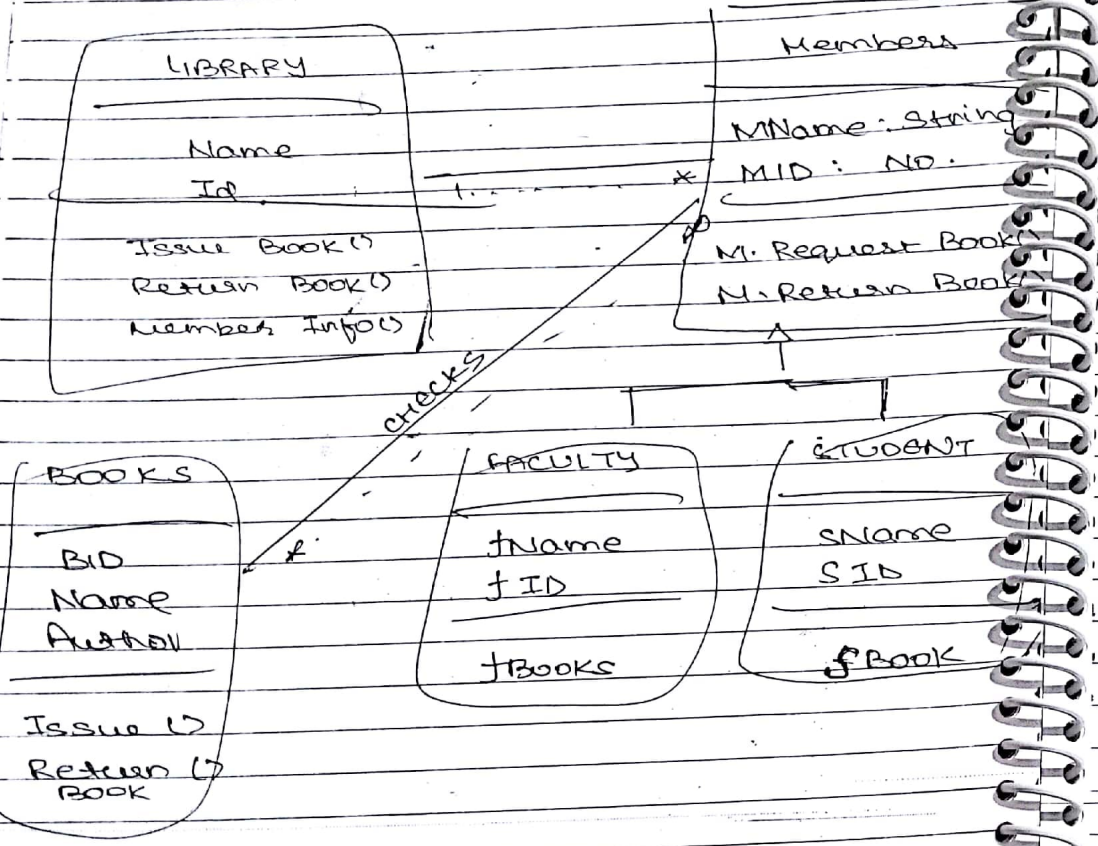
DM/CD
D(MC)

Draw the domain model/CD Library Agmnt



DATA FLOW DIAGRAM: (DFD)

Consists of following 3 members.

Level 0 which is also called as Context diagram.

Level 1

Level 2

DFD Components

1. Entities

2. Process

3. Data store —— OR

4. Data flow ——→ OR ←——

Are basically Graphical Representation of flow of data of information system.

DFD are capable of depicting the incoming & outgoing data flow & stored data.

The flow chart shows ~~in a~~ the flow of control in a program module whereas DFD shows flow of data and System at various levels.

DELTA Notebook

Entities: Is a source or destination of information represented by a rectangle. ☐

Process: Are the activities & actions taken on data shown by ○

Data Store: These are the data repositories

Data Flow: Shows movement of data.

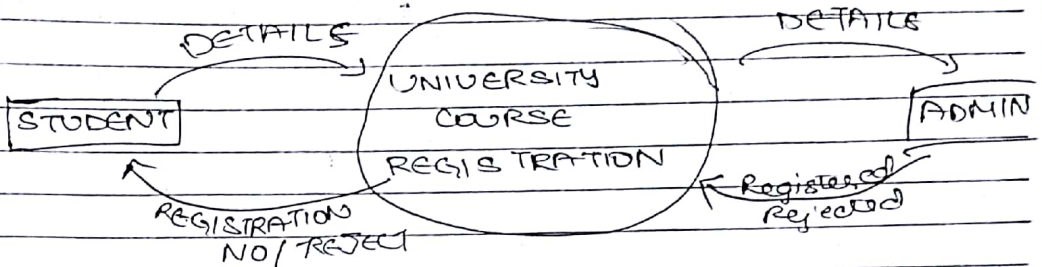
Levels of DFD:

Level 0 DFD: Shows highest abstraction Level: It depicts the entire information system as 1 diagram concealing all underlined domains. Also called context Diagram.

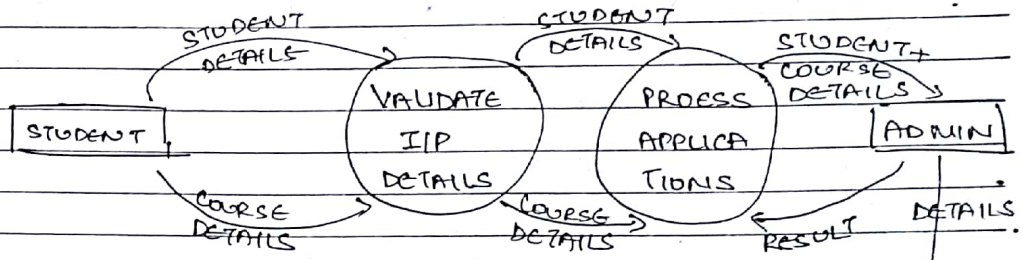
Level 1 DFD: level 0 DFD is broken into level 1 DFD. It depicts basic modules in the system and flow of data among various modules. It also mentions basic process and source of information.

Level 2 DFD: This shows how data flows inside diff modules mentioned in level 1 DFD.

Draw level 0, DFD for Course Registration level 1 & level 2



LEVEL 0



LEVEL 1

UNIT-5

Packages & Contract Operations

CONTRACT OPERATIONS:
Describes the detailed System Behaviour in terms of state change through objects in the domain model.
A contract operation identifies system change when an operation happen effectively, it will define what each system operation does.
An operation is taken from a system sequence diagram, it is a single event from that diagram.

Syntax:

Name: Appropriate Name
Responsibilities: Performing a function
Cross Reference: System functions & Use Cases.
Exception: Any inequal condition
Precondition: Something that pre-exist
post condition: An association has formed.

When taking an operation contract we think of the system before the action & state of the system after the action. We do not discuss how the state changes were done.

Pre-Cond: Includes assumptions about the state of the system or the object in domain before execution of the operation. Ps

POST-Cond: The state of the object in the domain model after completion of operation. Includes:
- An object attributes were changed
- An association was formed or broken.
- An instance of an object was placed.

Eg: Contract Operation for item sale 50%.

Name: Item Sale 50%
Operation: Enter Item(Item Id, Quantity integer)
Responsibilities: item price update for sale.
Cross Reference: Use Cases, Process Sale
Exception: -

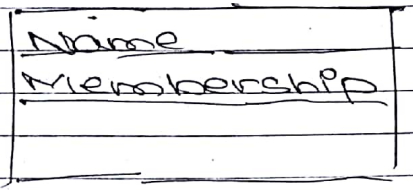
Precondition: This is a sale going.
Post Condition: A Sale line item created.

Packages: Basically UML mechanisms for grouping the items. The packages can be used to:

- Group semantically related elements
- Define a semantic boundary in the model.
- Provides Units for parallel working and configuration mgmt.
- Package is used to provide a encapsulate namespace within which all the names are must be unique.
- An analysis package contains a Used Case, Analysis class, use case realization

Syntax

a)



b) Visibilities

+ Name

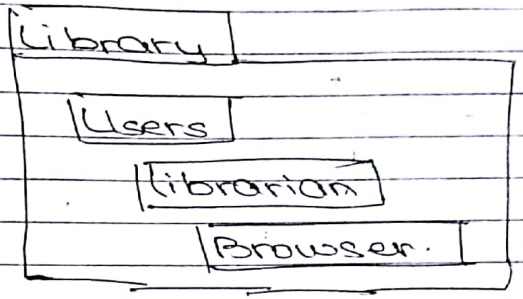
- Name

c) Stereotypes

<<frameworks>> Reusable Architecture

<<Model library>> any one use it with attributes.

d) Nested Package.

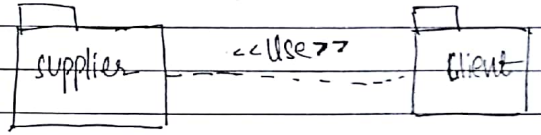


Package Dependencies:

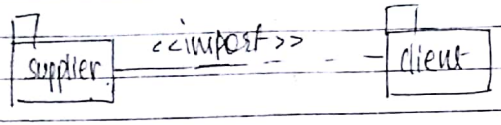
1. << use >>
2. << import >>
3. << access >>
4. << trace >>
5. << merge >>
6. << Transitivity >>
7. << Package Generalization >>

Indicates that 1 Package depends upon another package in some way. Following are the different dependencies available.

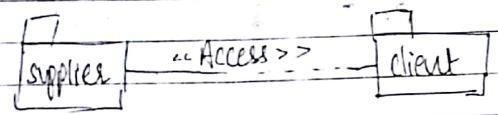
1. Use An element in the client uses a public element in the supplier in some way i.e the client depends upon the supplier.



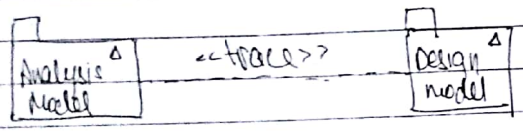
2. Import
Public elements of supplier namespace are added as public elements to client namespace. Elements in client can access public elements of supplier.



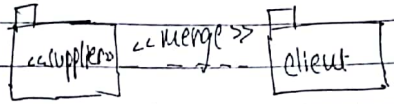
3. Access
Public elements of supplier namespace are added as private elements to client name space



4. Trace
Historical development of 1 element into another more developed version. It is usually relationship between two models rather than elements. A complete UML model can be represented by a package with a small triangle on its top.



5. Merge
Public elements of supplier package are merged with elements of client package. It is used only in meta modelling and not OO modelling.



6. Transitivity
If there is a relationship between A & B package and there is a relationship b/w B & C package then there is a implicit relationship b/w package A & C.
Ex. Import dependency is transitive in nature.

7. Package Generalisation
The more specialized child package inherits the public and protected elements from the parent package. The child package may add new elements and may override by providing an alternative implementation with same name.

UNIT 5

GRASP
General Responsibility Assignment Software Patterns

The Grasp guides in assigning the responsibility to collaborate objects. These responsibilities can be accomplished by single object or group of objects collaborating a responsibility.
The Grasp helps in deciding which responsibility should be assigned to which objects. It identifies objects & responsibilities from problem domain & also identify how objects interact with each other.

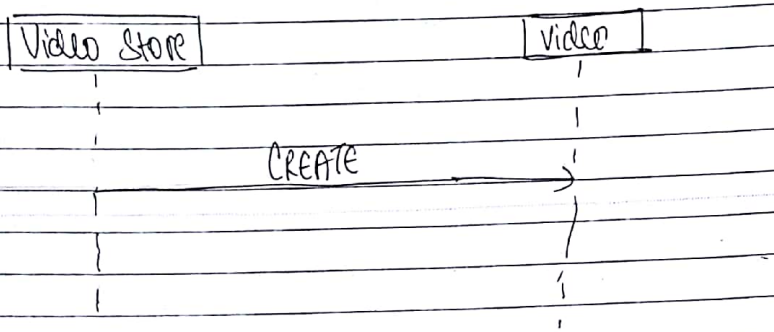
DELTA Notebook

Page No.
Date

9. GRASP PATTERNS:

1. CREATOR
2. INFORMATION Expert
3. Low Coupling
4. Controller
5. High Cohesion
6. Indirection
7. Polymorphism
8. Protected Variance
9. Pure Fabrication

CREATOR: Decides who creates an object or who should create a new instance of some class? Container object creates contained objects. The creator describes who can be the creator based on object association & these interaction.
Eg. Video Store & a video in Store. Video Store is a container & Video in Stored is a contained object. Therefore we can instantiate a video object in a video store class.



INFORMATION EXPERT: Given an object which responsibilities can be assigned to objects? Expert Principle says that assign those responsibilities to the objects for which the object has information to fulfil that responsibility. A video store has all the information about the videos therefore the responsibility

Page No.
Date

of giving videos is given to video store.

Low COUPLING: is basically how an object is dependent on another object. When the dependent changes it affects the dependent also. This is how we can reduce the impact of change in dependent elements by low coupling.

CONTROLLER: Deals with how to delegate request from UI layer object to domain object. When the request comes from UI layer, controller pattern helps us in determining which is the first object that receives the message from UI layer object. This object is called controller object which receives request from UI layer & then controls & coordinates with other objects and the domain layer to fulfil the quest. We can make an object controller if the object represents the overall system or an object represents a use case.

High Cohesion: Defines how operation of any element or class are functionally related to another. Benefits of high cohesion are low coupling, code reuse, easy maintainability

Polymorphism: Deals with how to handle related but varying elements based on element type. Polymorphism helps one in handling which object is responsible for handling those varying elements. Ex. Get Area

function to calc area of circle/square depending upon input.

Pure Fabrication: Fabricated Class or artificial classes are assigned set of related responsibility that does not represent any domain object. It provides high cohesion to the set of attributes.

Indirection: Introduces intermediate unit to communicate between other units so that the other units are not highly coupled. Indirection basically avoids direct coupling between 2 or more elements.

Protected Variance: How to avoid impact of variation of some elements to other elements. It provides a well defined interface so that there will be no effect on other units. It provides flexibility and protection from variance.

CRC Cards:
Class Responsibility, Cards
collaboration.
They are the brainstorming tools used in design of software & interclass cards. The CRC cards are basically used to represent first 1. Class
2. Responsibility of class
3. Interaction of class.

Class Name	
Responsibility	collaboration

Class Name
Super Class
Sub Class

Responsibility	Collaboration

Benefits:
- Can be used anywhere.
- Allow Participants to exp. how system will work.

Glossary

Glossary:
Class: It is a blueprint of an object
Responsibility: Something that a class does.
Collaboration: Another class with which the class interacts to fulfil its resp.
Subclass: A class which is inherited from this class
Superclass: A class from which it is inherited.

A. How to identify classes?
- Find Names in the system.
- Find use cases.

B. How to identify responsibility?
- Name of the class
- Use-case
- Find verb in use case

C. How to identify a superclass or subclass?
- Anyone can define. Guess from the inheritance chart.

D. How to identify collaboration?

- Analyse a use case if 2 or more classes exist could exists.

1. ~~Class ATM~~

~~Responsibility~~ ~~Collaboration~~

CRC Cards for ATM system
The classes that exist for ATM system are:

1. Class ATM
2. Class Cash Dispenser
3. Class Card Reader
4. Class log
5. Class Server
6. Class Receipt Printer.
7. Class WithDraw Money
8. Class Deposit.
9. Class Bank.
10. Class Operator Panel

Class ATM

Responsibilities	Collaboration
Startup when Switch on	Operator Panel
Shut down when Switch off	C
Start New session When card inserted	Cash Dispenser
Provide Access for	Session
	Class Bank

DELTA Notebook

Class Cash Dispenser.

Responsibilities	Collaboration
Keep track of cash in hand	ATM
Report if overflow cash available/not	Bank.
Dispense Cash.	

Class Card Reader

Responsibilities	Collaboration
Tell ATM when card Inserted	ATM
Read info from card	Card Reader.
Eject Card	
Return Card	

Class - Class Log	
Responsibilities	Collaboration
Log Manages Back to Back Log Response from Bank Log Receiving Envelope	BANK

DELTA Notebook

Page No.
Date

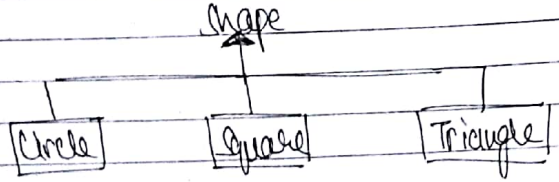
UNIT-6

1. Generalization
2. Inheritance
3. Overriding
4. Abstract Operation
5. Abstract Class
6. Polymorphism
7. Advanced Generalization

Generalization Set — Power Set

1. { concept }
2. { incomplete }
3. { Disjoin }
4. { overlapping }

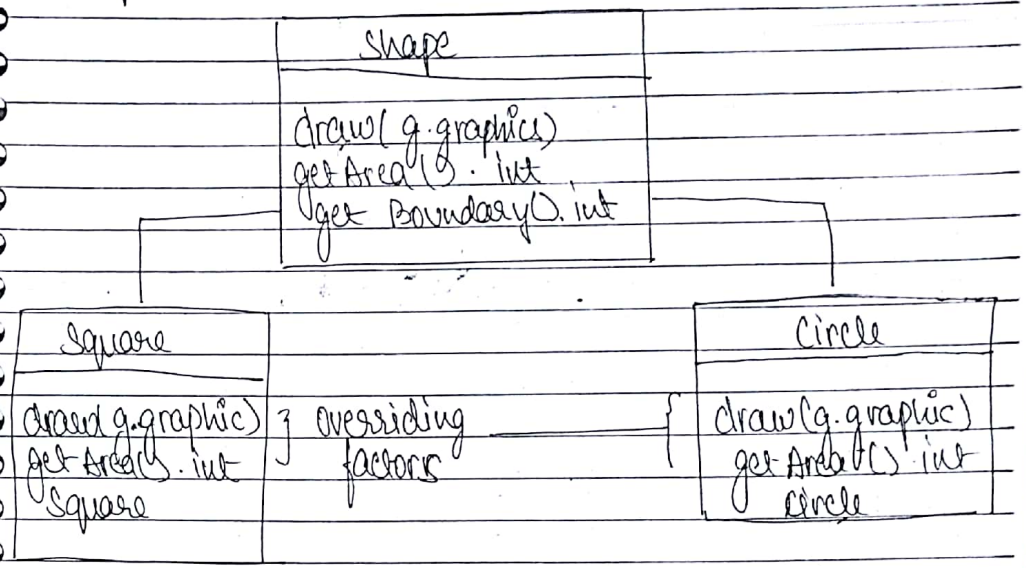
Generalization is relation b/w general items & more specific elements. Where more specific elements are consistent with general elements. It implies higher level of compile coupling b/w 2 elements. We create a generalization hierarchy of generalizing more special things.



Class Inheritance: Allows subclasses to inherit features from superclass. The features include attributes, operation, relationship, constraint. Subclass can also

add an override superclass operation.

Overriding: means adding more specific attributes of the superclass operation. Overriding is done by creating new operation with same signature as the parent.



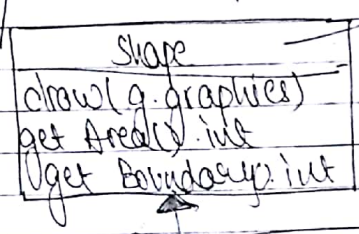
Abstract Operation has Abstract Class & operation. Any class that can be instantiated is called Concrete class.

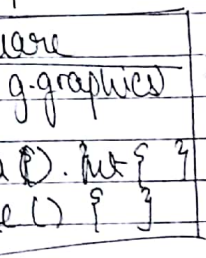
An Abstract operation is an operation which has no implementation and is denoted by 'Italics'.

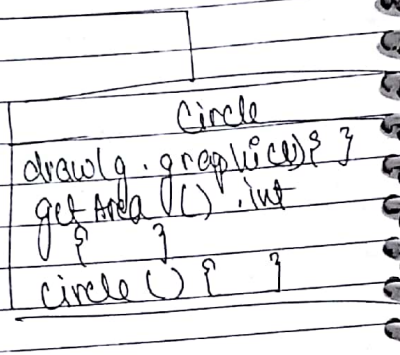
Abstract Class is a class that has 1 or more abstract operations And cannot be instantiated.

Abstract class defines a set of abstract operations that concretes subclass must implement.

Abstract Operation Example:







Polymorphism means 1 name and many forms. It means objects of different classes have operations with same signature but different implementation.

Advanced Generalization Concepts

There are 2 advanced generalization concepts.

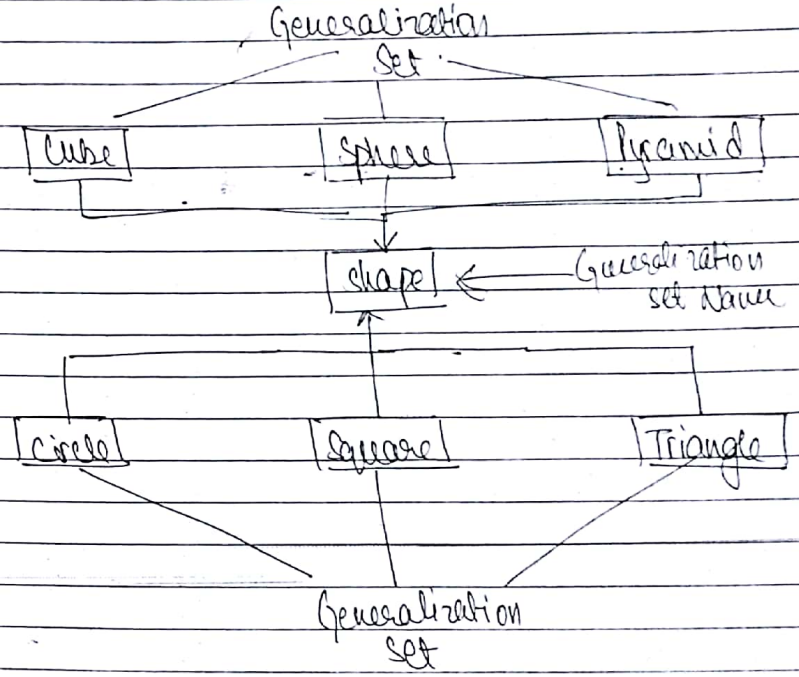
1. Generalization set → It includes set of subclasses organised acc. to a particular rule. We have following generalization sets.

The generalization set contains all possible members

{incomplete}: This generalization set does not contain all possible members.

{disjoint}: An object may be an instance of only 1 member of generalization set.

{overlapping}: An object may be an instance of more than 1 of the members of generalization set.



Power Set

A class whose instances are classes that are also subclass of another class it is called meta class.

We use the prototype to initiate to use power type software artifacts.

Artifacts are the items final or intermediate work products that are produced or used during a project.

Artifacts are used to capture project information. An artifact can be any 1 of the following:

1. Domain Model
2. Textual Document
3. Source File
4. Script
5. Binary Executable File
6. Graphic file
7. Database file

Artifact is a source of deployment to a node at a particular instance. Artifact is a classifier that represent some physical entities or piece of info. that are produced by a software development program.

UNIT-7

State chart Diagrams.
Compounds - State chart diagrams.
Diff. State chart vs Activity diagram.
Role of interface in state chart diagram
Representing State chart diagram through collaboration diagram.
Deployment diagram & its importance

State chart diagram describes different stages of components in a system. The states are specific to a component or object of a system. A State machine can be defined as a machine which defines different stages of an object & these stages and states are controlled by external & internal events.

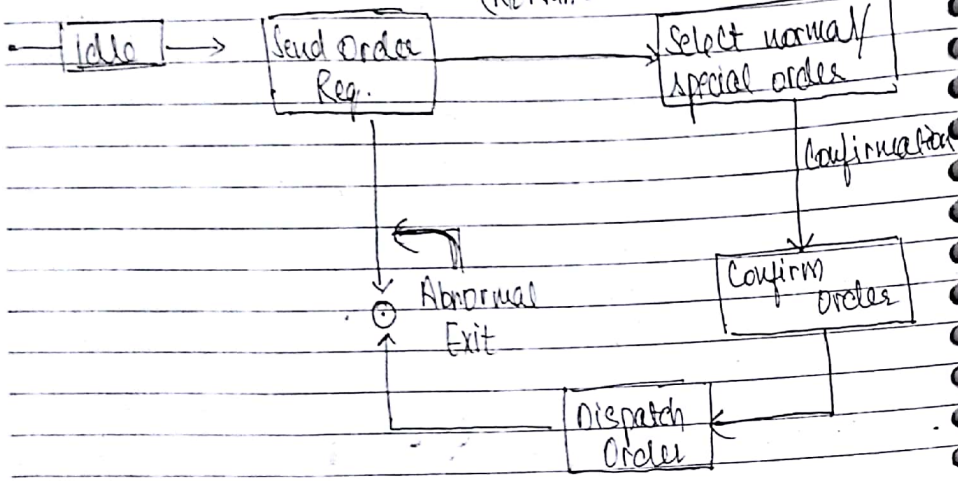
Purpose of SCDia.
Used to model dynamic nature of system. They are used to define different stages of an object during its lifetime & these states are changed by events. describe the flow of control from 1 state to another. States are defined as a condition in which an object exists. It changes when some events are triggered.

- To model dynamic aspect of the system.
- To model lifetime of a reactive system.
- To describe state of object during its lifetime.
- Define a state machine to model state of an object.

Symbols of State chart diagram:
- Initial State
- ⊙ final State
- ▭ intermediate State
- ------> transition from 1 state to another

Steps to draw SD.
- Identify imp. objects for analysis.
- Identify the states.
- Identify the events.

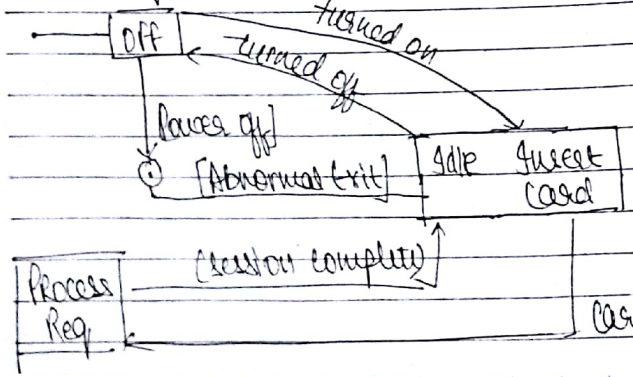
Draw a SCD for order processing.



Object: Order
Status: Idle
Order Req.
Process order
Confirm order
dispatch order

Events: Click order,
Confirmation,
Skipping order

SCD for ATM



DELTA® Notebook

Activity Diagram:
defines dynamic aspect of the system.
Basically flow chart to represent one activity to another.
Activities are basically operations of the system.
The flow from 1 activity to another can be sequential, branched, concurrent. The other 3 diagrams shows message flow from 1 object to another whereas activity diagram shows message flow 1 activity to another.

Purpose
- To show activity flow in a system.
- To describe sequence from 1 activity to another.
- To describe sequential, branched, concurrent flow of system.

When to draw AD:
- Modeling business rules.
- High level understanding of system.
- Modeling work flow by activities.

Symbols of AD:
- Initial
- ⊙ final
- decision node
- Activities
- [ ] Events
- → → → John
- → → → fork.

DELTA® Notebook

Scanned by CamScanner

Difference btw:

State Chart D. | Activity Diagram

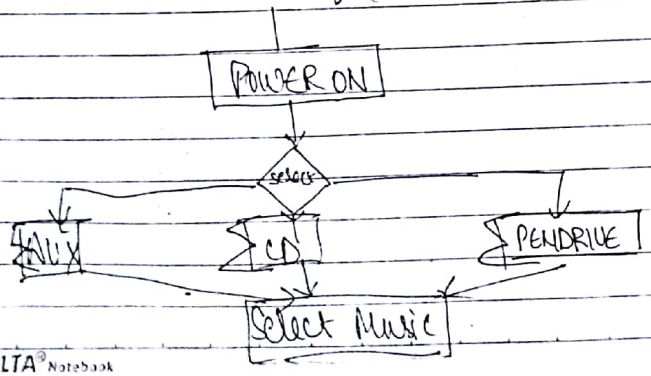
Both Activity and State chart model shows the dynamic behaviour of the system. Activity Diagram is created for flowchart showing the flow of control from activity to activity. State chart diagram shows the state machine showing the flow of control from state to state.
An activity diagram is a special case of state chart diagram in which almost all the states are activity state and transmission are triggered by completion of activity diagram.

State chart Diagram: It is a graphical representation of a system that is created for dealing with the dependent of behaviours of the system. Example: Telephone switching system & Control system.

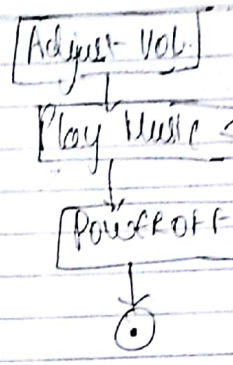
Difference btw

Sequence

Draw Activity Diagram for Music System.



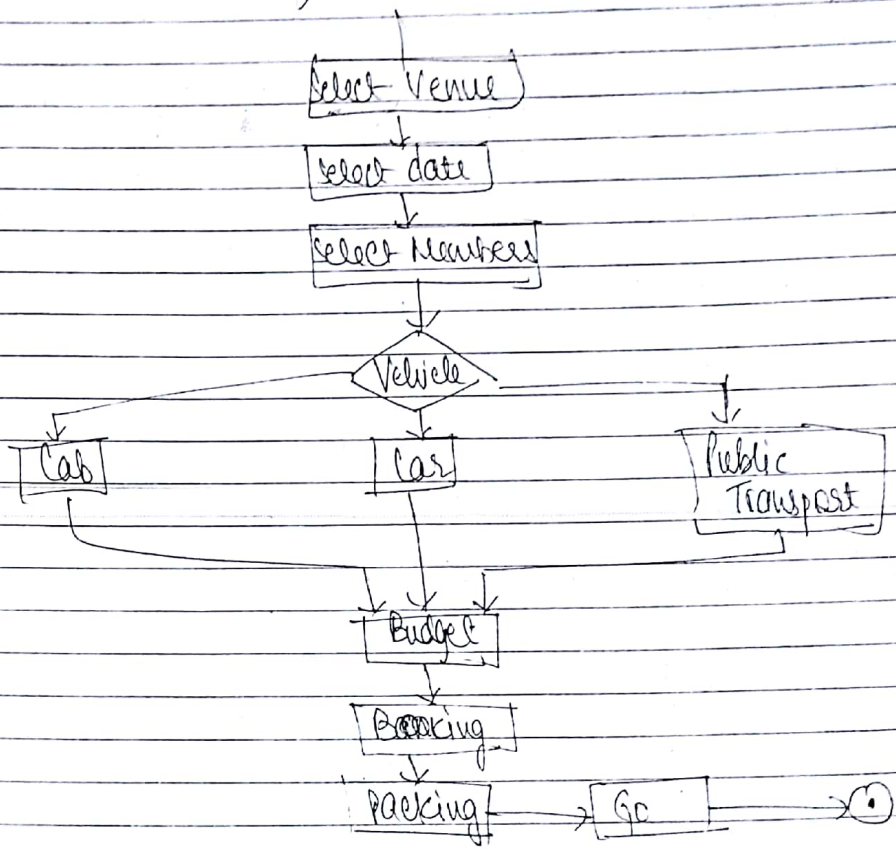
DELTA® Notebook



Input

Output

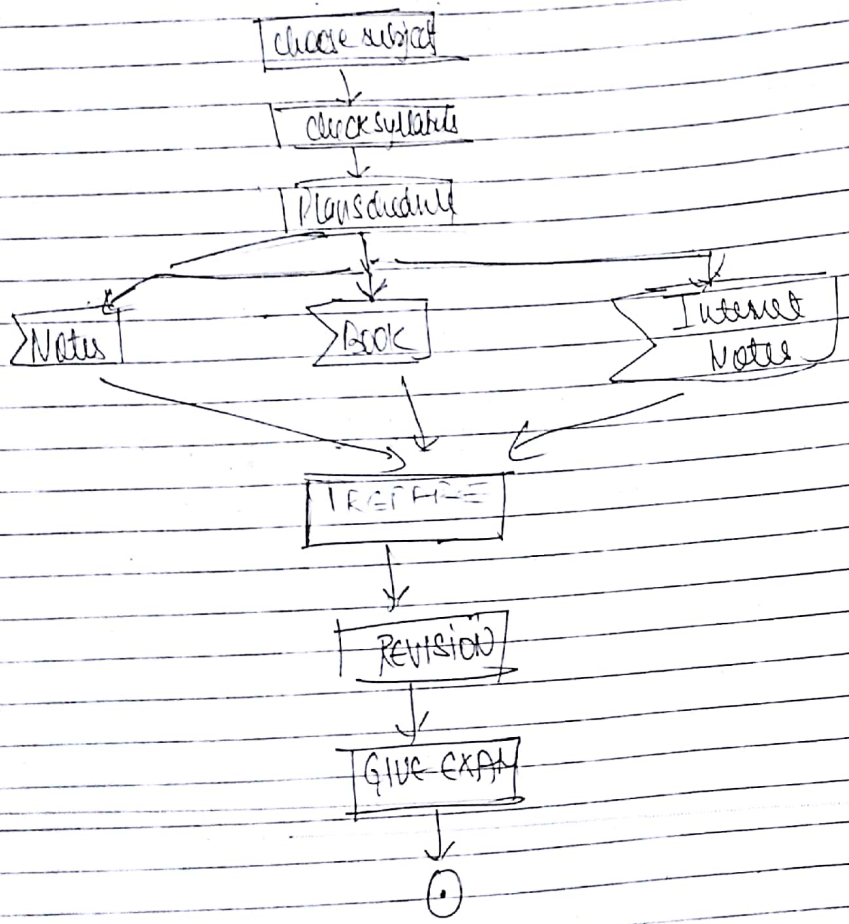
Draw an Activity dia to Plan a trip.



DELTA® Notebook

Scanned by CamScanner

AD for preparing for an examination:



Symbols of Activity Diagram

- Initial
- Final

Join Node is a control node that has multiple incoming edges & 1 outgoing edge. Join is used to synronise incoming concurrent nodes.

Fork Node is a control node that has 1 incoming edge & multiple outgoing edges & ~~edges~~ it is used to split incoming flow into multiple concurrent flows.

Page No.
Date

Deployement Diagram.

Deployement Diagram are used for describing the hardware components where software components are Deployed.
Deployement Diagrams are used to describe the status of the System.

=> USAGE :

1. hardware topology of the System.
2. forward & Reverse Engineering.
3. Deployement Diagrams are basically used to visualise topology of Physical components of a System, where software components are Deployed.

Deployement Diagrams are used to describe static Deployement view of System & Consist of nodes & their relationship.

=> Purpose :

1. They are used for describing hardware components where software components are Deployes.
2. Component Diagrams are used to focus on components & Deployement Diagrams are used to show how they are Deployed in hardware.

(8) The Deployement Diagrams focuses on hardware topology of a System.

=> How to Draw Deployement Diagram?

The Deployement Diagram Consists of Nodes. Nodes are nothing but Physical hardware Components used to Deploye an application. An efficient Deployement Diagram Controls, Performence, Scailability, maintainability & affordability.

=> Where to use Deplayment Diagram?

(1.) Deployement Diagram are used to Describe Physical Components, their Distribution & association.

(2.) Deployement Diagrams are used to model network topology of a system.

(3.) To Model A hardware Detail of a Distributed Application.

(4.) To Model embeded System.

P.T.O

=> Deployement Diagram Elements:

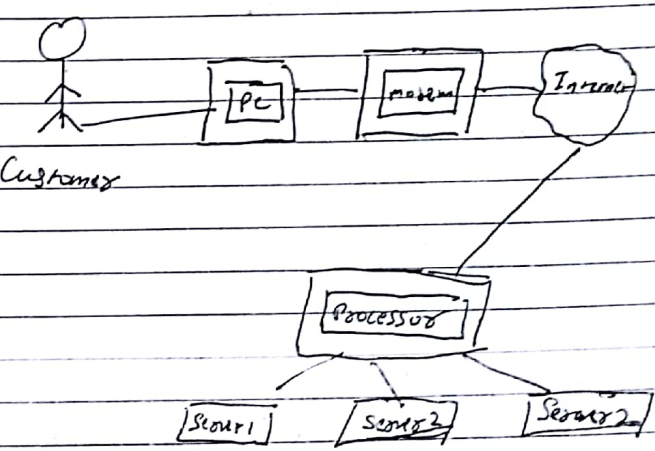
(1.) Artifacts : Artifacts are the Products Developed by Sw.

(2.) Association: Association is a line indicating Combination btw Nodes.

(3.) Components : Rectangle with 2 tabular lines indicating Software Elements.

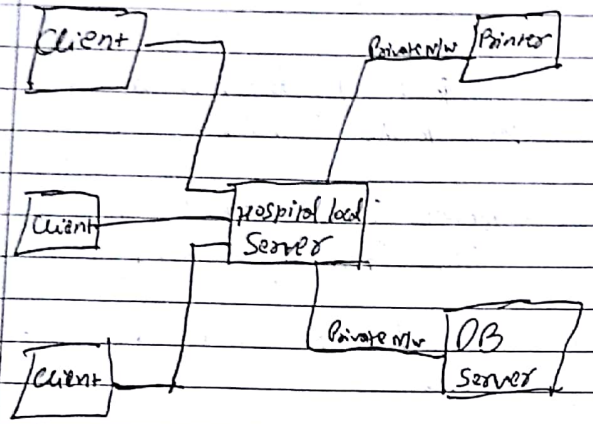
(4.) Node: It is a hardware, Software objects.

Q. Draw A Deployement Diagram for order Management System.



Page No.
Date

Q. Draw a Deployement Diagram for Hospital Management System.



Q. Draw A Deployement Diagram for ATM Machine.

Same As Above

=> Component Diagrams.

Component Diagrams are used to model Physical Aspects of a System. They deal with Executable libraries, files, Documents etc. which resides in a Node. The Component Diagrams are used to visualise the organisation and relationship among Components in a System.

=> Purpose of Component Diagrams:

(1) They are used to Describe Static implimentation view of a System. Static implimentation Deals with Organisation of Components at a particular moment.

(2) They are used to visualise the Components of a System.

(3) Construct Executables by using forward and reverse Engineering.

(4) Describe Organisation and Relationship of Components

(5) Flow

=> How to Draw Component Diagram?

(1.) Identify the files used in System.

(2.) Identify library and other artifacts relevant to application.

(3.) Identify relationship among objects.

(4.) After identification :

(A) Give meaningful name to identify the Components for which the Diagram is to be Drawn.

(B) Prepare layout before using the tools.

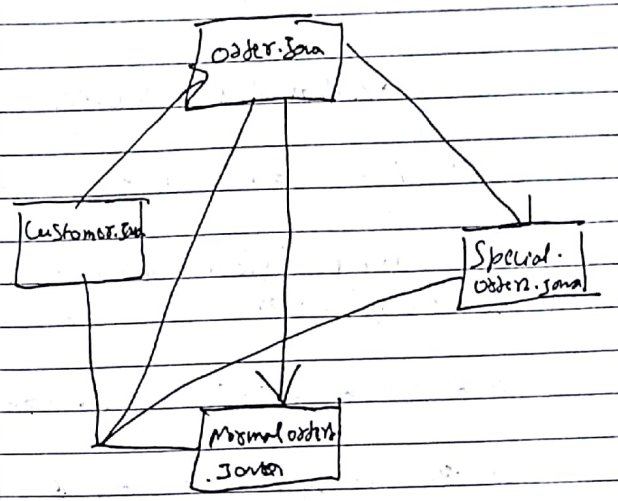
(C.) Use nodes to identify important Points.

=> (5.) ~~Just~~ Where to use Component Diagrams :

(1.) To model Components of a System.
(2.) To model Database, Schema
(3.) To model Executables ~~function~~ of an application.
(4.) To model System Source Code.

R To

Q. Draw a Component Diagram for Order Management System.



=> Object Diagrams : Are derived from Class Diagrams. They represent an instance of Class Diagram. They represent Static view of a System. But this Static view of System is a Snapshot of a System at a Particular moment. Object Diagrams are used to Represent Set of Objects and their relationship as an instance.

P. T. O

Page No
Date

⇒ Purpose of object Diagram:

1. Forward & Reverse Engineering.
2. Object relationship of a System.
3. Static view of an instance.
4. Understanding Object behaviour & relationship from Practical Perspective
5. ⇒ How to Draw Object Diagram?

1. Analyse the System & Decide, which Instance have important Data associated.
2. Consider only those instances which will cover the functionality.
3. More Some optimisation as the number of instances are unlimited

6. ⇒ Important Points to remember about Object Diagram:

1. Object Diagrams Should have meaningful name to indicate the Purpose

Page No
Date

2. Identify the most important elements.
3. Classify the Association among objects.
4. Value of Different elements seem to be Captured. to include in object Diagram
5. Add Proper Notes and Points where more Quality is required.

⇒ Where to use object Diagram:

1. for making the Prototype of the System.
2. Reverse engineering.
3. Modify complex Data Structure.
4. Understanding System from Practical Perspective.

⇒ Difference b/w object Diagram and Class Diagram.

P.T.O

Page No.
Date

=> Class Diagram represents an abstract model consisting of classes and their relationship	An Object Diagram represents an instance at a particular moment which is concrete in nature.
=> An Object Diagram is more close to actual System.	