URBAN EDGE 30/1/19

MODULE 1

Object

It means a combination of data & logic that represents some real world entity. Objects are defined in terms of set of attributes / properties & set of methods / procedures.
Properties describe the state of an object. Methods describe it's behaviour. That means the set of things normally an object can do.
The fundamental difference btw object oriented systems & the traditional methodology lies in the way in which they approach the problem. Most traditional methodologies are either algorithmic-centric or data-centric. In algorithmic-centric you think of an algorithm that can accomplish the task, then build data structures for that algorithm to use.
In data-centric methodology, you think how to structure the data, then build the algorithm around that structure.
In object-oriented system, the algorithm & data structures are packed together as an object. The attributes & methods are equal & inseparable parts of the object.

Class

Classes are used to distinguish 1 type of objects from another. In object-oriented system, a class is a set of objects that share a common structure & a common behaviour. A single object is simply an instance of a class. A class is a specification of structure, behaviour / method & inheritance for objects.

Scanned by CamScanner

managed by the object itself.

Messages

To do an operation, a message is sent to an object. Objects perform the operation in response to the message. Following a set of conventions ~~all~~/protocols protects the developers/users from unauthorized data manipulation. Messages essentially are non-specific function calls. Different objects can respond to the same message in different ways. The message makes no assumptions about the class of receivers or arguments. They are simply objects. It is the receiver's responsibility to respond to a message in an appropriate manner. This gives a great deal of flexibility since different objects can respond to same message in different ways. This is known as polymorphism.

1/2/19

Principles of Object Oriented System

(i) Encapsulation

The mechanism of wrapping the data & the code acting on the data together as a single unit. Variable of a class will be hidden from other classes & can be accessed only through the methods of their current class. Encapsulation is also known

Scanned by CamScanner

URBAN EDGE 1/2/19

(ii) Data Abstraction

It is a natural extension of encapsulation property. It is the process of hiding implementation details from the user. Only the functionalities will be provided to the user. The user will have the info what the object does instead of how it does it. Each object should only expose a high-level mechanism for using it. It hides internal implementation details.

(iii) Inheritance

Class Hierarchy

Classes are organized into super classes & sub classes. Super classes are also known as parent or base classes. Sub classes are also known as child classes or derived classes. Super classes specify most general states & sub classes specify most specific states.

- Sub classes inherit all properties & methods of super class
- Sub classes add new properties & methods specific to that class
- Sub classes may refine or constrain the state & behaviour inherited.

Formal / Abstract Class

It has no instances & it defines common behaviours that can be inherited by more specific classes.

Inheritance allows objects to be built from other objects. It can be considered as the relationship btw a parent class & a child class. It allows share & reuse of behaviours & attributes.

Multiple inheritance inherits from more than 1 super class

iii) Polymorphism

Different objects can respond to the same message in different ways.

5/2/19

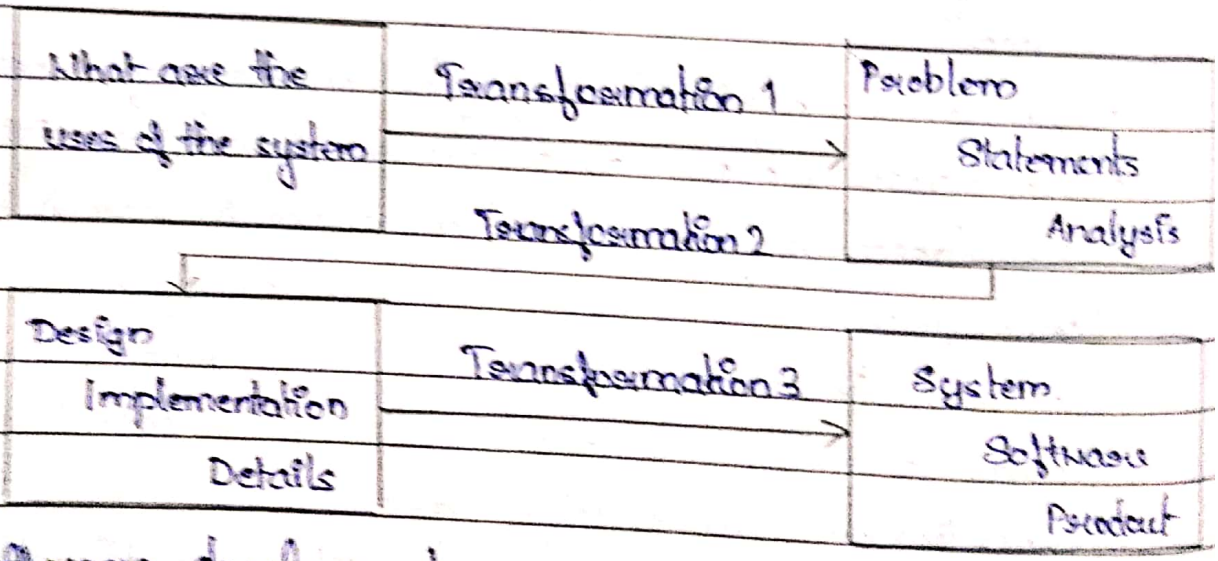
Software Development Life Cycle (SDLC)

Software development includes all activities that go into producing an info system. Software development process consists of Analysis, Design, Implementation, Testing & Refinement. Software development process is divided into subprocesses & each subprocess must have the following:

i) Description in terms of how it works

ii) Specificat^n of the i/p required for the process

iii) Specificat^n of the o/p to be produced.



Software development process can be viewed as a series of transformations where o/p of 1 transformat^n becomes the i/p to subsequent transformation.

i) Transformation 1 i.e. Analysis

URBAN EDGE 5/2/19

It translates user needs into system requirements & responsibilities. It is represented as problem statement

(ii) Transformation 2 i.e. Design

It begins with problem statement & ends with detailed design that can be transformed into an operational system

(iii) Transformatn 3 i.e. Implementation

It represents embedding the software product within it's operational environment

6/2/19

Building High Quality Software

High quality products must meet user's need & expectations. The ultimate goal of building high-quality software is user-satisfaction. The product must attain with minimal errors or defects & the focus being on improving products prior to delivery rather than correcting them after delivery. 4 quality measures are used to evaluate the system

(i) Correspondence

It measures how well the delivery system matches the needs of operational environment as described in the original requirement statement.

(ii) Validation

It is the task of predicting correspondence.

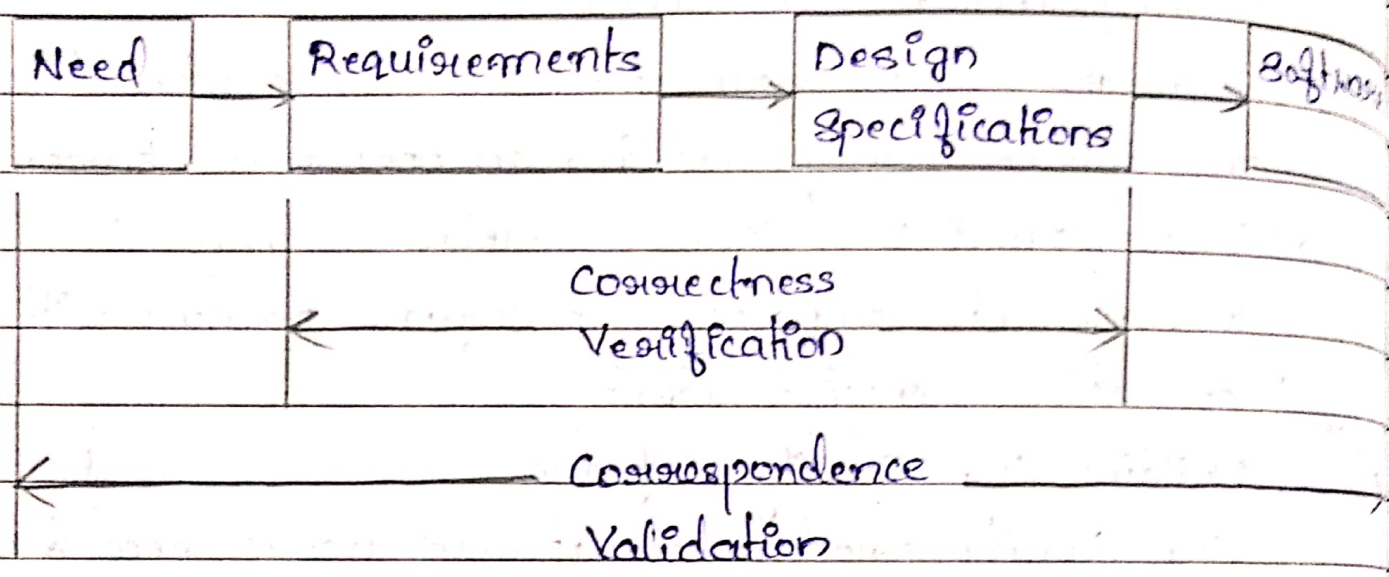
(iii) Correctness

It measures the consistency of product requirements w.r.t design specification.

Scanned by CamScanner

(iv) Verification

It is the exercise of determining correctness



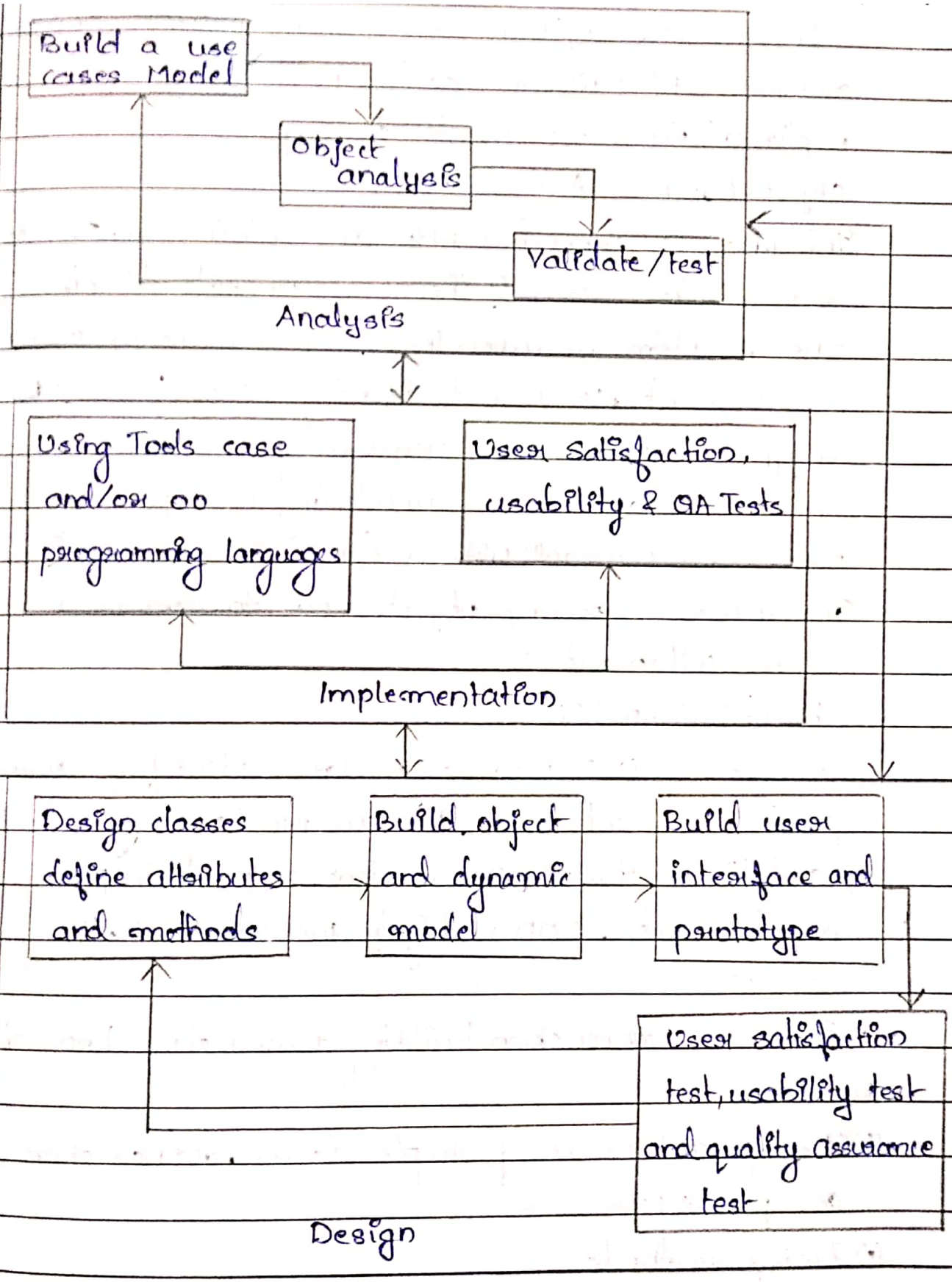
Verification specifies "Am I building the product right?"
validation specifies "Am I building the right product?"

Object Oriented System Development Life Cycle

Object oriented software development life cycle consists of 3 macro processes:

(i) Object oriented analysis
(ii) " " design
(iii) " " implementation

Scanned by CamScanner



Scanned by CamScanner

Object Oriented system development includes object oriented analysis, object oriented design, prototyping, object oriented implementation & incremental testing

(i) Object-Oriented Analysis (OOA)

This phase is concerned with determining system requirements & identifying classes & their relationships to other classes in the problem domain. In order to identify system requirement, we have to identify users/actors. Use-cases is a typical interactn b/w users & systems that captures user goals & needs. Use-case modelling is expressing high-level processes & interactn with customer in scenario & analyzing the interactn among objects roles to achieve a given goal is called collaboratn.

(ii) Object-Oriented Design (OOD)

The goal of OOD is to design classes identified during analysis phase & user interface. We can also identify & design additional objects & classes that support implementatn of requirements. OOD is highly incremental. The guidelines to use OOD is

(a) Reuse i.e. rather than building a new class, know the existing classes

(b) Design a large no. of simple classes, rather than small no. of complex classes.

(c) Design methods

(d) Critic what you have proposed. If possible go back & refine

Scanned by CamScanner

URBAN EDGE 8/2/19

the classes.

13/2/19

iii) Prototyping

It is a version of software product developed in earlier stages of product's life cycle for specific experimental purpose. Different categories of prototypes are:

(a) Horizontal prototype

It is the simulatn of interface but contain no functionality

(b) Vertical Prototype

It is the subset of system features with complete functionality. Few implemented funct's can be tested in great depth

(c) Analysis Prototype

It is an aid for exploring problem domain. It is used to inform user & demonstrate proof of a concept. It is discarded when it has served it's purpose

(d) Domain Prototype

It is an aid for incremental development & it is a tool for staged delivery of subsystem.

iv) Object Oriented Implementation (OOI)

Computer Aided Software Engineering (CASE) tools allow the user to rapidly develop info system. The main goal of CASE technology is the automatn of the entire info systems development life cycle process. Content Based Development (CBD) is an industrialized approach to the software development process. An app development moves from customer development to assembly of pre-built, pre-tested, reusable

software components that operate [illegible] linear. Rapid App Development (RAD) is a set of tools & techniques that can be used to build an app faster than typically possible with traditional methods. But the quality of product may be less.

Incremental Testing

It involves step-by-step testing of software product

Unified Modelling Language (UML)

It is a language for specifying, constructing, visualizing & documenting the software system & it's components.

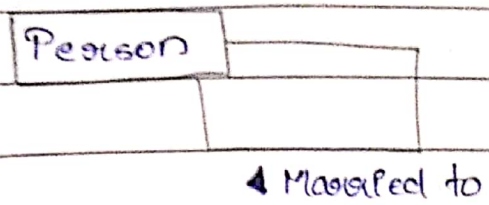
Class Diagram

It is a static model. It represents relatnships b/w classes

Notations

	Name	→ Represents both classes & objects
	Variables	
	Methods	

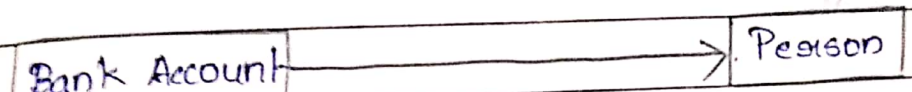
Scanned by CamScanner



Cardinality → No. of classes / objects in an associatn

→ UML Associatn Notatn

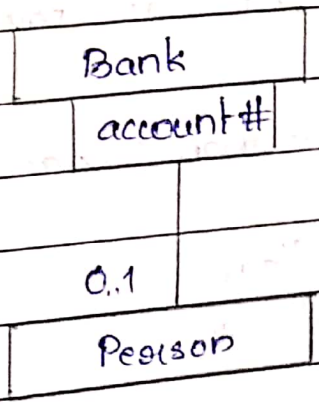
Associatn is represented as open arrow in UML



→ Qualifier

It is an associatn attribute.

eg: A person object may be associated to a Bank object. An attribute of this associatn is the account#. The account# is the qualifier here.



19/2/19

→ Multiplicity

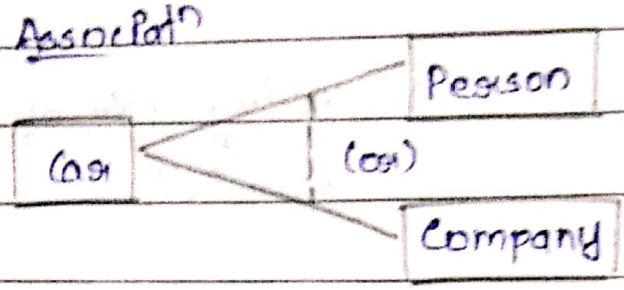
It specifies the range of allowable associated classes. It is given for roles within associatns, parts within compositns, repetitns & other purposes

lower bound .. upper bound

eg: 0..1 → 0 or 1

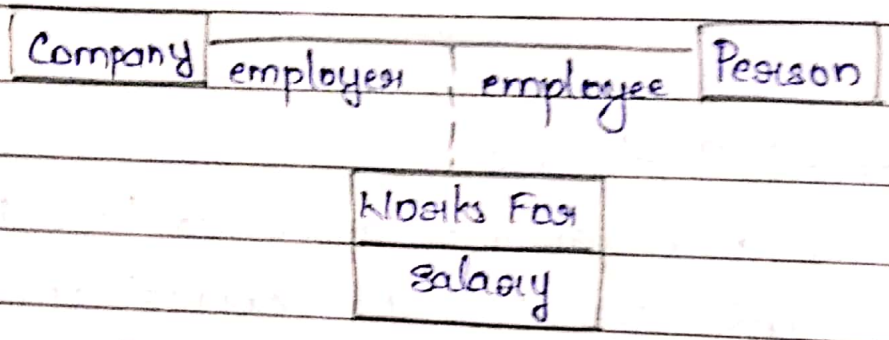
* → Unlimited

OR Associatn



Only 1 of several potential associatns may be substantiated 1 time for any single object.

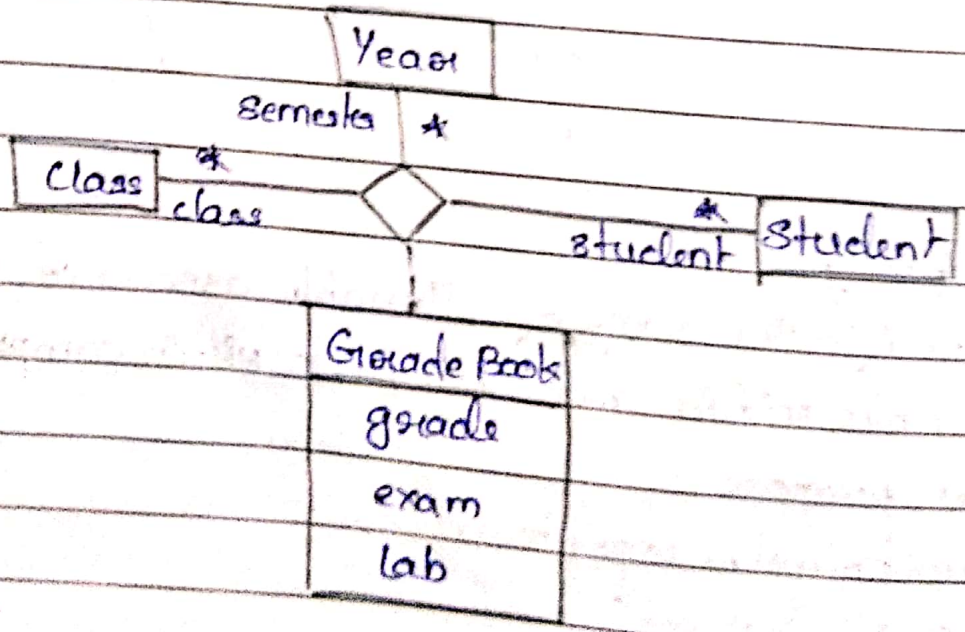
Associatn Class



An associatn that also has class properties

N-ary Associatn

Associatn among more than 2 classes



Scanned by CamScanner

→ Aggregatⁿ

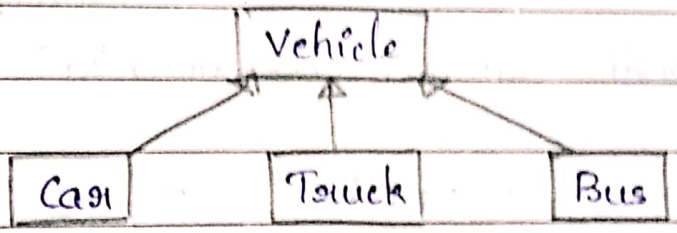
Attributes inside a class can be an object. This is known as aggregatⁿ. A hollow diamond is attached to the end of the path to indicate aggregatⁿ

→ Compositⁿ

There exists a strong ownership to represent component of complex object. Represented using solid diamond.

→ Generalizatⁿ

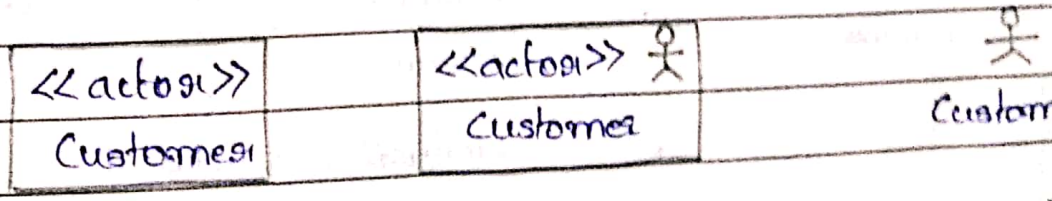
It is represented using a closed hollow-ended arrow. The arrow always points to general class. It is a relatⁿship b/w a more general class & a more specific class.



(ii) Use-Case Diagram

The use-case model defines the outside (actors) & inside (use case) of the system's behaviour.

- Actor Notatⁿs

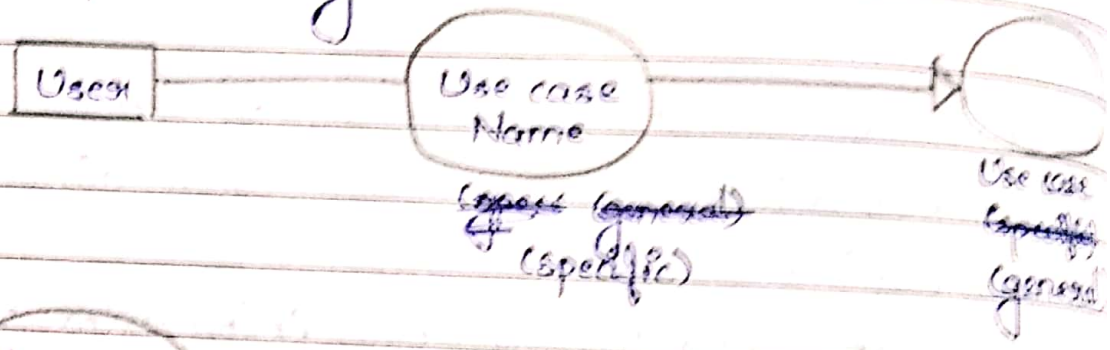


- Use-Case Notat^n

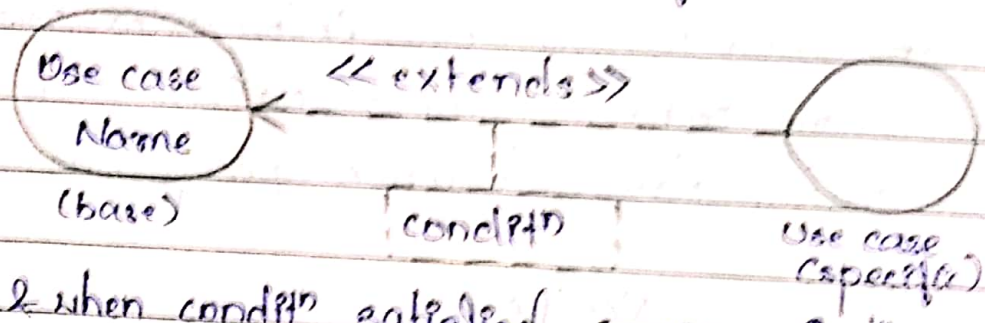
- Communicat^n

It is represented by a solid line.

→

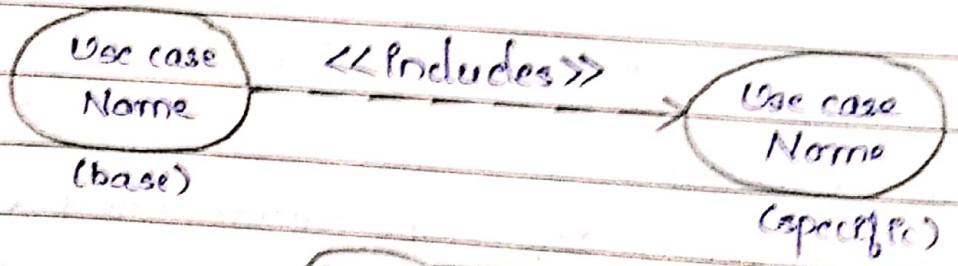


→

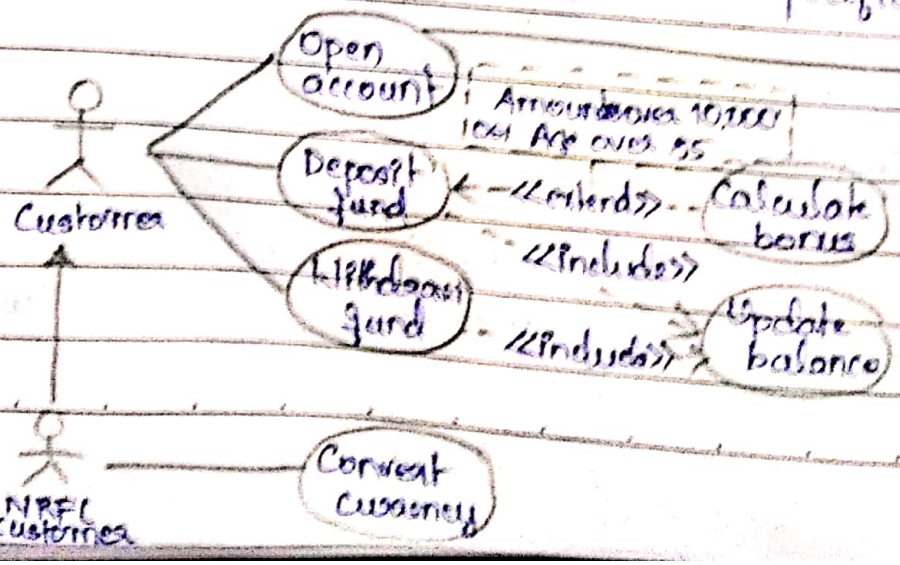


If & when condit^n satisfied, communicat^n occurs

→



eg:



Scanned by CamScanner