other things like packages.

Eg: ① Diag to show Generalization rel"ship:



Syntax: SubClass ——▷ ParentClass.

eg① Rectangle ——▷ Shape. ; Employee ——▷ Department

Note → line/arrow style differs based on type of parent class.

→ If the parent class is a normal class, then the line should be a solid line with blank/white arrow head.

i.e., ——▷

→ If the parent class is an abstract class then the line should be a solid line with white arrow head.

→ If the parent class is an interface then the line should be a dashed line with white arrow head.

- - - - - ▷

(2)

Association:- It's a structural relnship that specifies that objects of 1 class are connected to objects of another 1 or more classes. An assocn bln 2 classes is called binary assocn. An assocn that connects more than 2 classes is called n-ary assocn.

→ It is a structural relnship that describe a set of links, which establishes a connection among objects.

→ Aggregation is a special kind of association, representing a structural relnship bln a whole (bigger one) & its parts (small elements).

→ Graphically, it is shown as a solid line, possibly directed, occasionally including a label, & often containing other adornments such as multiplicity, & rolenames.

→ It is shown as a solid line connecting the same or diff classes.

Eg:- Org

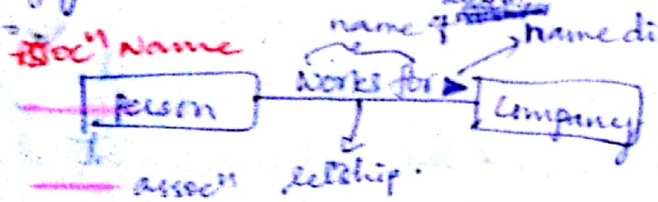
→ Arrowhead may be placed to indicate the reading direction of the assocn.

+ The four adornments that apply to associations are:

① Name ② Role ③ multiplicity ④ Aggregation

Name:- An assocn can have a name which describes the nature of the relnship so that there is no ambiguity about its meaning, a direction to the name is given by a direction triangle that points in the direction you intend to read the name.

Eg:- Assoc'n Name

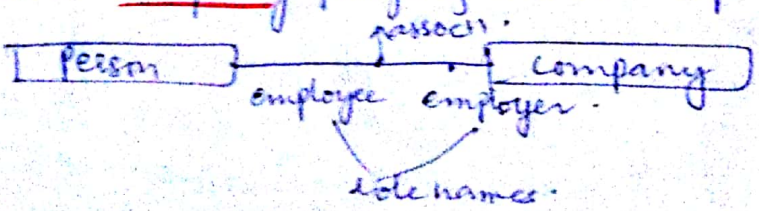


assocn relnship.

Role:- When a class participates in an assocn, it has a specific role that it plays in that relnships, i.e. we can name the role of a class plays in an association.

→ It shows the face of the class at the near end of the assocn presents to the class at the other end of the assocn.

Eg:- A person playing the role of employee is associated with a company playing the role of employer.

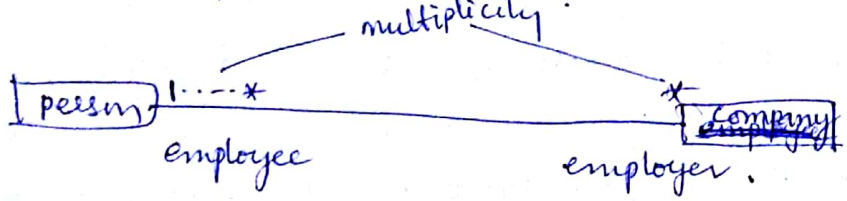


Multiplicity :-

→ An assoc represents a structural relnship among objects. In many modeling situations, it is impt to show that how many objects may be connected across an instance of an assoc.

→ This "how many" is called the multiplicity of an association's role & is written as an expression that evaluates to range of values or an explicit value.

Eg :-



→ when we state multiplicity at one of an assoc, we are specifying that for each obj of the class at the opposite end, there must be that many objects at the near end.

→ we can show the multiplicity as :

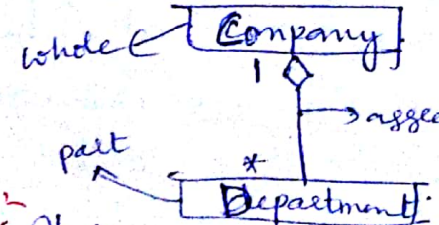
* → 0, 1, or more.
1 → exactly 1
2...4 → btn 2 & 4.
3...* → 3 or more, etc.

Aggregation :-

→ To model a "whole-part" relnship, in which one class represents a larger thing (whole), which consists of smaller things (parts). This kind of relnship is called aggregation, which represents a "has-a" relnship, which means that an obj of the whole has object of the part.

→ It is a special kind of assoc & is represented by an assoc with an open diamond at the whole end.

Eg :-



Composition :-

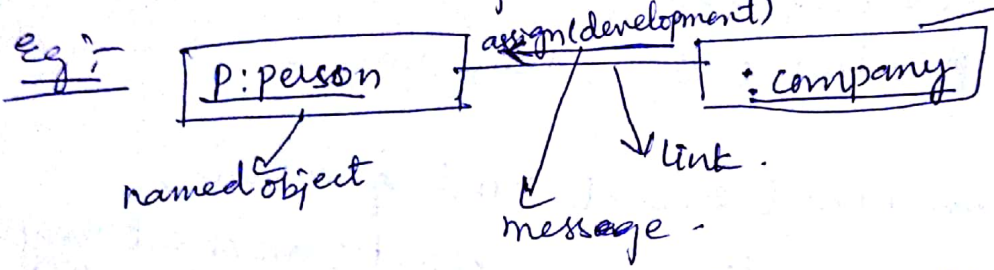
It is a strict form of aggregation with strong ownership in which the parts are existence-dependent on the whole. i.e life of the parts closely ties to the life of the whole.

→ when the whole is created, the parts are created & when the whole is destroyed, parts are destroyed. It is represented as a filled diamond drawn at the composite end.

Explain about links & messages in UML?

→ A link is a semantic connection among objects. In general, a link is an instance of an association; wherever a class has an association to another class, there may be a link bln the instances of the 2 classes & wherever there is a link bln 2 objects, one obj can send a message to the other object.

Eg:-



→ A link specifies a path along which one obj can dispatch a message to another (or the same) object. most of the time, it is sufficient to specify that such a path exists. we need to be more precise about how that path exists, & we can adorn/decorate 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 obj is visible because it is the dispatcher of the operation.
3) Global: specifies that the corresponding obj is visible because it is in an enclosing scope.
4) Local: specifies that the corresponding object is visible becoz it is in a local scope.
5) Parameter: specifies that the corresponding obj is visible bcz it is a parameter.

→ As an instance of an association, a link may be rendered with most of the adornments appropriate to associations, such as name, association role name, navigation & aggregation.

Messages:-

→ A message is the speech of a comm among objects that should convey information with the expectation that activity ~~to be desired~~ happen or occur.

→ The receipt of a message instance may be considered an instance of an event.

→ When we pass a message, the action that results is an executable statement, that forms an abstraction of a computational procedure. An action may result in a change in state

→ In UML, we can model several kinds of actions:

① call : Invokes an operation on an object; an object may send a message to itself, resulting in the local invocation of an operation.

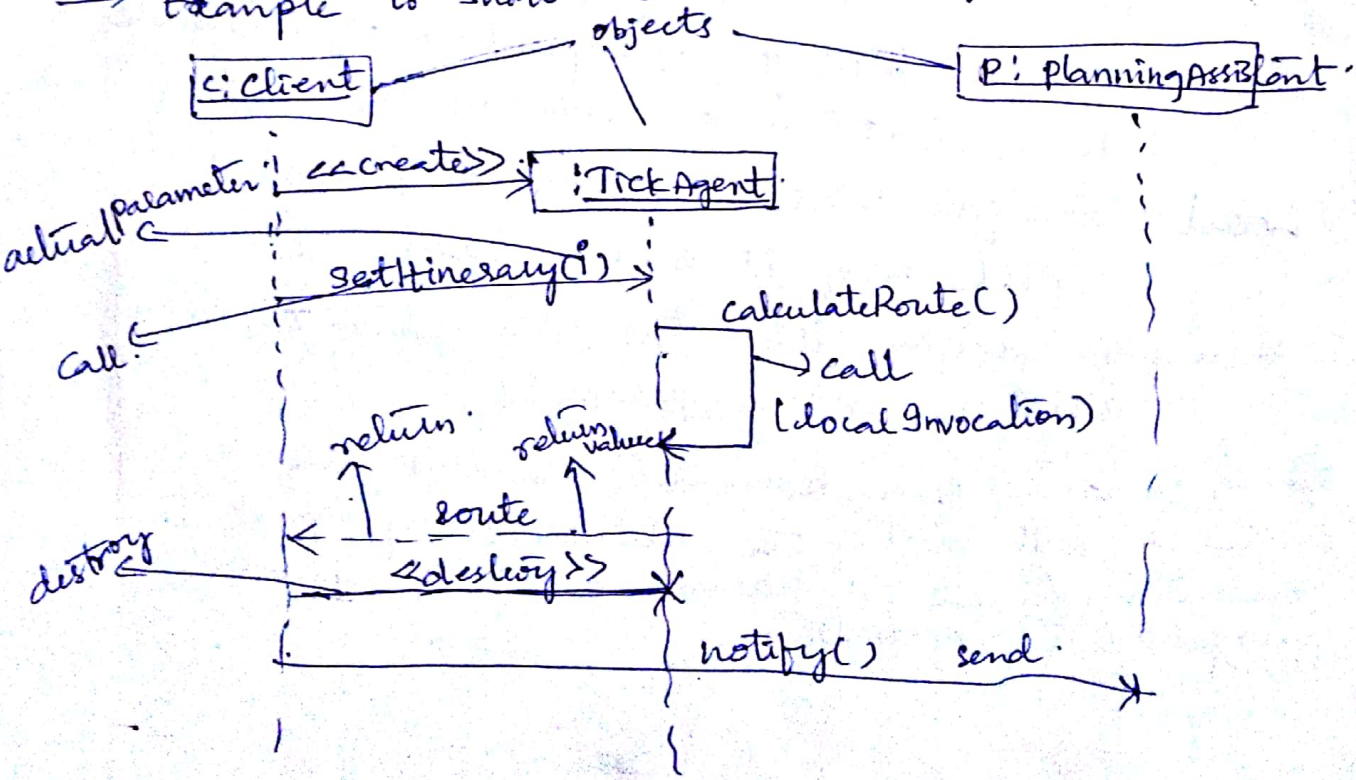
② Return : Returns a value to the caller.

③ send : Sends a signal to the object.

④ create : creates an object.

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

→ Example to show the above kinds of messages. as shown in fig,



→ The most common kind of message we model / use is the call (message), in which one object invokes an operation of another (or same) object.

→ An object can't just call any random operation.

→ If an object, such as c in the example above, calls the operation setItinerary on an instance of the class TicketAgent, the operation setItinerary must not only be defined for the class TicketAgent (i.e, it must be declared in the class TicketAgent or one of its parents), it must also be visible to the caller c.