Date
Page 13th June

Ch2. Object Oriented Fundamentals

Structure vs OOD

Parameters	Structured	OO
i) Methodology	SDLC (Waterfall)	Iterative / Incremental
ii) Emphasis / Focus / Decomposition level	Functions	Objects
iii) Risk	High	Low
iv) Usability	low	High
v) Maturity	Mature and widespread	Emerging
vi) Suitable for	Simple & Small program with well defined user requirements	Complex programs with changing requirements
vii) Phase	E-R Diagram Data Analysis CRC Flowchart DFD (level D)	Use-Case Domain Model
viii) Design	UI Design DB Design	Interaction Diagram Class Diagram

OOA & OOD

Parameters	OOA (what)	OOD (How)
i) Defn	Process that examines requirements from the perspective of classes and objects	Process that provides concepts solution to fulfill those requirements.

Parameters	OOA	OOD
a) Goal/Emphasis	Emphasis on describing objects/concepts in problem domain.	Emphasis on d... objects and how... collaborate with ea...
b) Questions asked?	Wh? Questions	How? Questions.
c) Models Used	~~Interaction Diag~~ Use-case Domain Model	Interaction Diag... Design Class Dia...

Requirements

- capabilities and conditions/features to which system m... conform/fulfil.

Requirements

- Functional/Behavioral → compulsory
- Non-functional/Quality requirem... → optional
  - performance
  - security
  - cost
  - reliability

Types: FURPS

- Functional → Necessarity: features/compo...
- Usability → UI, support, documentation
- Reliability → security, reliable
- Performance → throughput, TAT high
- Supportability → modif. flexibility, easy modifi...

TAT: Turn around time

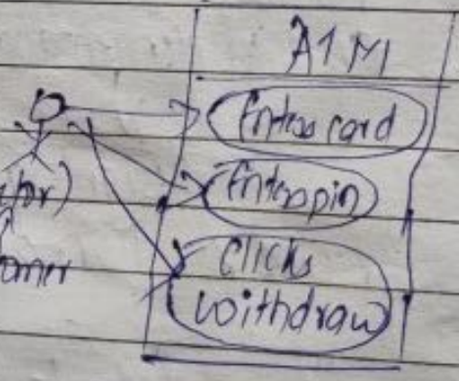
Requirement Process:

1) Fact finding and Requirement Elicitation
   i) Interviewing
   ii) Open-ended + Close-ended question
   iii) FAST (Facilitated Application Specification technique)

2) Questionnaire, Survey, Research of market by company.
   ii) Client + Analysis team (Democratic)
   iii) Open-ended :) random questions. :) context-free Questions
   Close-ended :) specific (task specified) :) for technical clients.
   ↳ done after open-end Questioning when known most about the system

iv) joint team of client + company - (off-site meeting) to identify problem, propose solutions, negotiate and specify preliminary requirements.

* Use-Case
- actor, goal, system relationship



- narrative document/diagram that describes sequence of events/goals of actor using a system to fulfill his/her requirements

Goals:- Discover + Record functional requirements
Actor:- External agent/entity that act participates in a system having definite goals to be fulfilled.

Types of Actor

i) Primary : lead Actor, direct participation with sys.
ii) Secondary : Supporting Actor, provides info to primary actor.
iii) Off-stage : No Indirect participation, provides supp. to both primary and Secondary actor.

ATM - P:- (Customer), S (Machine) O (Bank)

Eg:-

P	S	O
Customer	Cashier	Manager

Type of Use Case Text

Use-case Text

i) Essential UC → OOA (WH) → remains Mic of technical and implementation details → In designing helps ∧ Domain model

ii) Real / concrete UC · (How) → OOD → - Technical - Committed to I/O technology → Helps Interaction Diag

Use-case Diagram

Diagram

Case Study :- Customer withdrawing money from ATM

Essential Use Case → (Abstraction in nature)

Actor Response	System Response
1. The customer identifies himlherself	2) Display Menu
3. Select withdraws from menu	4) Hands amount

Real/Concrete (less abstraction with more technical details)

Actor Response	System Response
ATM	
1. Inserts card	2) Prompts for PIN
3) Customer enters their 4-digit pin.	4) Checks for verification. If valid, display menu else, another try prompt
5) Select withdraw from menu	6) Prompt for withdrawlamount
7) Enters withdraw amount	8) Check for balance if sufficient hand money else display error message
9) Eject card. ——→	

Scenario:- Sequence of events /actions or interaction between actor and system

Date 19th June
Page

Types:- Main scenario (basic flow) & Alternate Scenario (alternate flow) or extensions

Formats of Use-Case

i) Brief
ii) Casual
iii) Fully-dressed

i) Brief Format
- One paragraph summary (few sentences) of main scena
- abstract / high level of abstraction.
- essential in nature (non-technical)
- Quickly understand system requirement and complexi
- Essential in OOA phase when gathering requirements.

Success guarantee Main Scenario (All prob solved) Alternate

Scenario: Buying Goods from Supermarket
Brief format
- Customer selects/(collects) goods (and items) and arrives at counter. Cashier records sale items and calculates total. Cashier asks for payment. Customer pays and the required amount and leaves with goods.

with if main scenario fails backup plan

ii) Casual Format
- multiple paragraphs that cover various scenarios (main and alternate scenario)
- addresses alternate scenarios and provide backup plan
- Real in nature :) technical (& concrete)
- Generally (2 para, 1 → main scenario ...

.....

Alternate Scenario ...

Casual Format
Main Scenario:
Customer selects good and arrives to POS counter. Cashier logs in into system and scans Universal Product Code (Identifier code) of products. Cashier enters all item and generates total. Cashier ask for payment and customer pays through Debit Card. The Receipt is generated and customer leaves with goods and receipt.

Alternate Scenario
If system fails, cashier restarts system and reports problem the system returns to prior working state. If Product code is absent, cashier manually enters code through keyboard. If card fails or it has insufficient balance, ask authorisation for another payment method.

iii) Fully-dressed format
- more detailed and complex
- less abstraction
- designed after all other use cases have been identified.

Fully-dressed template
Use Case Name: Start with verb
Scope: System under design
Level: User goal or sub function
Primary Actors:
Stakeholders and Interests:
Preconditions:

Success Guarantee (Post-conditions)
Main Scenario:
Extensions (Alternate scenario):
Special Requirements: (Non-functional)
Technology and Data Variation list
Frequency
Miscellaneous (open ended Questions)
↳ Discount, tax

Eg:- Buying goods

1) ← Use case name: Buying goods
Scope : POS Application
Level : User goal
Primary actor : Customer : wants item as soon as possible
Stakeholders and Interests: (Cashier, SalesPerson, Manager
Cashier:- fast and accurate item
Salesperson:- more purchase from custo
so as to get commission
Manager:- More profit
Pre-conditions :- Customer has already selected goods to buy
Cashier is authenticated and logged in.

Post-conditions:- Receipt is generated
Goods are received by Customer
Payment is Received.

Main Scenario: 1) Customer selects good and arrives at
counter.

UPC - Universal Prod. Code (12 dig.)
EAN - European article NO. (13 dig)
JAN - (Japan " ") (13 dig.)
SKU (Stock keeping unit)
↳ alphanumeric

Barcode Scanner (Scanner attributes)

2. Cashier logs into system and scans Product (code of product).
3. Cashier enters all items and generates total.
4. Cashier ask for payment and Customer pays through Debit card.
5. The receipt is generated and customer leaves with goods & receipt.

Alternate Scenario:

2 a1: If system fails, cashier restart system and reports problem.
2 a2: If product code is absent, cashier manually enters code through keyboard.
4 a. If card authorisation fails or has insufficient balance, ask ~~authorisation~~ for other payment method.

Special Requirements:
Credit card authorisation should be made within 30 seconds.
Cashier reads items and calculate total under a minute.

Technology and Data Variation list
The product code is scanned through Bar-code scanner if present else manually enter through keyboard.
Product may be in UPC, EAN, JAN, SKU coding scheme

Frequency: continous

Miscellaneous: How much tax percent was included?
Is there any discount or offers?

Date Page 19th June

2. Example :- ATM

Use case name : Withdraw amount/ Money
Scope : ATM
Level : User goal
Primary actor : Customer : wants money as soon as p
Stakeholders and Interests : Customer : fast and easy access, wan
appropriate amount to be deducted. a
Bank : Wants more Customer /day
Manager (Issue ATM cards)
ATM operator : Store money on ATM

Pre-conditions : Customer should have ATM card and ATM should be in working condition. OR ATM ~~Condition~~ Customer is logged on

Post-conditions : Amount is received., account is debited a
Receipt is generated. to

Main Scenario : 1) Customer inserts card ~~and~~ gets verified.
2) Prompts for PIN.
3) Customer Enters 4-digit pin ~~to~~ select
4) Display menu. (If verified)
5) Select withdraw 6) Enters amount
7) Hands amount.

Alternate Scenario : 3a : If pin is incorrect, prompt another 3 t
6a : If withdraw amount is not suffic
display error message.

Special Requirements:

Pin code and display message should be authorised and viewed in less than ~~90~~ 30/20 seconds.
ATM should function well.
Easy UI, Quick authentications, Disable Friendly UI
Technology and Data Variation list
- UI language can be either in Nepali or English
- Receipt can be generalised in hard copy if printer available, else digital notifications

Frequency: Continous

Miscellenous: What is the maximum & minimum withdrawal limit?
Are there any free transactions?

3. Example:- Doctor appointment

Use case name: appointing Doctor only
Scope : Doctor Appointment Application
Level : User goal
Primary actor : Patient : wants appointment and treatment as soon as possible

Stakeholders and Interests : Application System: provides appo

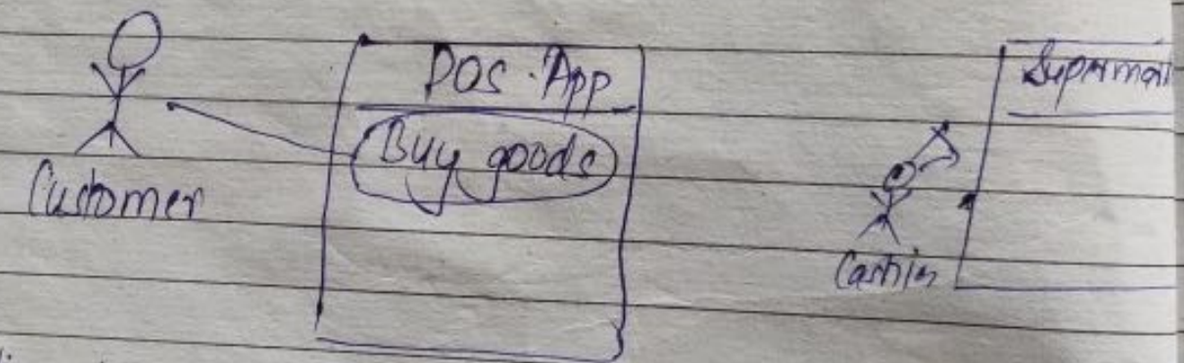
June 26

Use Case Diagram

-> Relationship between actor, goals and system
-> Diagrammatic representation of system functional requirements.

Representation:-

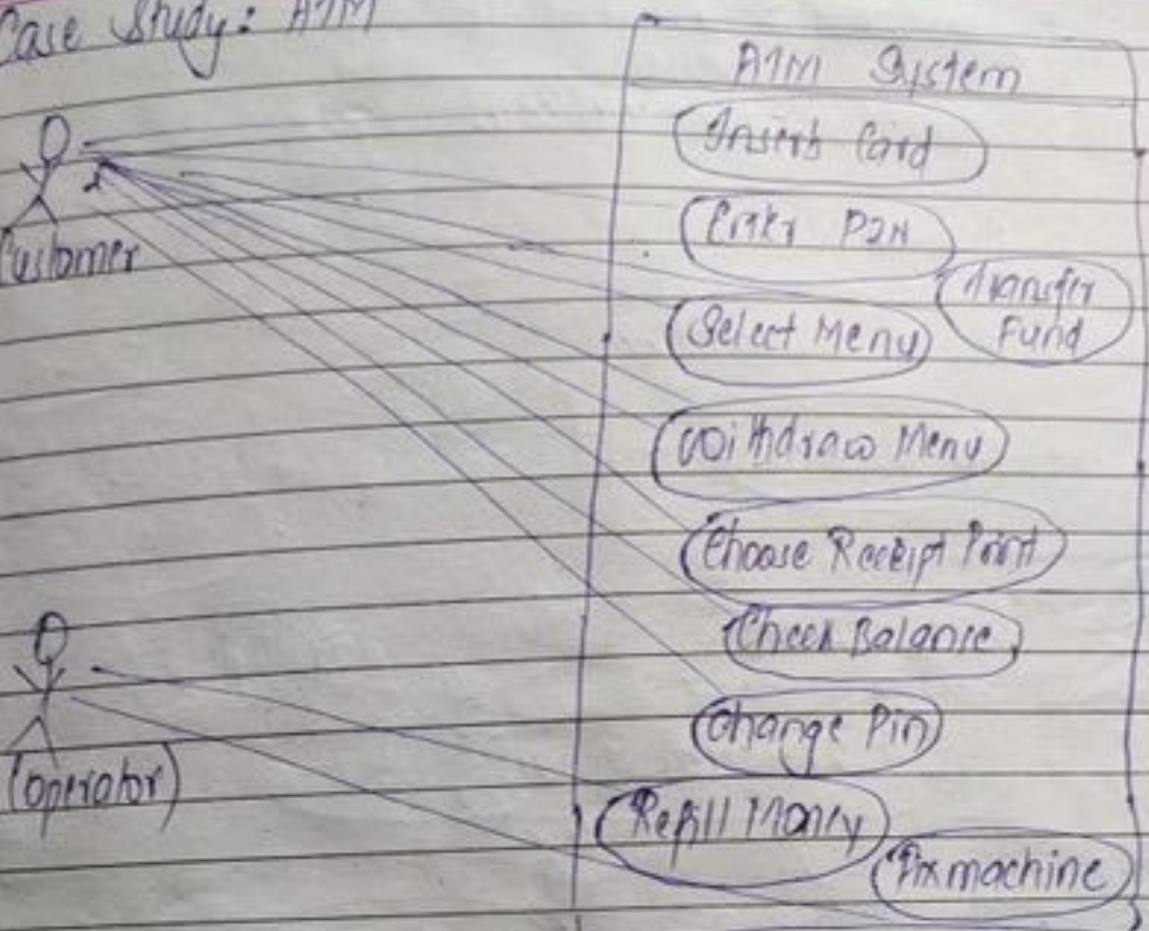
Eg:-



Guidelines:

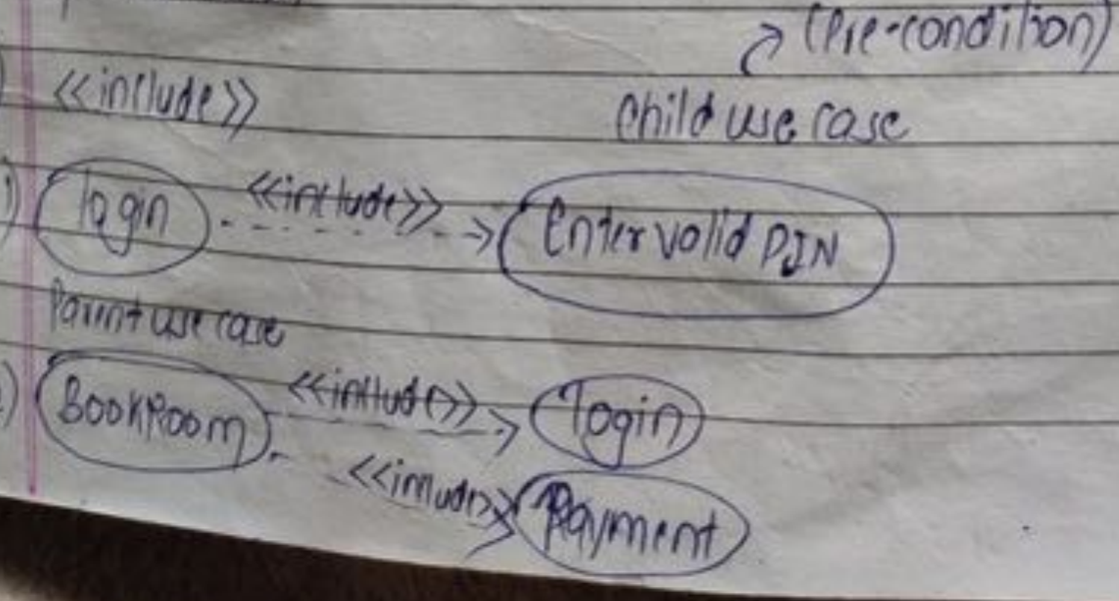
1) Choose system Boundary (Identify internal Vs external actors)
2) Identify primary actors.
3) For each actor, identify user goals.
4) Define use-case that satisfies user goals. (Start with u

Case Study: ATM

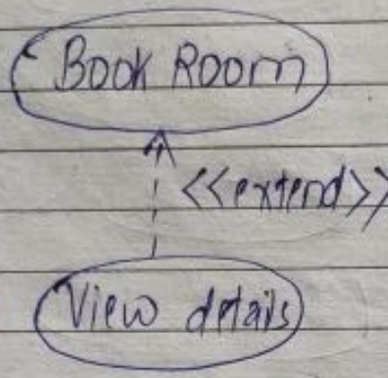


Use case Terminologies / Stereotypes

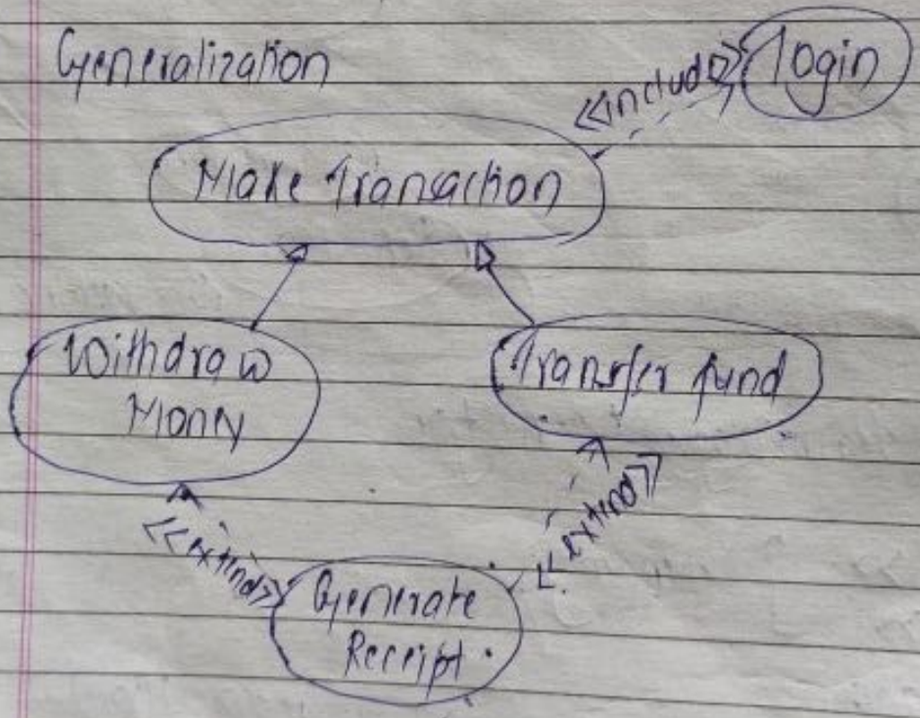
1) <<include>>] Dependency
2) <<extend>>
3) Generalization

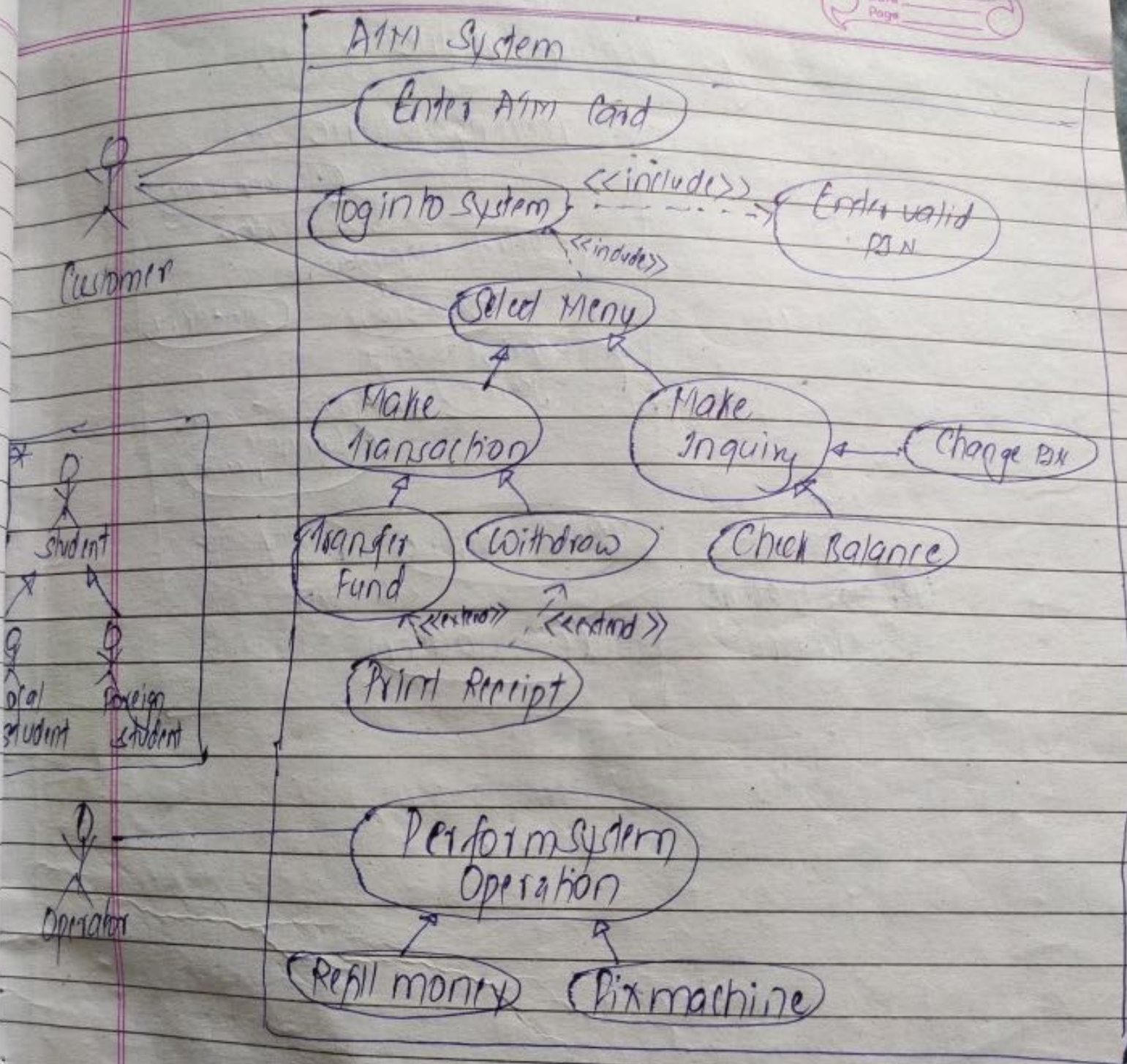


2) <<extend>>
- optional use case
- non-functional use case

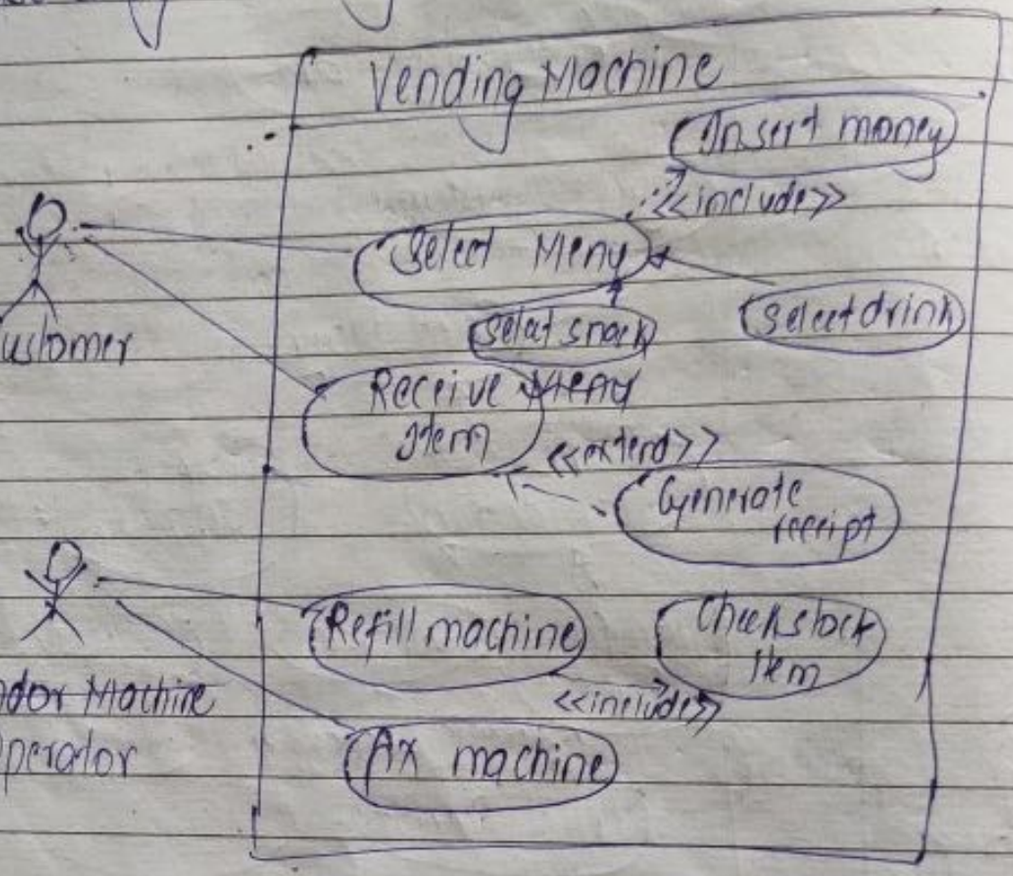


3) Generalization



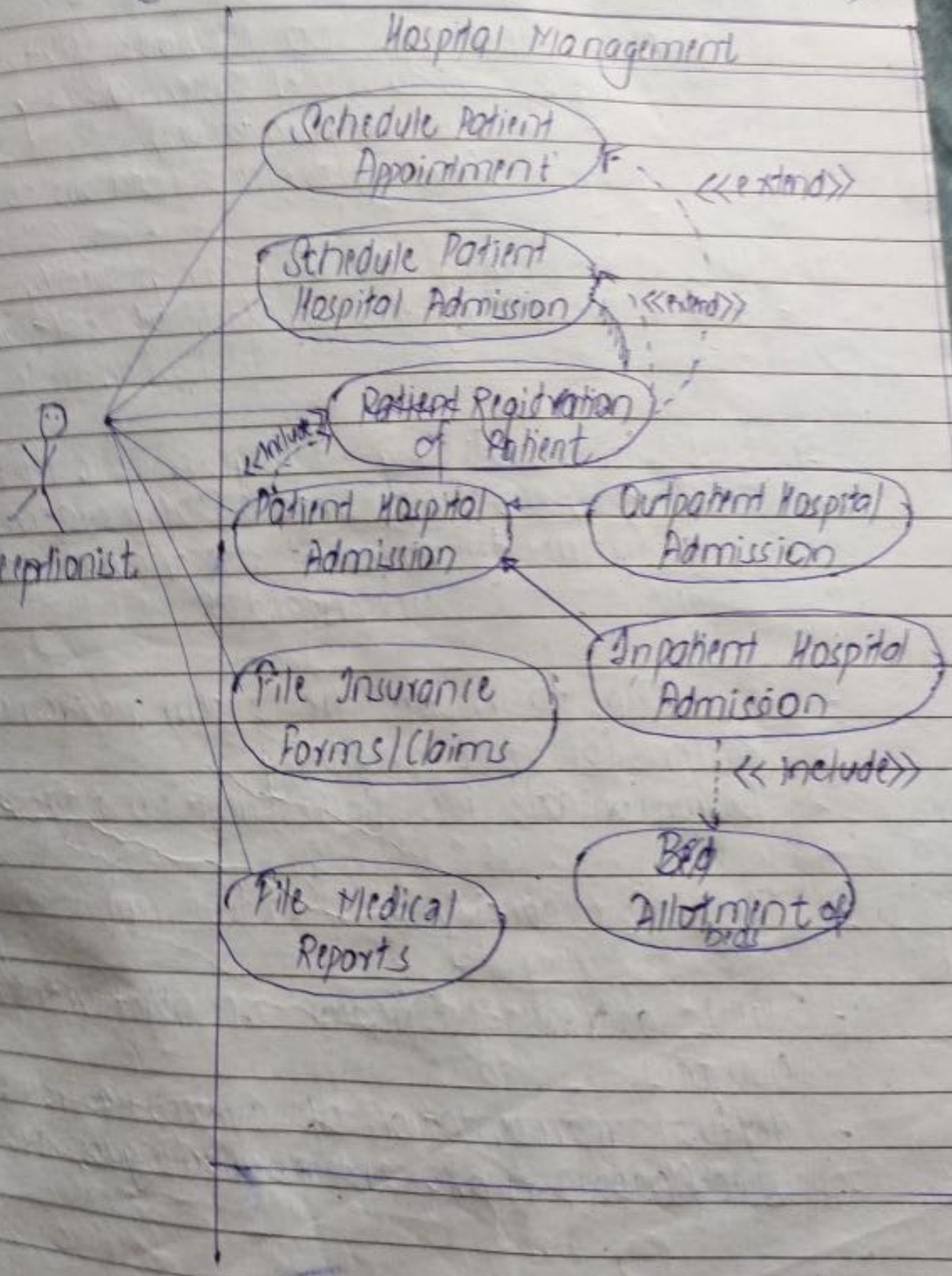

ATM System
Enter ATM Card
Customer
Login to System
<<include>>
Enter valid PIN
<<include>>
Select Menu
Make Transaction
Make Inquiry
Change PIN
Transfer Fund
Withdraw
Check Balance
<<extend>>
<<extend>>
Print Receipt
Student
Local student
Foreign student
Operator
Perform System Operation
Refill money
Fix machine

Case Study: Vending Machine



- Rational Rose
- Dia
- Star UML

Case Study : Hospital Management (for reception)



June 28 UML (Unified Modeling Language)

Unified: combine OO Methods (OOA, OOD, OOI)
Modeling: used to model system visually (System Visual Representation)
Language: Offers syntax and notations to model syst

Defn Q:- UML is a language for visualising, specifying and, cons and documenting the artifacts of software system.
↳ human-made object

1) UML for visualising (Diagram make rep.)
- Without visual representation of a system,
  a) Complex to represent the requirements.
  b) understand " "
  c)
- UML helps to model the system requirements, making easier for code mapping
- Communicate ides from client's end to developer's end

2) UML for specifying (Textual rep).
- UML addresses specification of object Oriented Analysis (OOD, OOI).
- Textual representation of system requirements.
- Specification make system unambiguous, precise and com

Rev. Eng
Coding follows
by In. Design part

② UML for Constructing
- UML helps to map executable code from various UML Diagrams.
- Forward engineering ← Documentation
- Reverse engineering
  A → D → C → Diag → Codes
  Codes → Diag.

diff of precise & ambiguity - UML

③ UML for Documenting
- Software = Executable code + All set of Artifacts
  ↳ Requirement Plans
  ↳ Analysis Plans
  ↳ Design Plans
  ↳ Code "
  ↳ Prototype "
  ↳ Test "

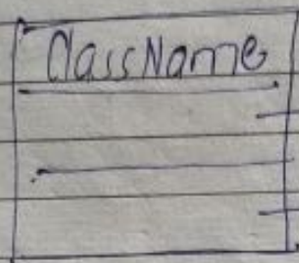
admin comp. doc.
- Document requirements for:
  - Client
  - Company - Developer
  - User

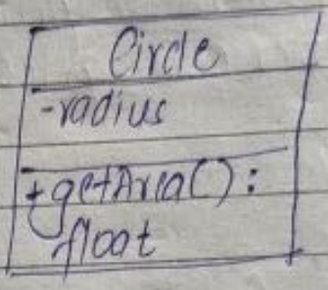
Ip ⟷ Op
↳ duration (TAT)

Importance of UML
→ Imp V.S. O.D
→ TAT (Turn around time) less / Market time reduced.
(Analysis to launching time)
→ Reduces cost
→ Manages complex architecture of system.
- Communication of ides is easier.

UML Notations

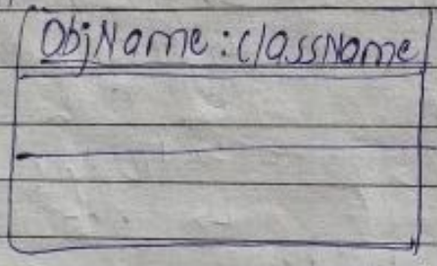
1) Class

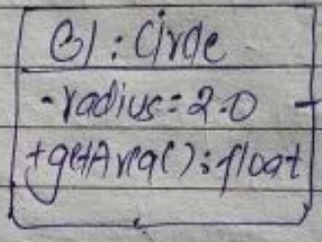




- private
+ public
# protected

2) Object

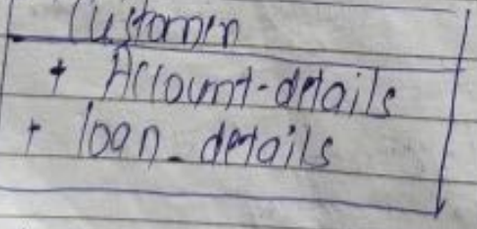




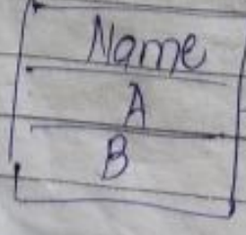
Attributes with values.

3) Package

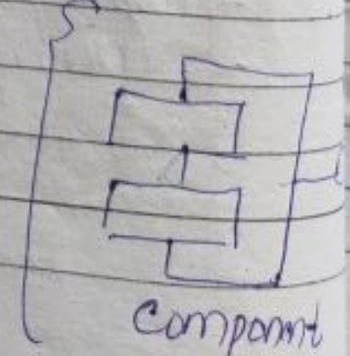
- grouping class, interface, component (collection of class, interface).



4) Interface

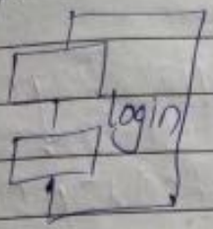


Interface

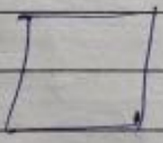


Component

5) Component - physical and replaceable module of system



6) Actor 

7) System Boundary 

8) Initial state
   - Start of process

9) Final State
   - ⊙ End of process

10) Relationship
   - Association ——
   - Dependency - - - - →
   - Generalization / Inheritance ——▷
   - Composition ◆——
   - Aggregation ◇——

(If College is del, student is independent can exist)

UML Diagrams

Structural Diagrams (Static aspects)
- Class Diagrams
- Object Diagrams
- Package Diagrams
- Composite Structure Diagrams
  - Component Diagram
  - Profile Diagram
  - Deployment Diagram

Behavorial Diagram → (Dynamic)
- Use Case Diagram
- Activity Diagram
- State Machine Diagram
- Interaction Diagram
  - Sequence Diagram
  - Communication
  - Timing Diagram
  - Interaction Overview Diagram

Structural Diagrams
- shows static aspects of system
- shows structure of system
- features that don't change with time

Behavorial Diagrams
- shows dynamic aspects of system
- shows behaviour / requirements
- features that change with

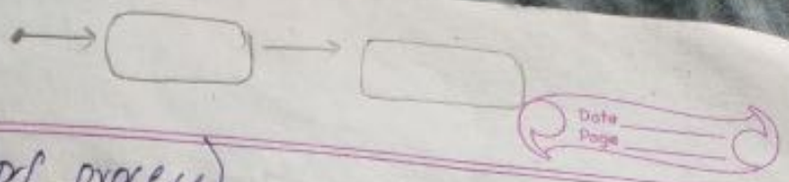
Activity Diagram → Illustrates decision points
→ illustrates concurrent actions

- Behavioural UML Diagram that shows flow of event from start point to finish.

Notations:

a) Initial Node / Start node (Start of process)

•



b) Final node (End of process)

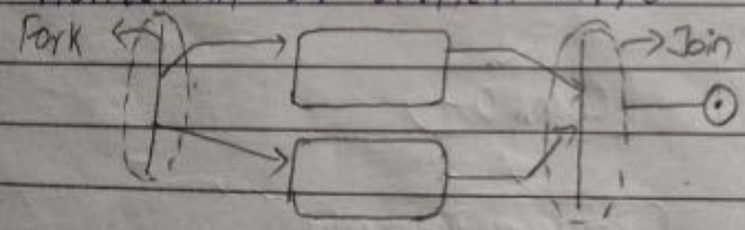
c) Activity / Action : sequence of events

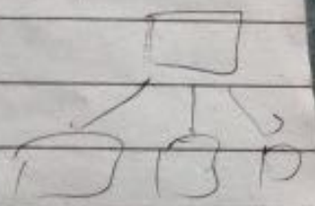
Rounded rect.

d) Control Flow
flow of control from one action to another.

e) Decision and Merge Nodes

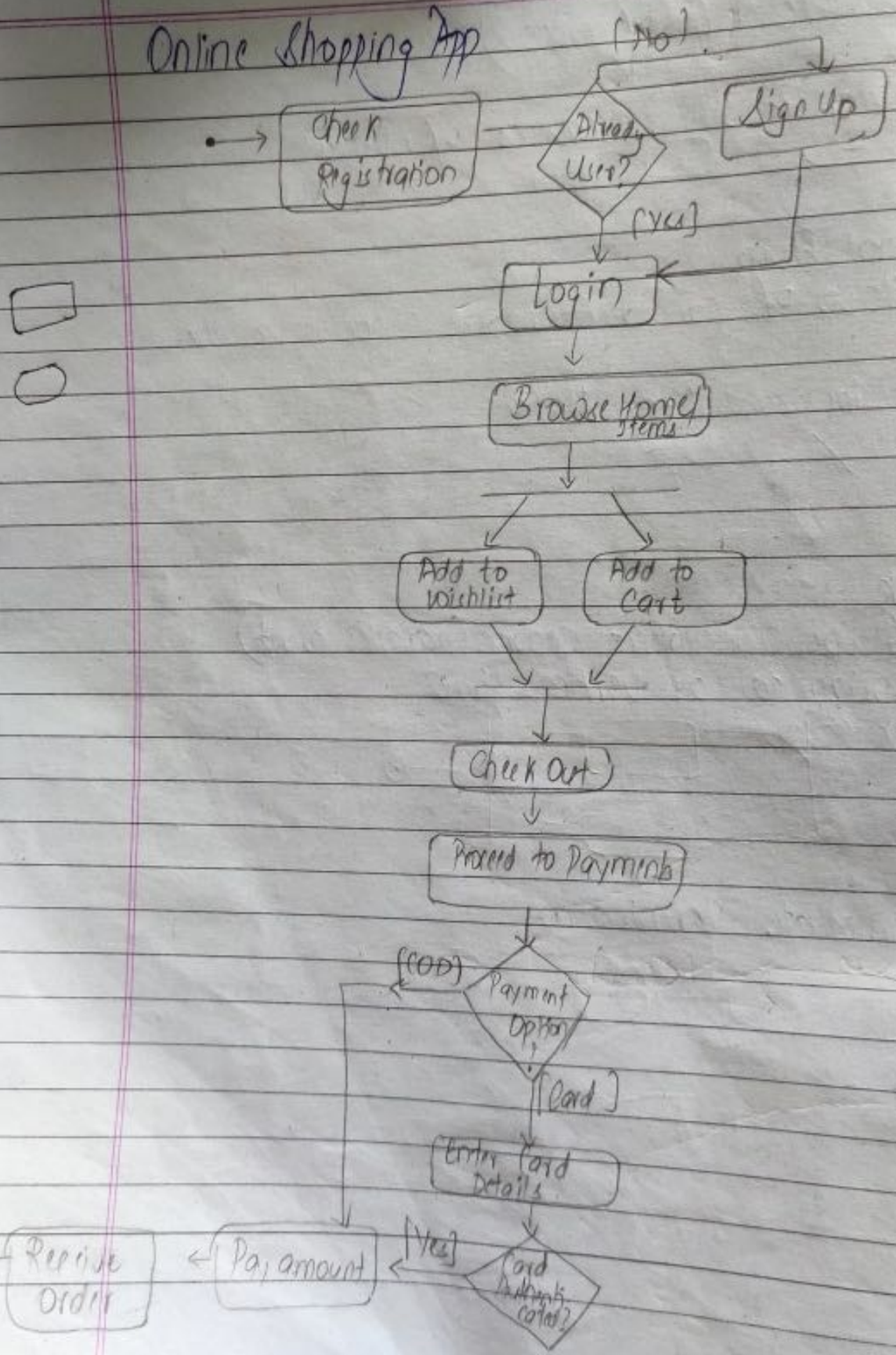
f) Fork
shows concurrent actions. (parallel events)
- horizontal or vertical line

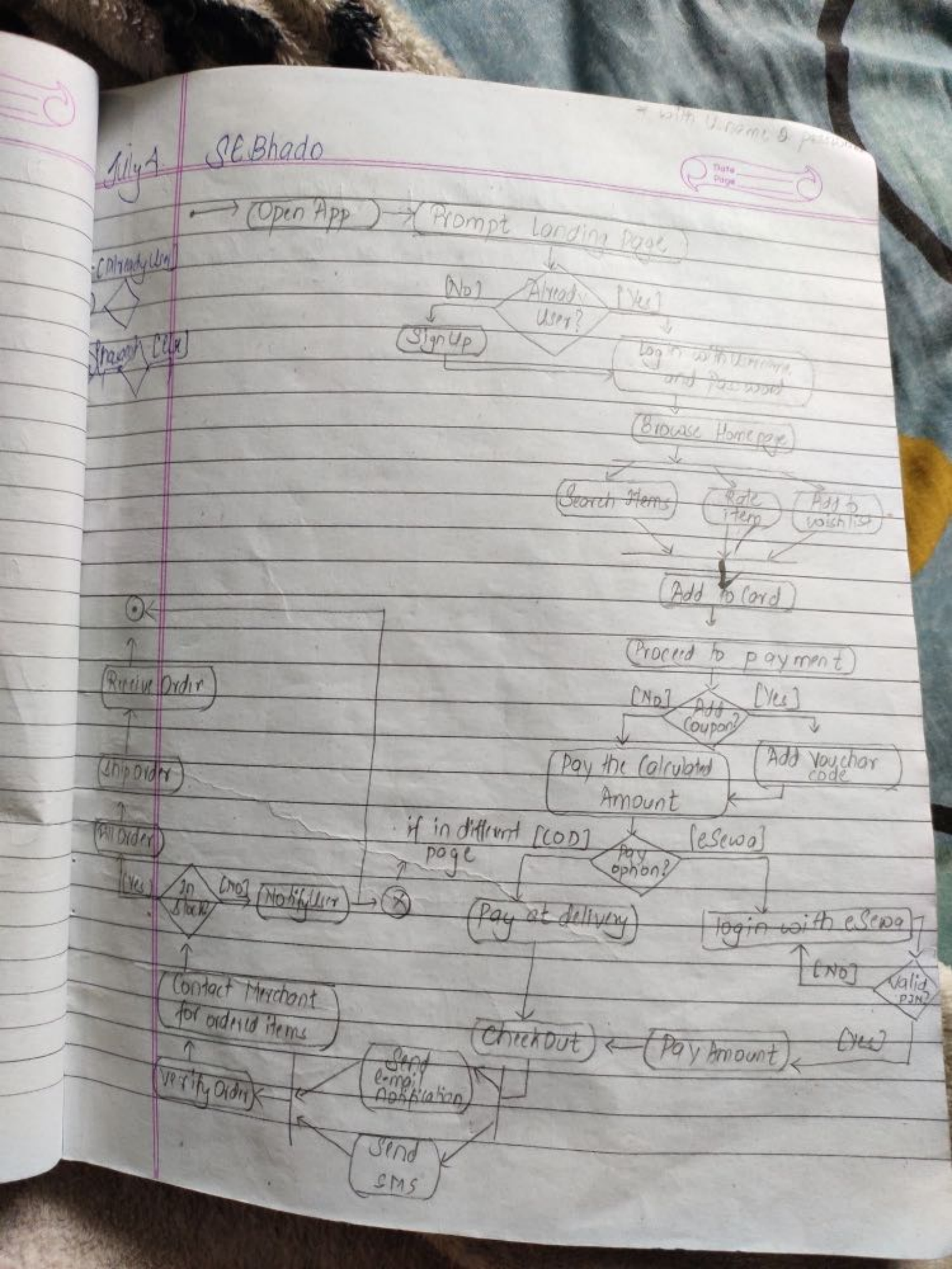




g) Immediate Termination

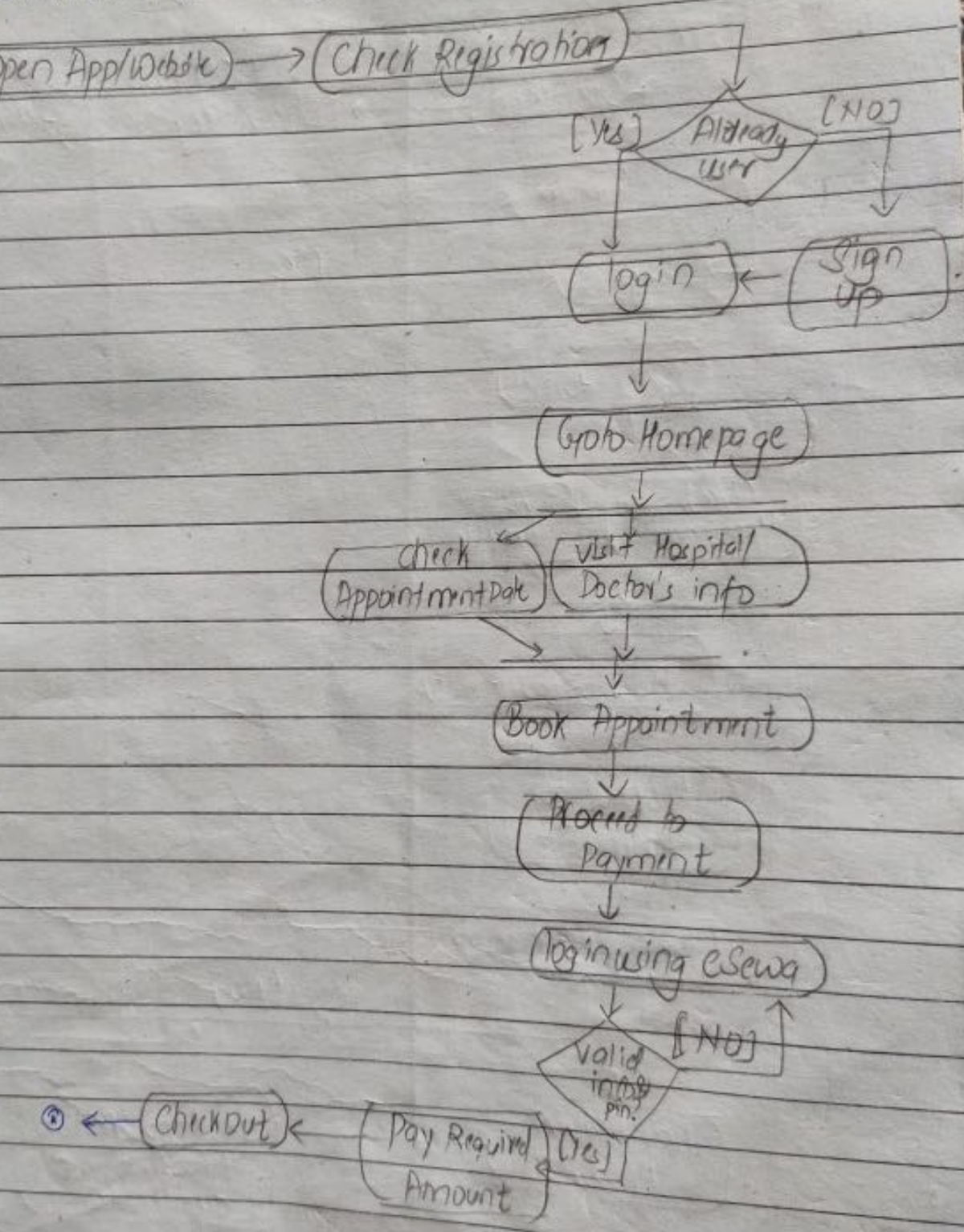
Online Shopping App




Open App
Prompt Landing Page
Already User?
[No]
[Yes]
Sign Up
Log in with Username and Password
Browse Homepage
Search Items
Rate Item
Add to Wishlist
Add to Cart
Proceed to payment
Add Coupon?
[No]
[Yes]
Pay the Calculated Amount
Add Voucher code
Pay option?
[COD]
[eSewa]
if in different page
Pay at delivery
login with eSewa
Valid PIN?
[No]
[Yes]
Pay Amount
CheckOut
Send e-mail Notification
Send SMS
Contact Merchant for ordered Items
Verify Order
Notify User
In Stock
[No]
[Yes]
Fill Order
Ship Order
Receive Order

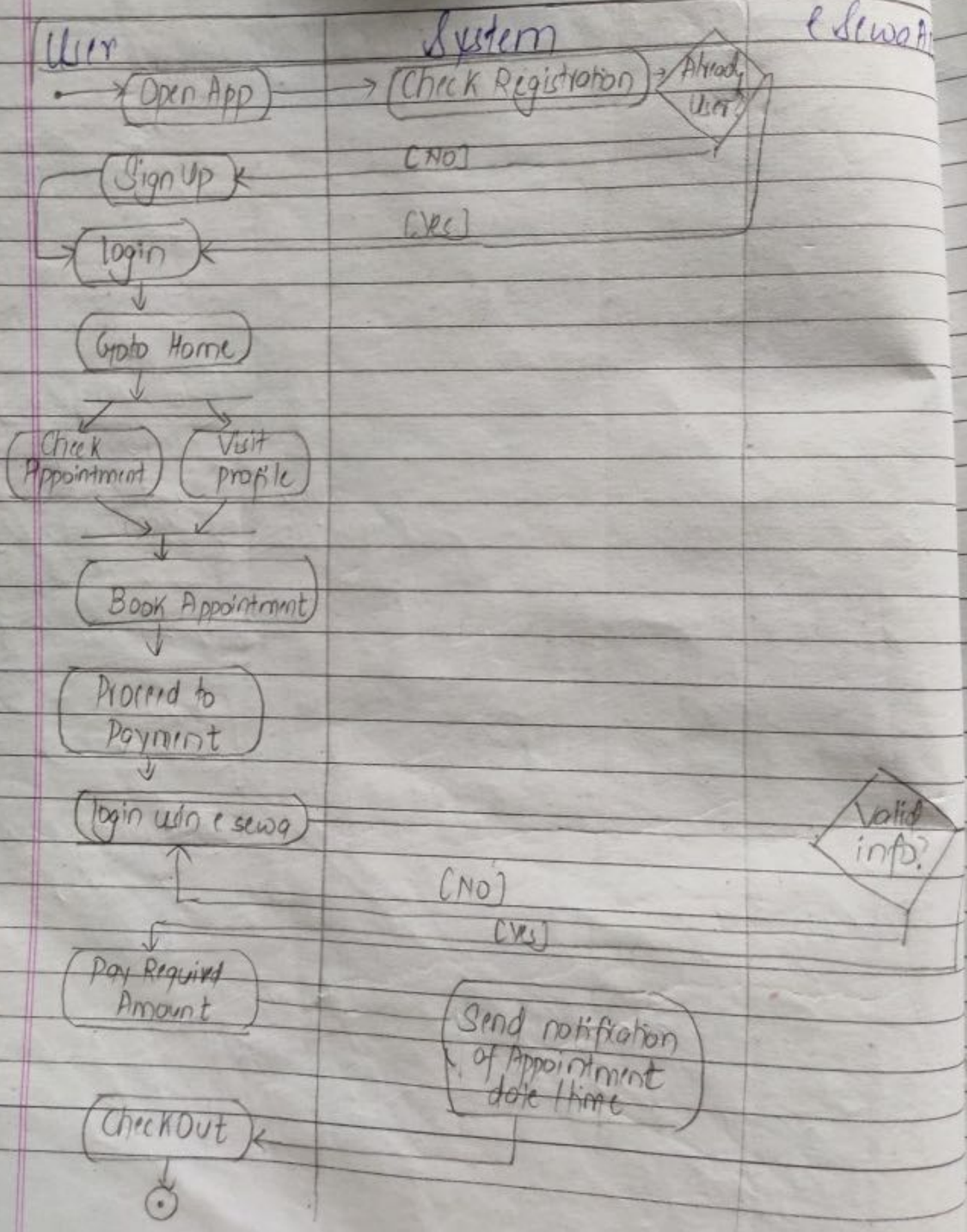
Activity Diagram

1 Book Appointment (Online) : H-Booking



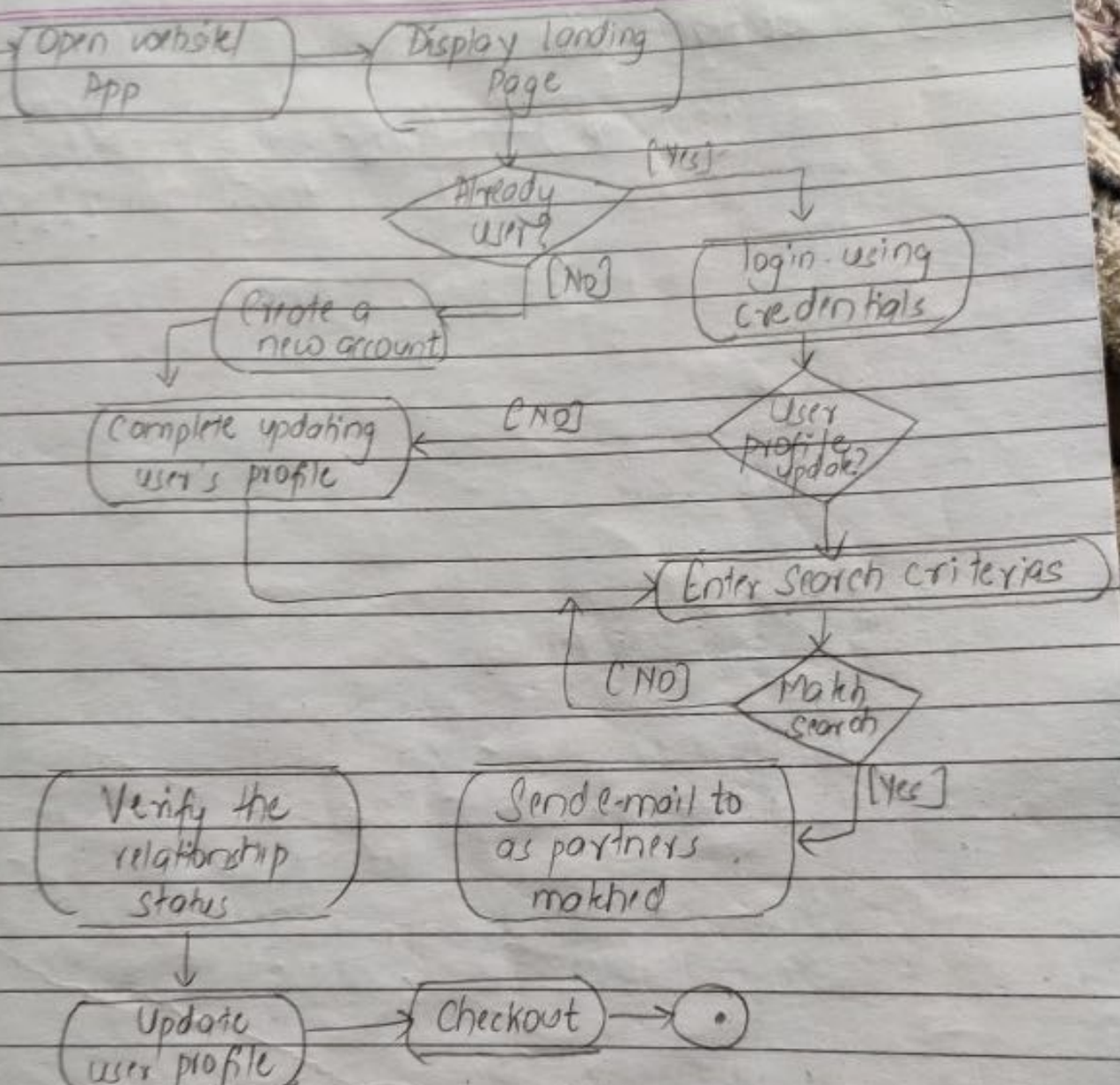
July 5 Using Swimlanes

a) User b) System c) eSewa API



Σk $\frac{K \times (K+1)}{2}$ $K+$

Q2:-



3 ways to apply UML (Short Note)

i) UML as Sketch
ii) UML as Blueprint
iii) UML as Programming language

3 Perspectives to Apply UML
i) Conceptual Perspective
ii) Specification Perspective
iii) Implementation Perspective

↓ level of abstraction

UML as Sketch
- Draft diagrams of UML (white boarding copies)
- Informal and incomplete diagrams

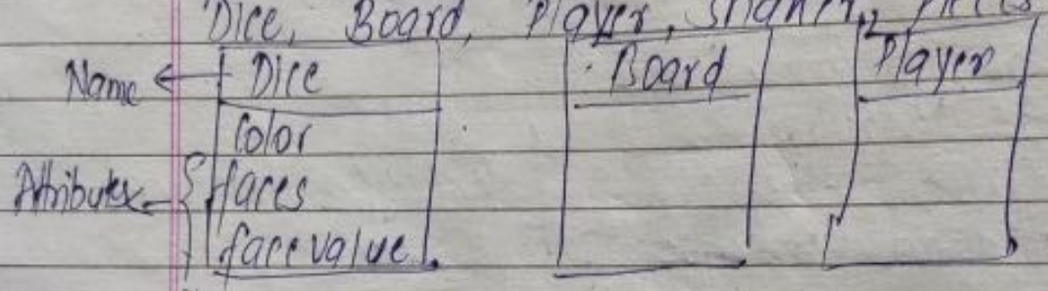
- UML as Blueprint
- Map for code / System
- Formal and Complete
- Blueprint for - forward Engineering
  - Reverse

UML as Programming Language
- UML as programming language because it provides complete executable specification of system requirement.
  working code
- Shows classes and their specification and implementation of system
- Describes functionalities / requirements.

Page No.

Conceptual Perspectives
- illustrated by Domain model
  ↳ Describes concepts (idea, thing/ object)

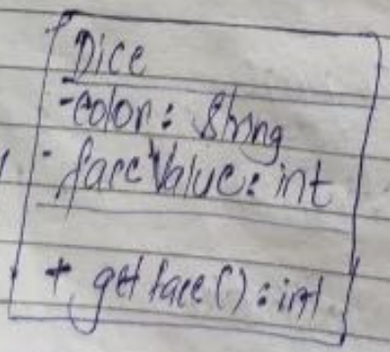
for ludo,
Dice, Board, Player, Shaker, Pieces

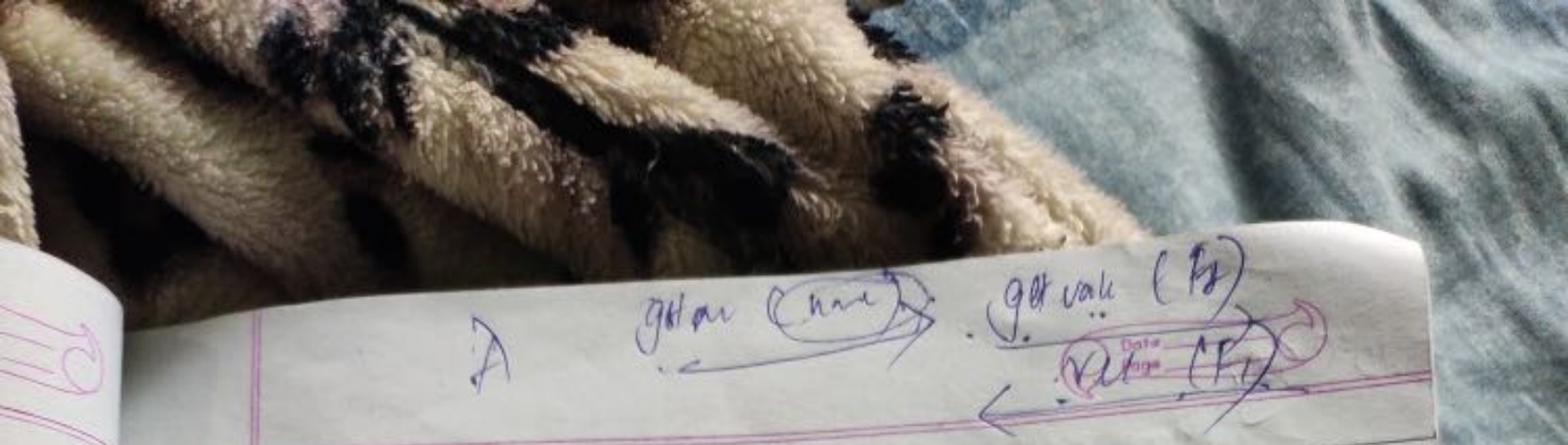


- gives the overall representation of system using concepts / Conceptual class.

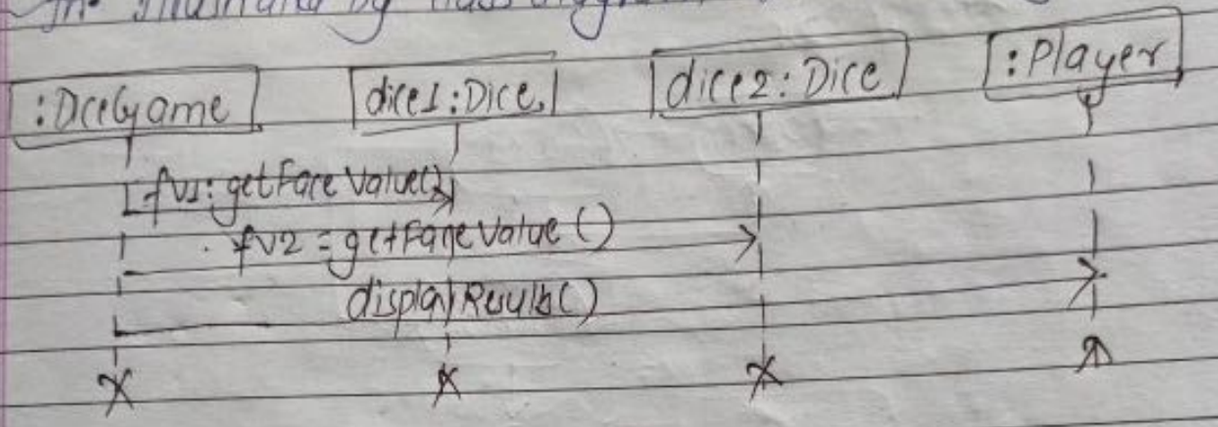
Specification Perspectives
- illustrated by class Diagram
  ↳ describes various software classes in system.
- define Interface. public Interface SS
  void draw(); };

Interface
↳ class without implementation of function





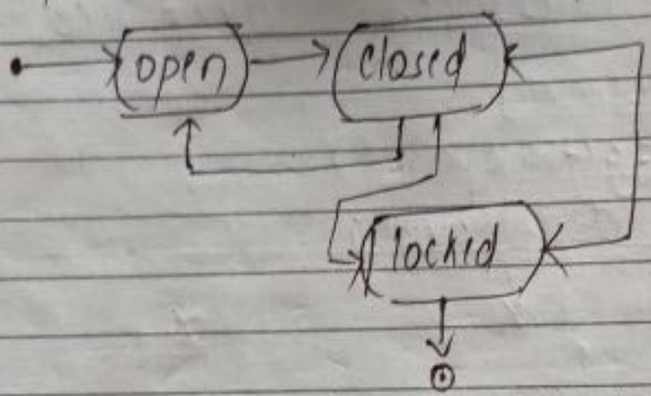
Implementation perspective
- describes implementation of Specification Perspective
- Illustrated by class diagram + Interaction diagram



10th July

State Machine Diagram / State Diagram / State Transition Diagram

States of 'Door'

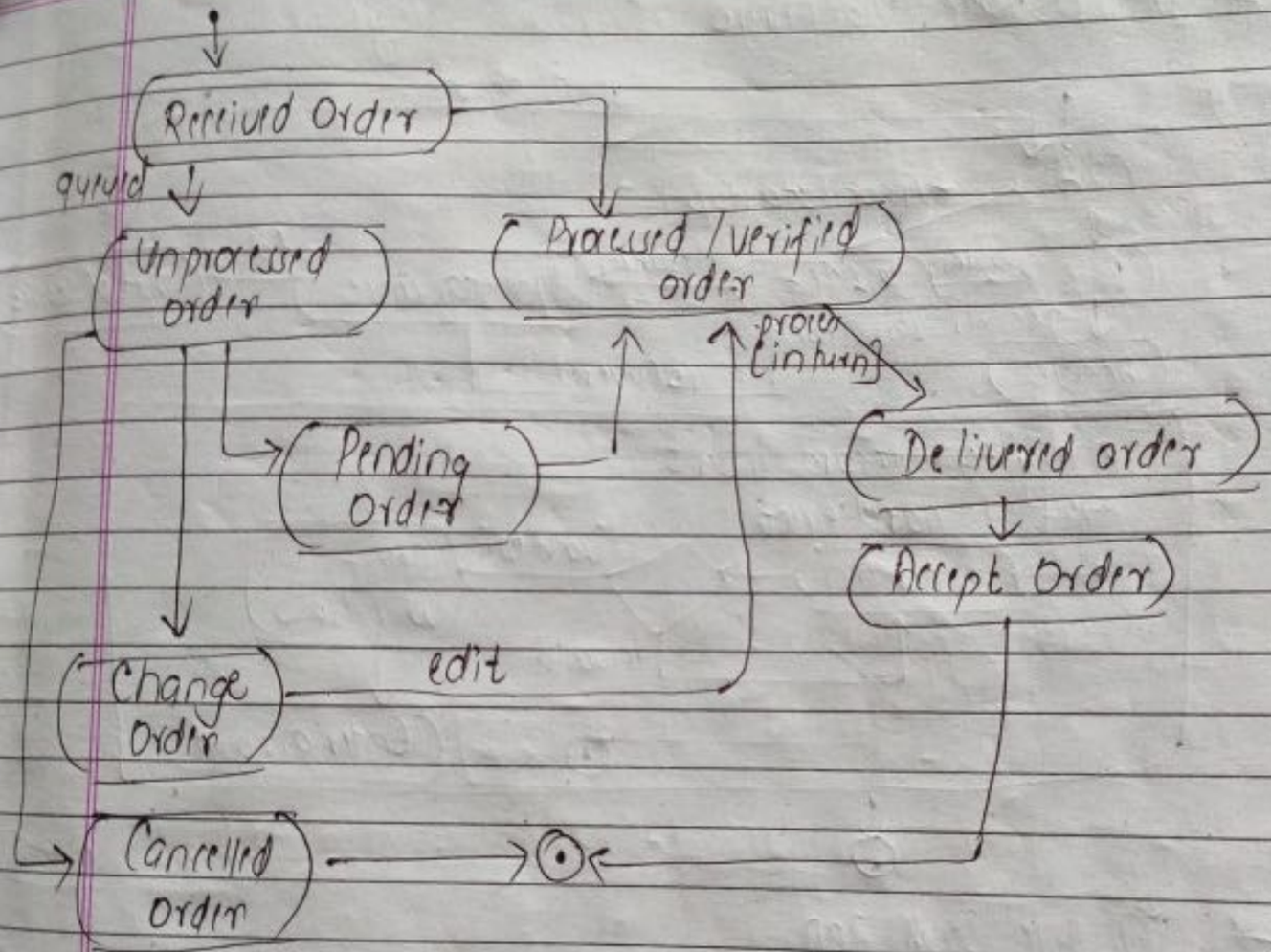


- Behavorial UML diagram that shows various states & transition between them

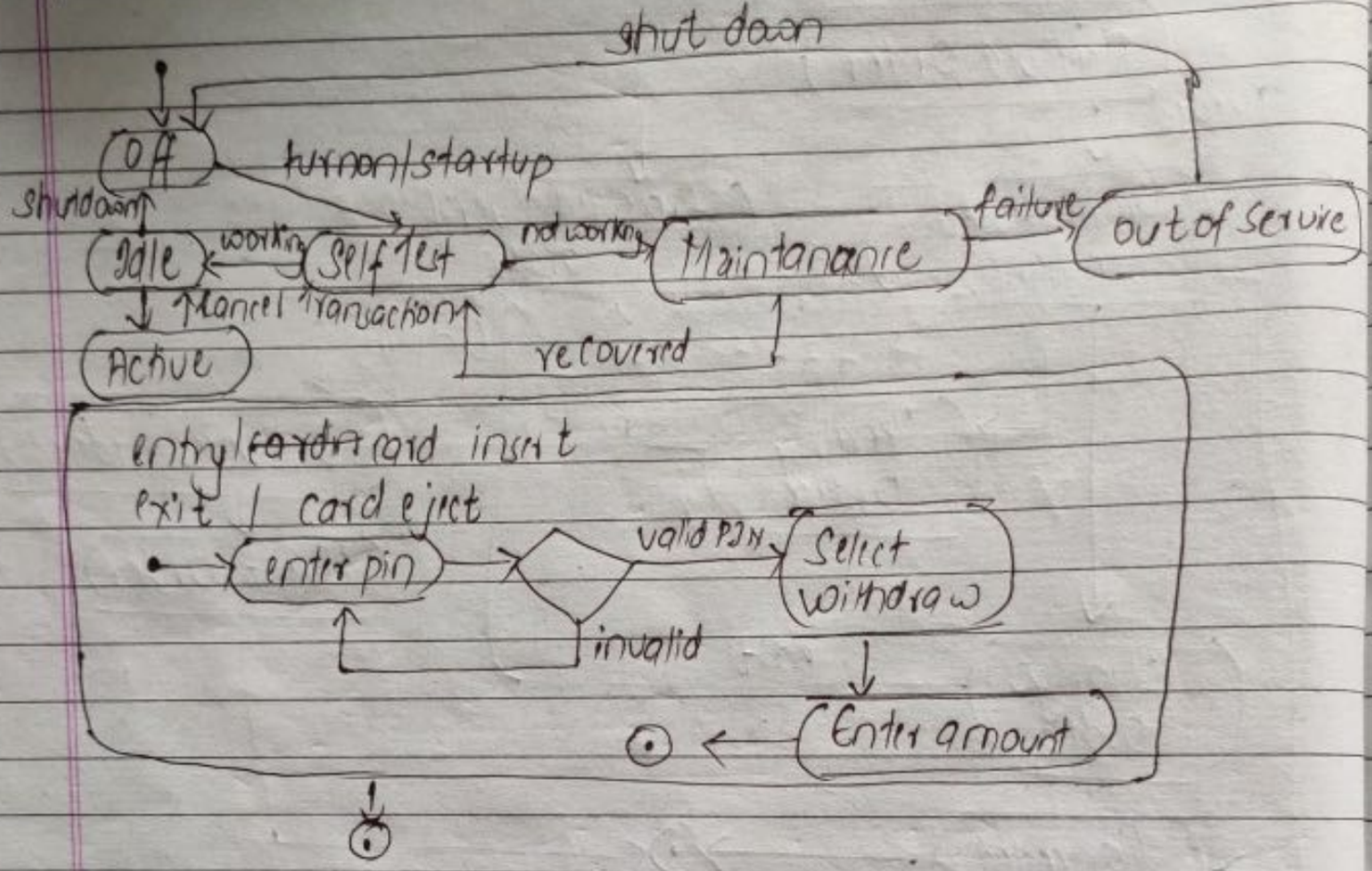
States :- Condition or constraints in the lifecycle of an object in which the constraints holds the object executes an activity or waits for it.

'Door' States → notation similar to activity Diagram

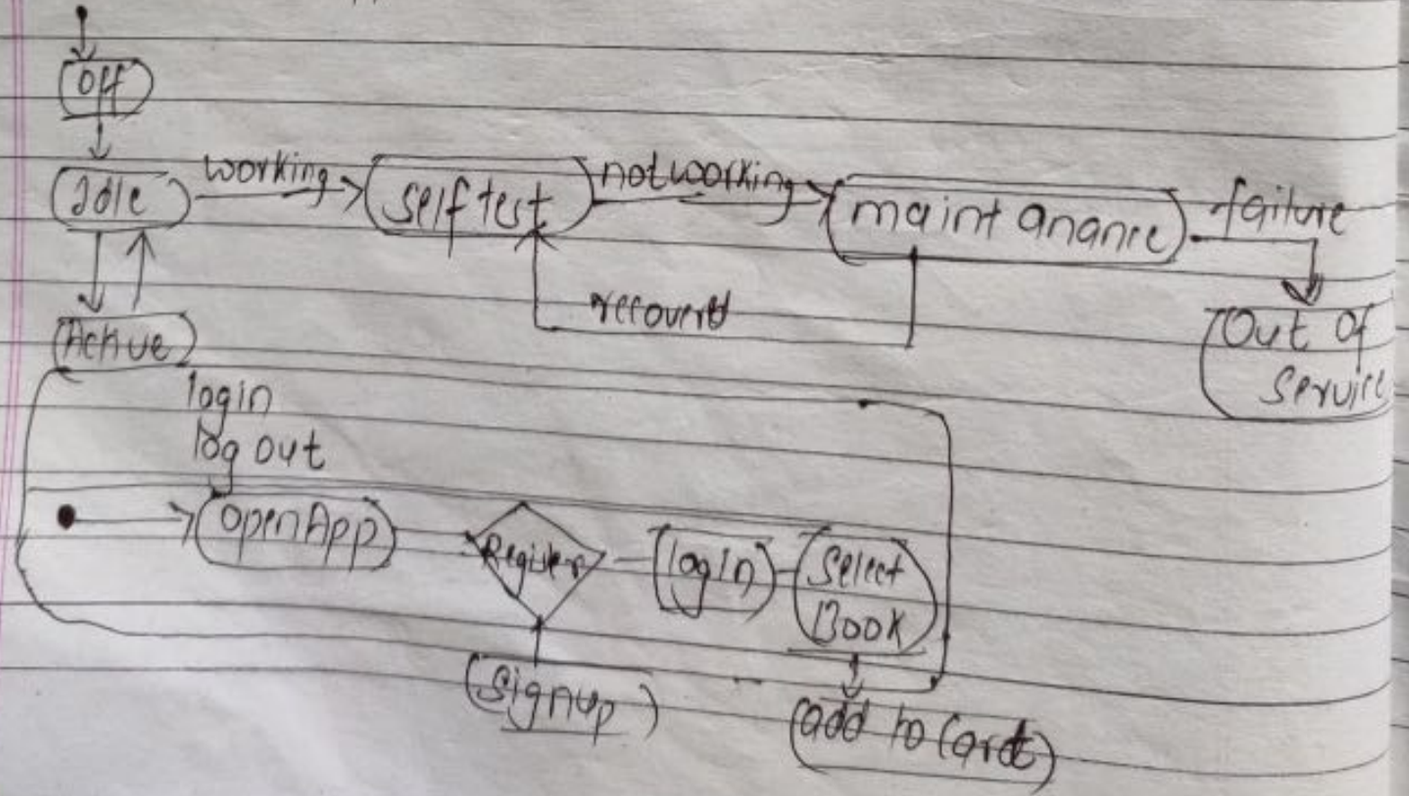
"Order"
- Received order
- Delivered order
- Unprocessed order
- Processed / verified order
- Reject order / cancelled order
- Accept order
- Pending order
- Change order


Received Order
queued
Unprocessed order
Processed / verified order
proceed in turn
Pending Order
Delivered order
Accept Order
Change Order
edit
Cancelled Order

ATM state Machine



Online Book app



11-10-22

Date
Page

Interaction Diagram

describes interaction between objects,

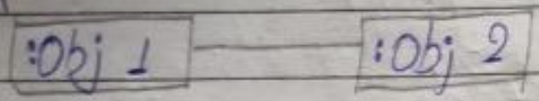
4 types:

a) Sequence Diagram
b) Communication / Collaboration Diagram
c) Interaction Overview Diagram
d) Timing Diagram

Parameter	Sequence Diagram	Communication Diagram
Def^n:	Interaction diagram that emphasizes on time order.	Interaction diagram that emphasizes on structural Organisation of objects.
	call flow ring of messages sequence (Message passing)	
Format:	Fence Format	- Graph or Network Format
Objects placement:	New objects are added do right	New objects can be placed anywhere
Strength:	- Clearly shows sequence of message passing - Simple, and has large set of notations	- Space flexible / Economical
Weakness:	Preferred for documentation	
Weakness:	Consumes horizontal space	- difficult to find / track order of message passing

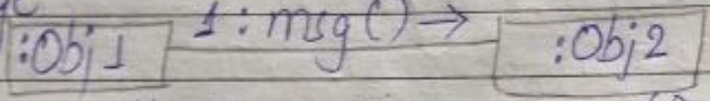
Notation in Communication Diagram

1) Link



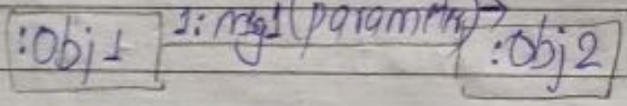
- Connection path between objects

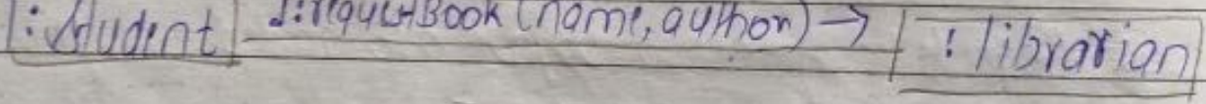
2) Message



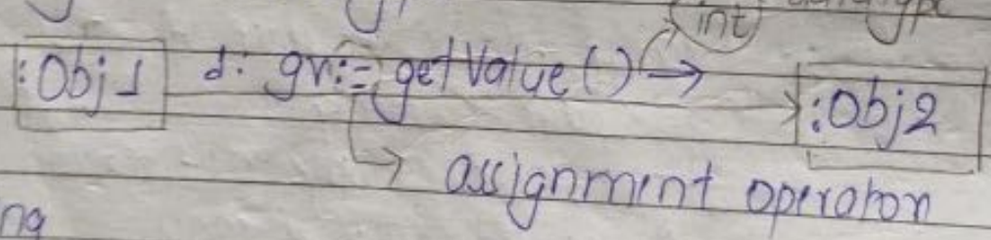
Number : Message name () Arrow dir?

3) Illustrating Parameters

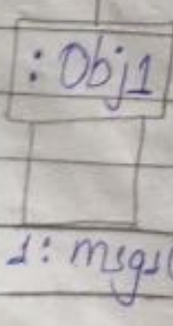


eg. 

4) Illustrating a Return type



5) Illustrating Message to self or this



6) Creation of object / instances

(i) :obj1 — 1: create(myobj2) → :obj2

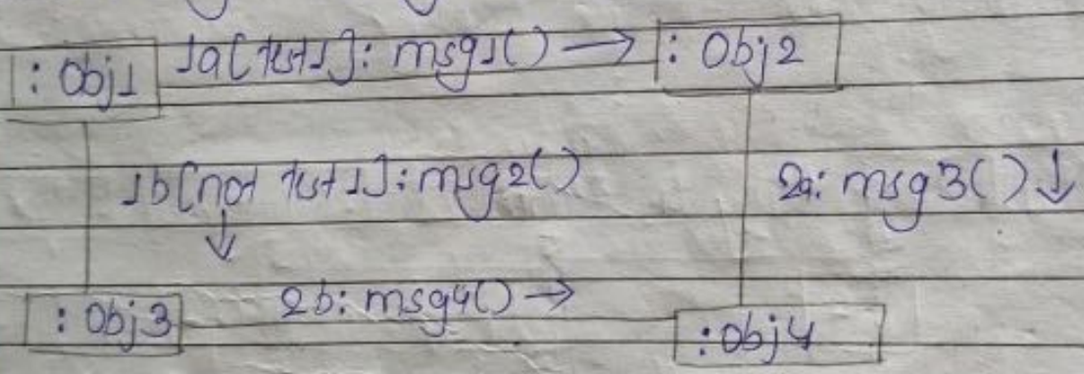
(ii) :obj1 — 1: <<create>> make(myobj2) → :obj2

7) Conditional msg
(represented inside square bracket)

:Obj1 — 1*[cond^]: msg1() → :obj2

eg:

8) Illustrating Mutually Exclusive Conditions



Either 1a or 1b executes.

9) Illustrating Iterations / looping

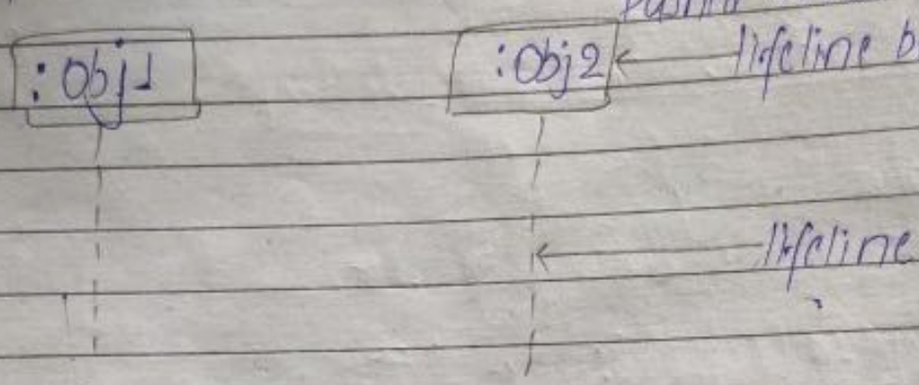
:obj1 — 1: msg() → :obj2 // without loop.

:obj1 — 1*[i:1..n]: msg1() → :obj2

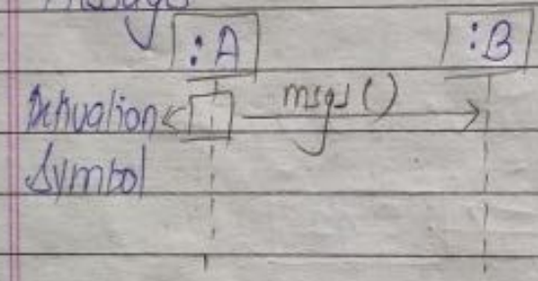
eg: :User — 1*[i:1..3]: retry() → :System

Notation in Sequence Diagram

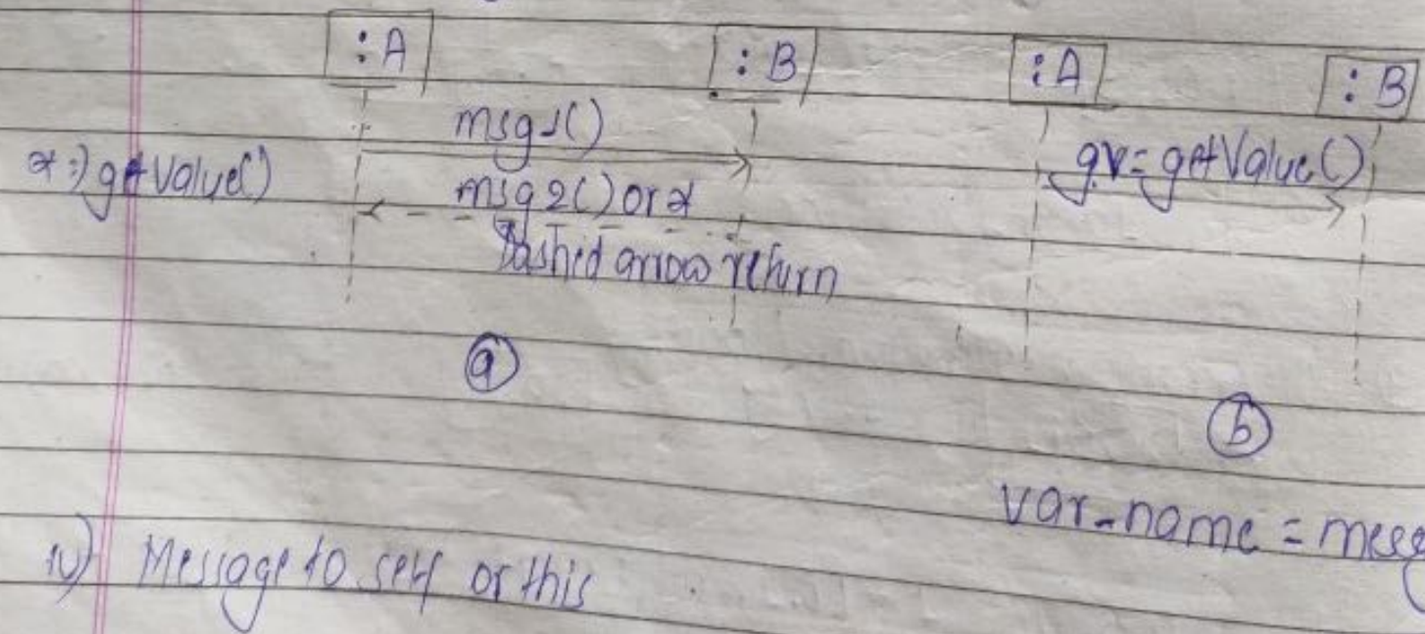
i) Lifeline Boxes and lifelines



ii) Messages

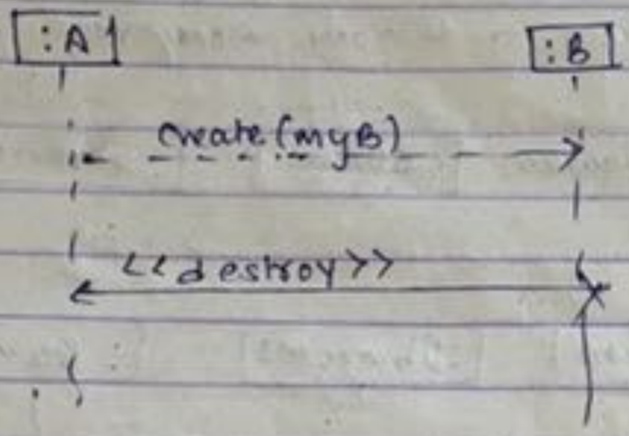


iii) Illustrating Reply or Return



iv) Message to self or this

6. object Destruction

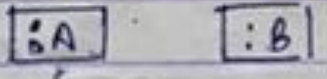


#

7. Design Frame

alt	mutually exclusive fragments
loop	for looping fragment
opt	option fragment ('if')
par	Parallel fragment / Concurrent Action
region	Critical action where only one thread can run

- frame Contain label and guard.

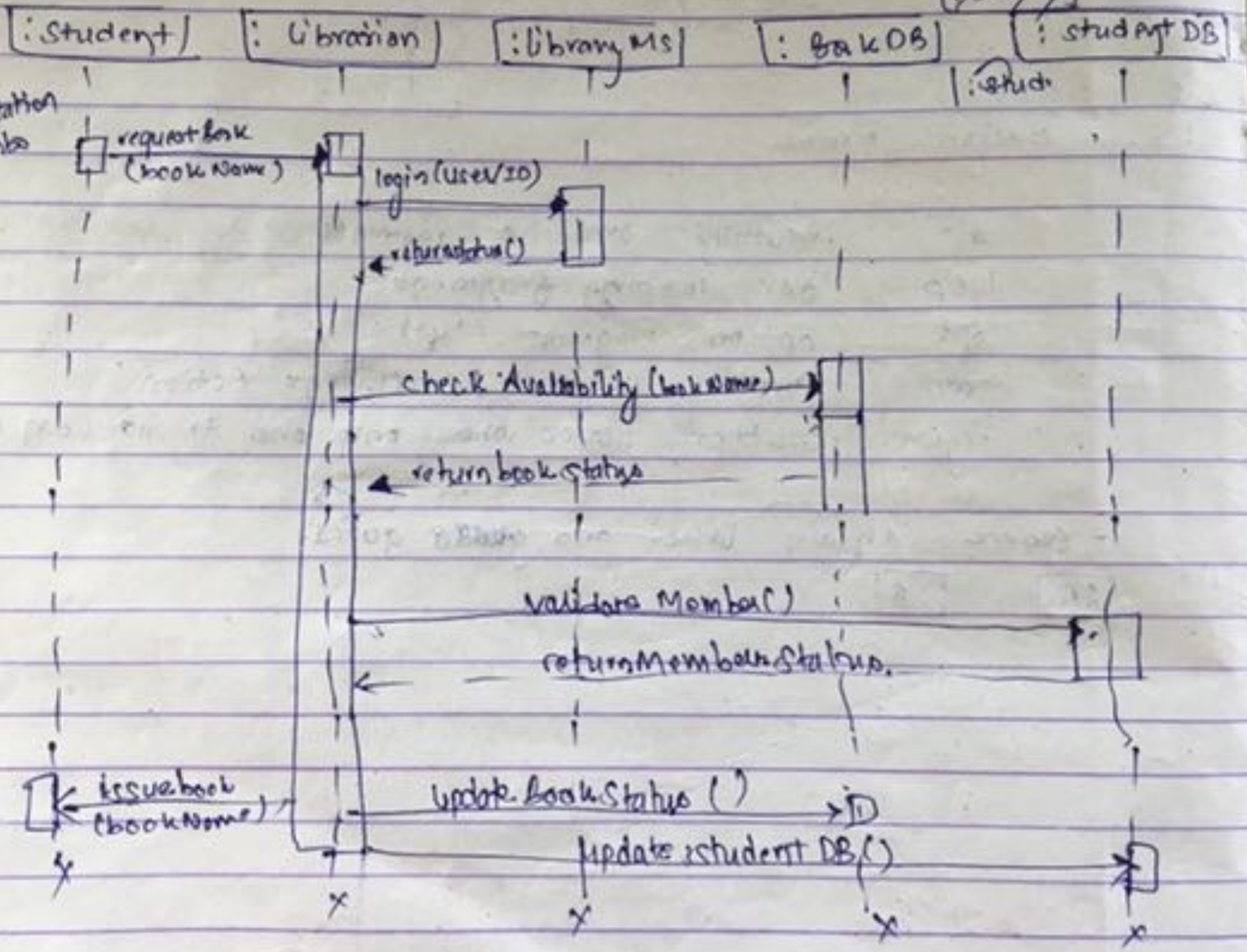


12th July 2019

Case Study

Issuing Book from a Library Management System.

a) Draw Sequence diagram



26th July 2019

Interaction Diagram

~~#~~ - Sequence "

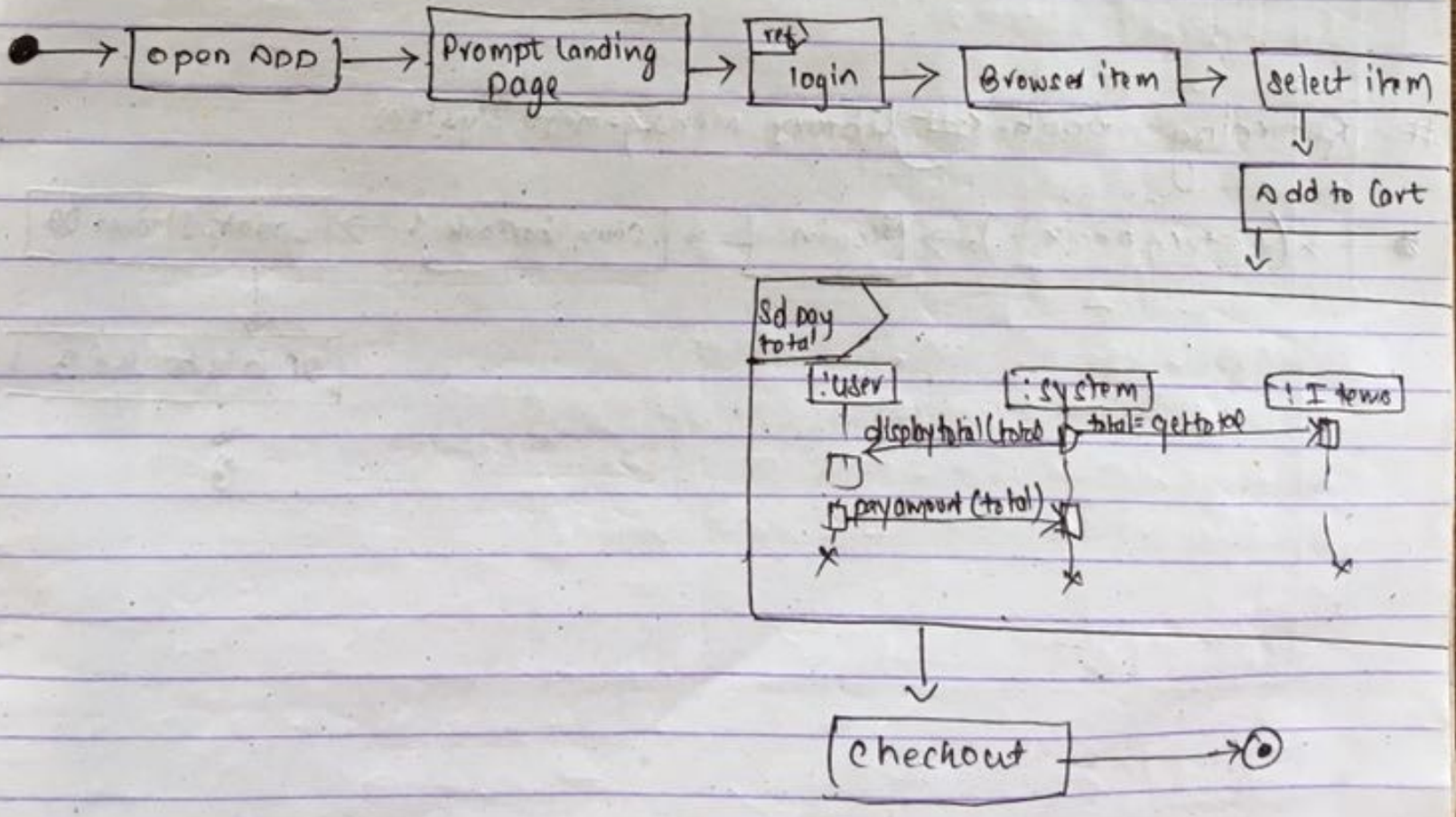
~~#~~ - communication "

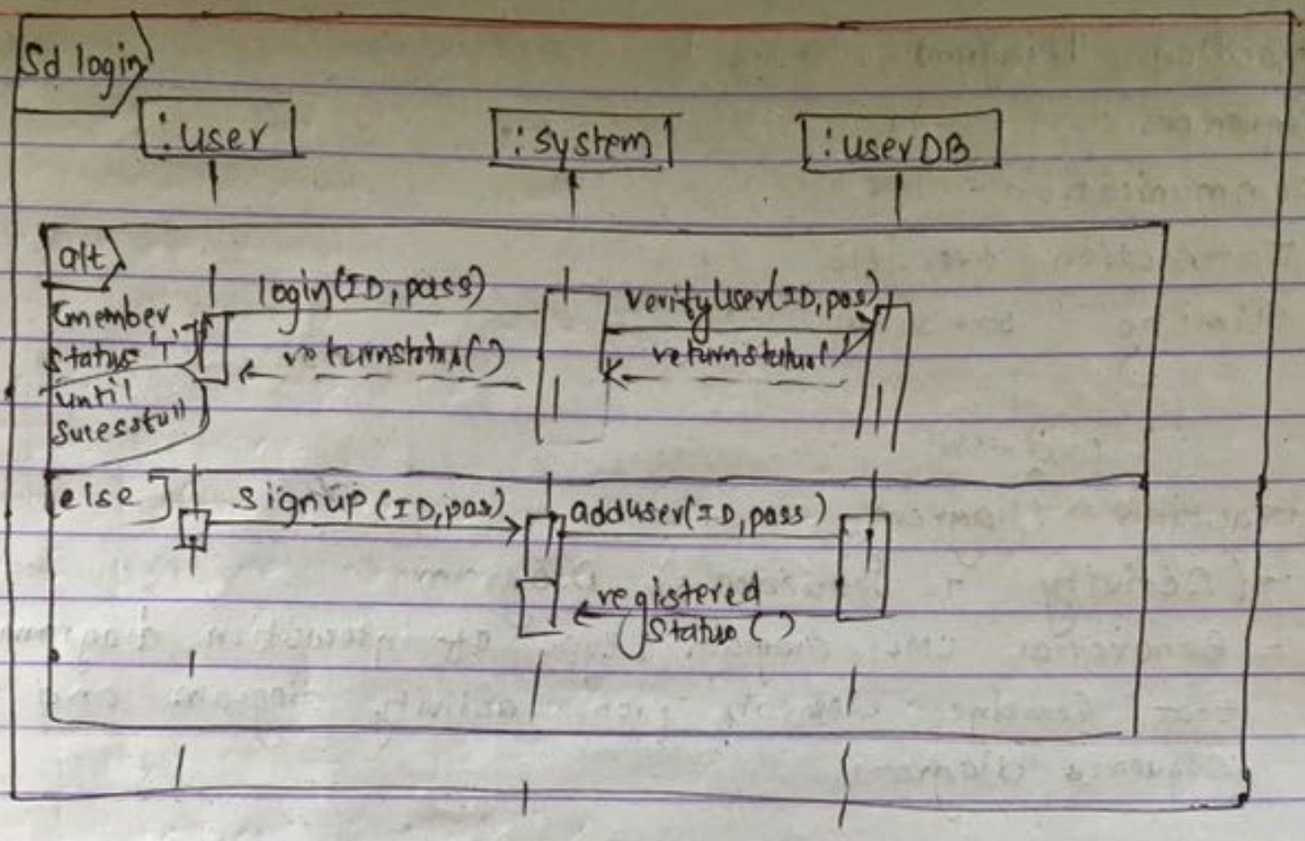
Short notes - Interaction overview "

- Timing overview "

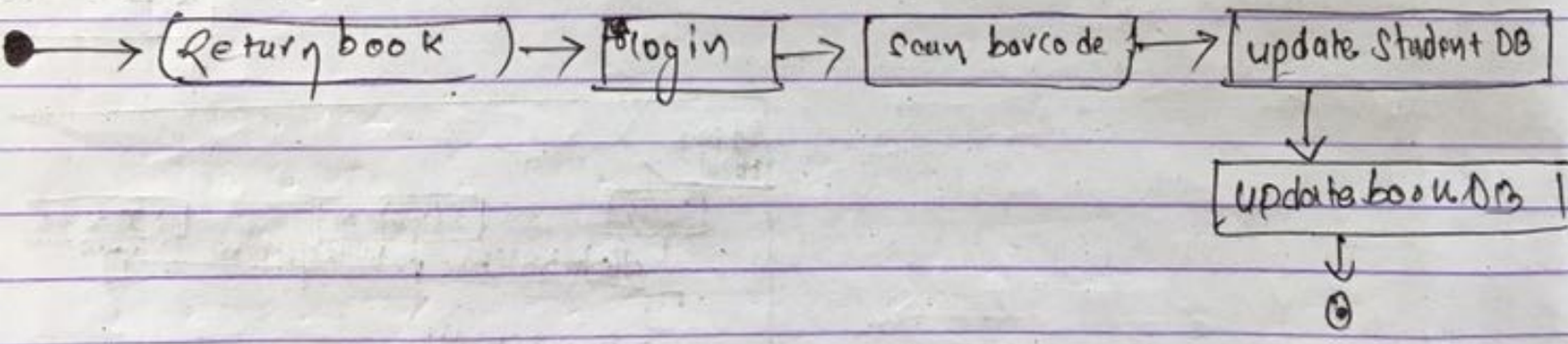
Interaction overview Diagram

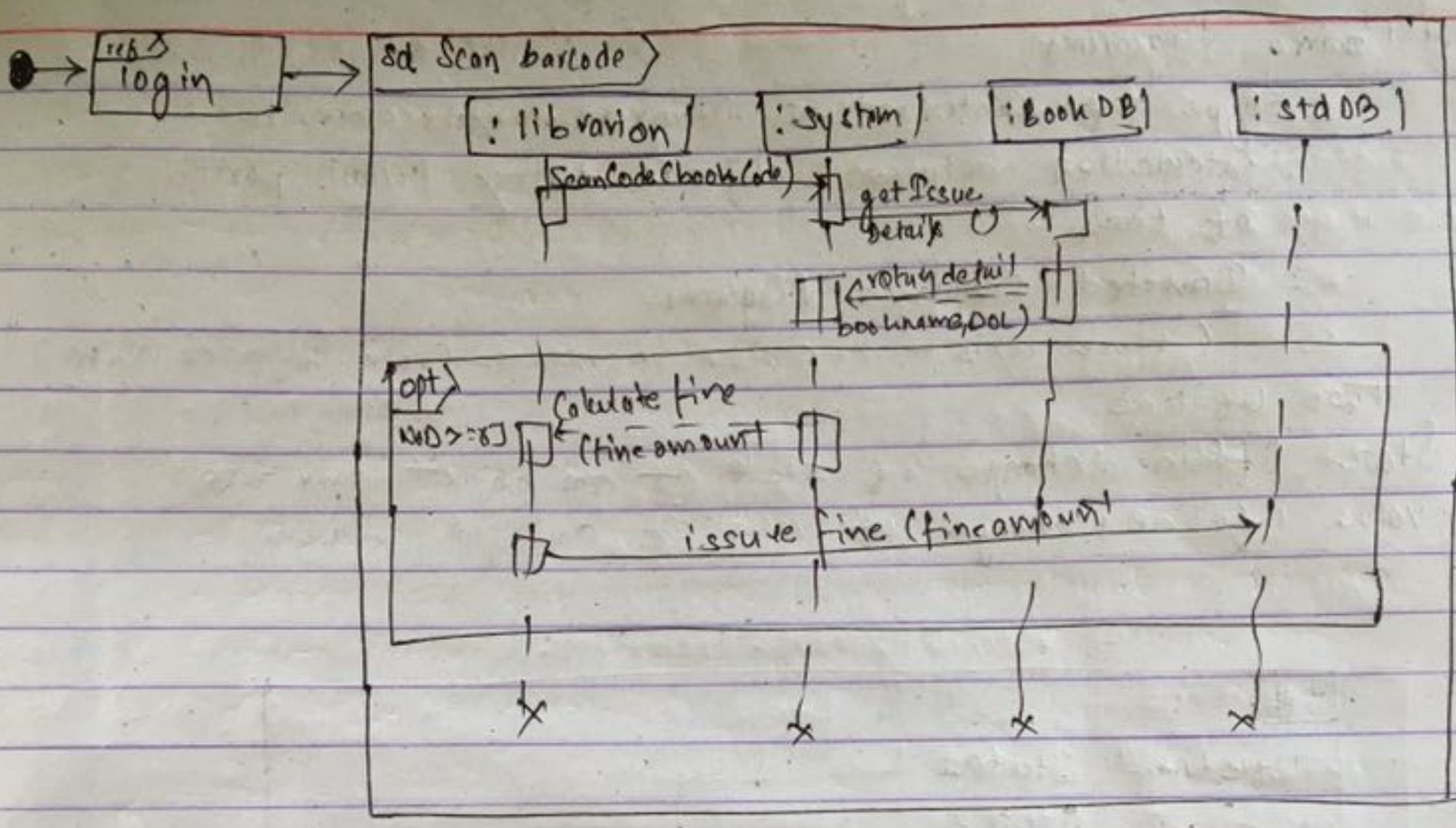
- (Activity + Sequence) Diagram
- Behavorial UML diagram / type of interaction diagram that combines elements from activity diagram and sequence diagram.





Returning Book. ~~###~~ Library Management System




ref
login
sd Scan barcode
: librarion
: system
: BookDB
: stdDB
ScanCode(bookCode)
getIssue
Details
return detail
bookname, DOL)
opt
[NOD > 8]
Calculate fine
(fineamount
issue fine (fineamount

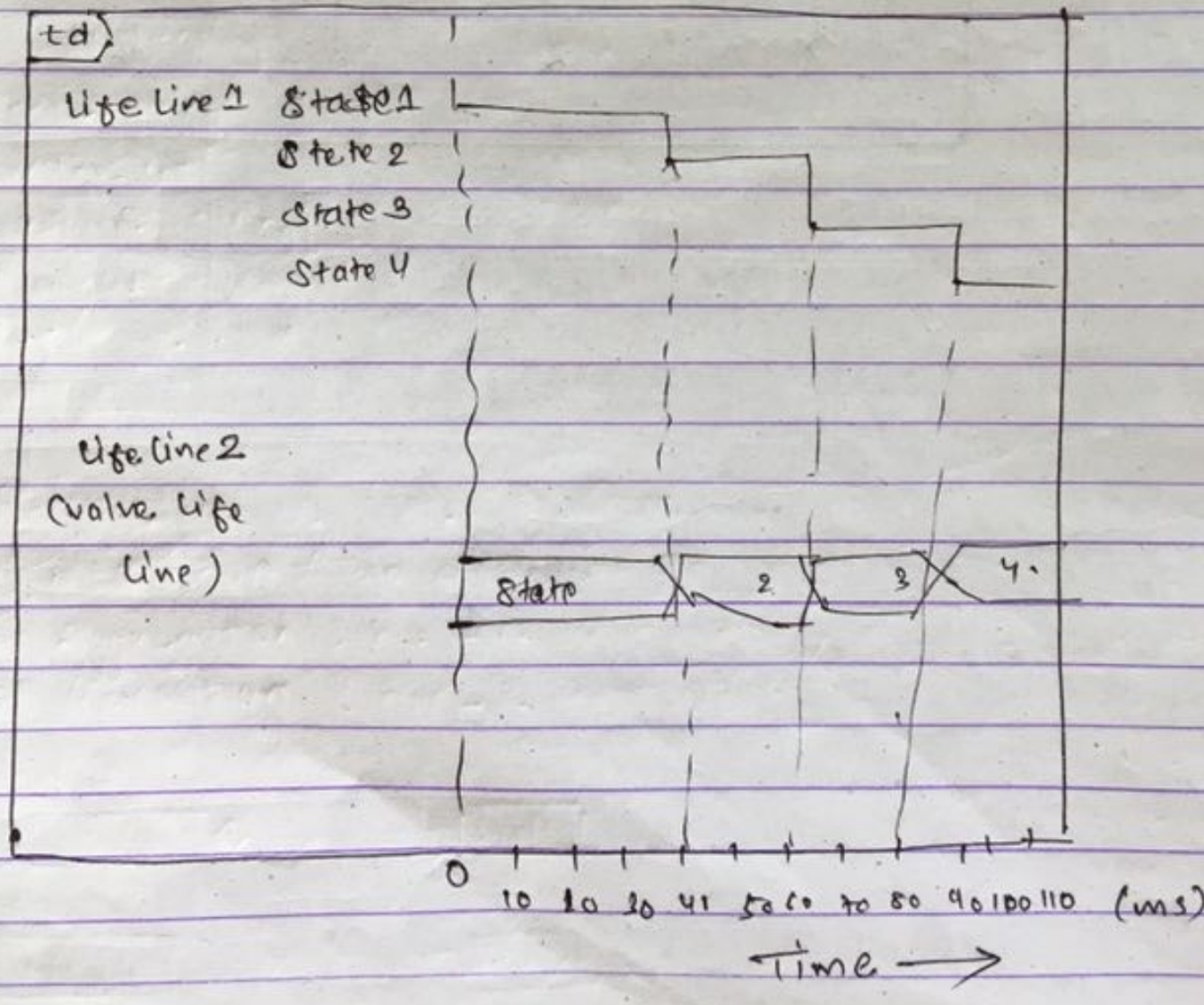
X-axis as
Y axis
s.d.

Timing Diagram

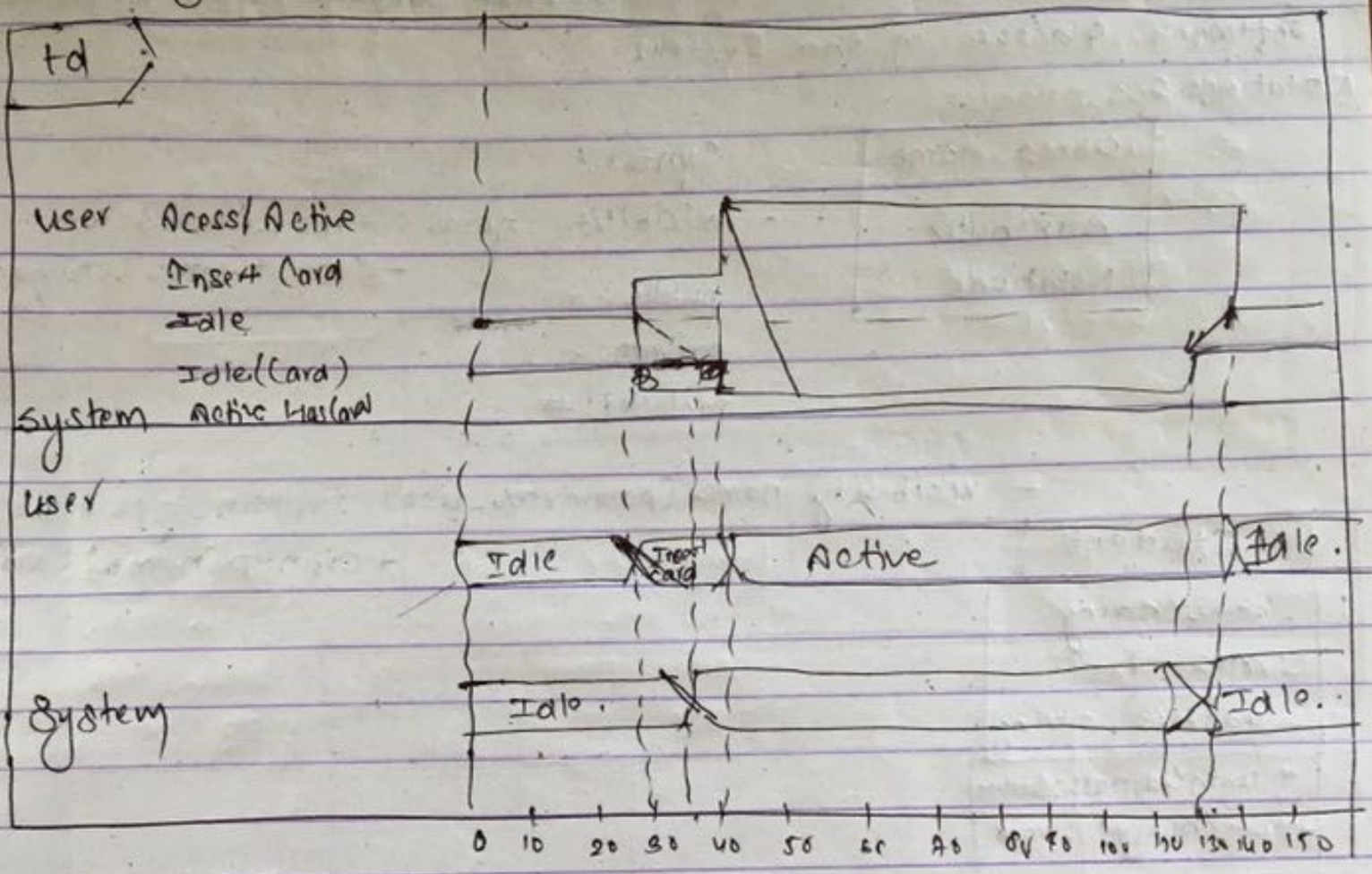
- Type of Interaction Diagram that describes interaction between objects during certain period of time.
- Inverted Sequence Diagram
  (time axis - x-axis. increase from left to right)

Two lifeline

- State lifRline : change of state of an object over time
- value lifeline : change of volve of an object over time



Timing Diagram : ATM System

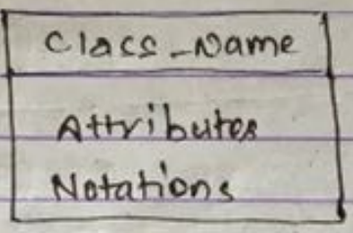


6th Aug 2019

Structural UML Diagram / Static diagram
1) Class Diagram — feature that don't change with time

Structure UML Diagram that represents various software classes in the system.

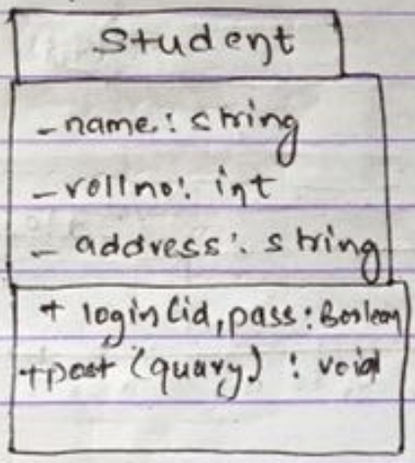
Notations :



Syntax:-
- visibility name : type_expression
-Student Name : string

public : +
private : -
protected : #

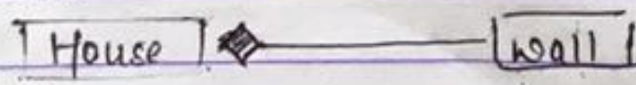
- visibility name (parameter_list) : return_type express
+signup(name) : void



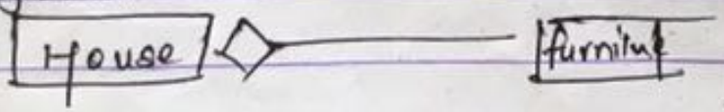
Relationship in Class_Diagram

1) Association (Two classes are directly associated (linked) (Notation: ——— (solid line)
2) Aggregation : has-a relationship ◇ whole part rel^n
3) Composition : has-a relationship. ◆

Composition

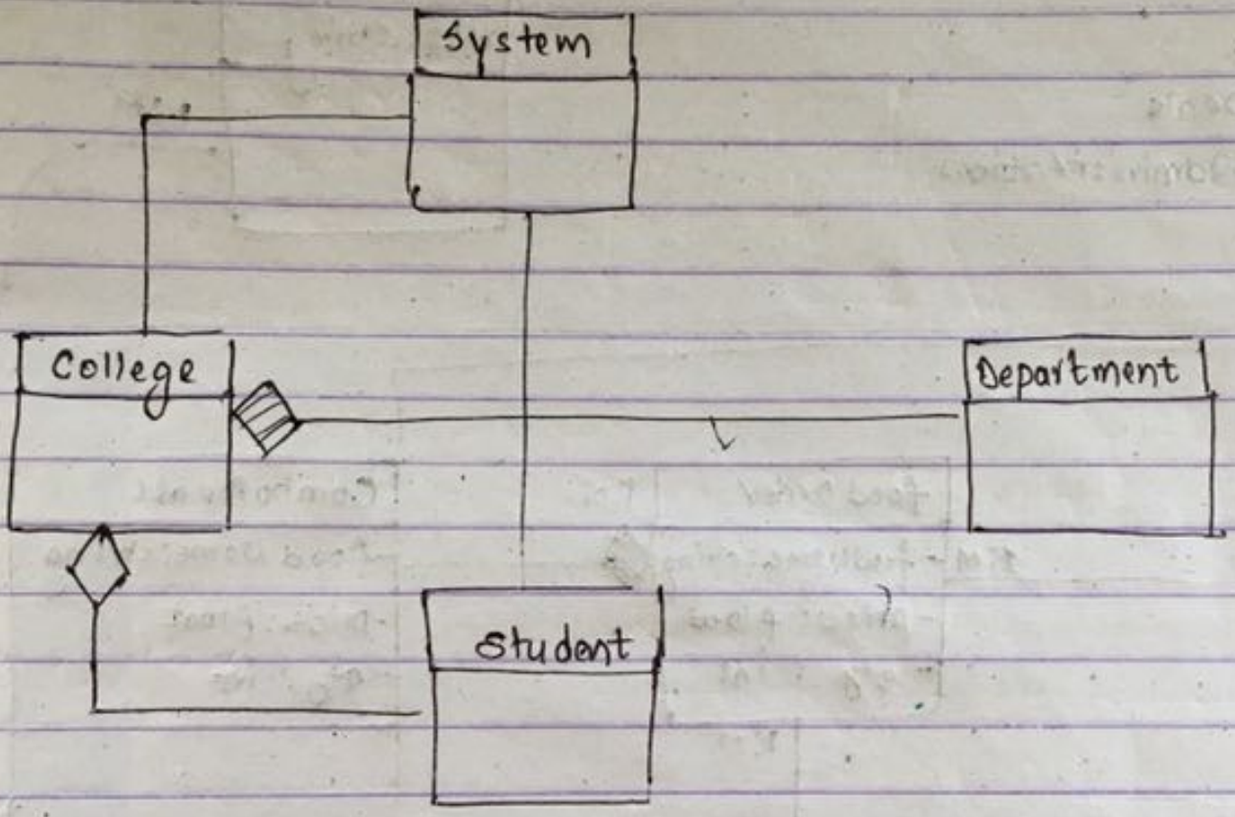


Aggregation



Aggregation : Child_class can independently exist without parent_class

Composition : cannot independently exist.



Multiplicity / Card.

1) one-to-one (1 : 1)

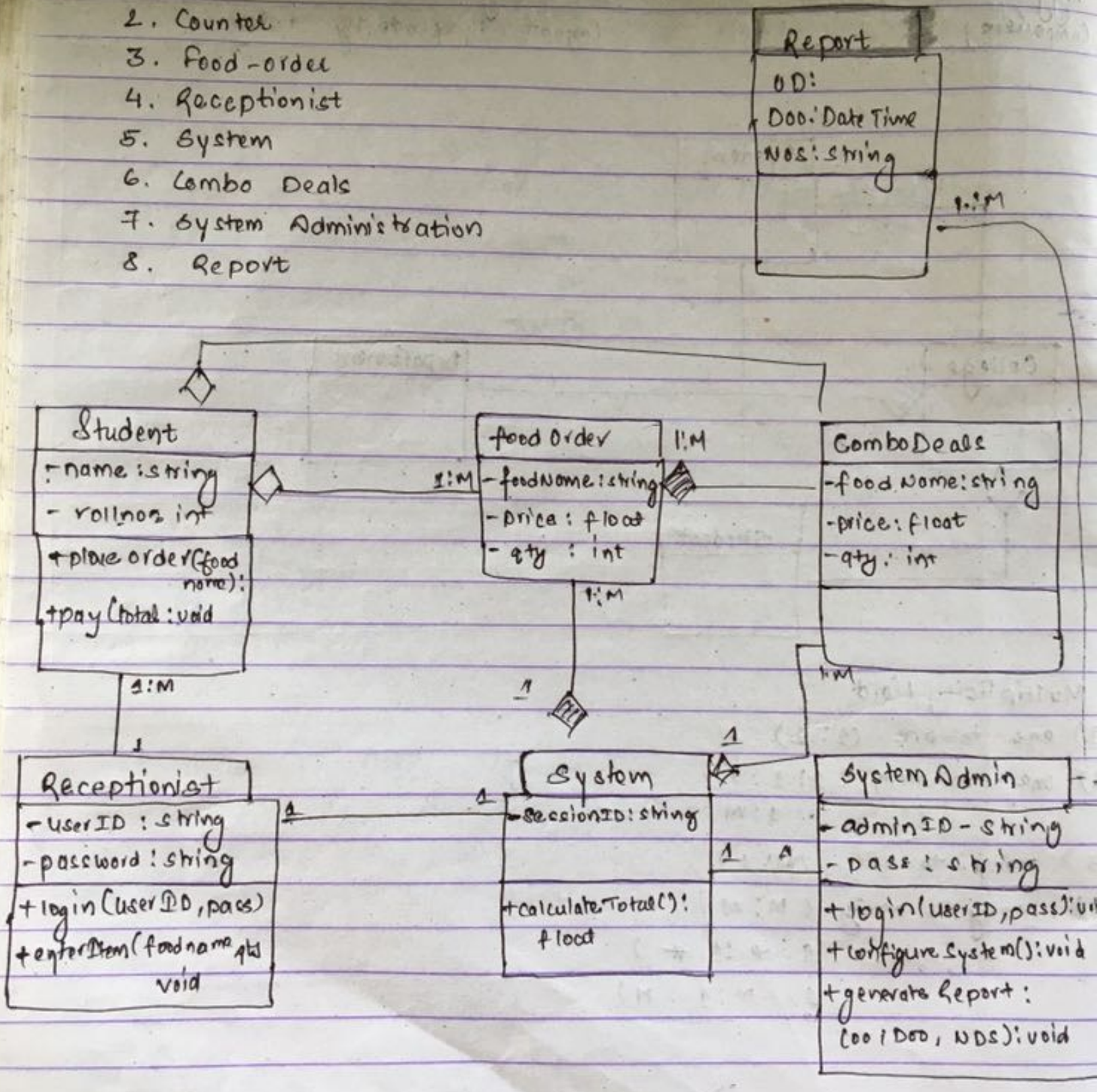
2) one - to-many (1 : 1 : *)
1 : M.

3) Many - to - one (M : 1)

4) many to many (M : M)
1 .. * : 1 .. *
1 .. M : 1 .. M)

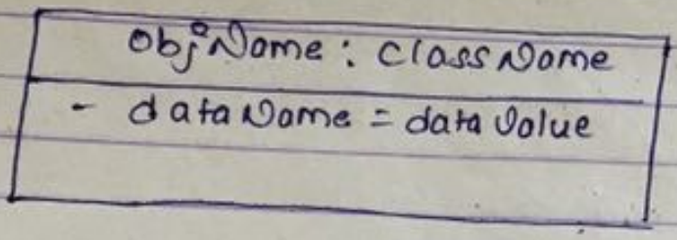
NCIT Canteen MS

1. student
2. Counter
3. Food-order
4. Receptionist
5. System
6. Combo Deals
7. System Administration
8. Report

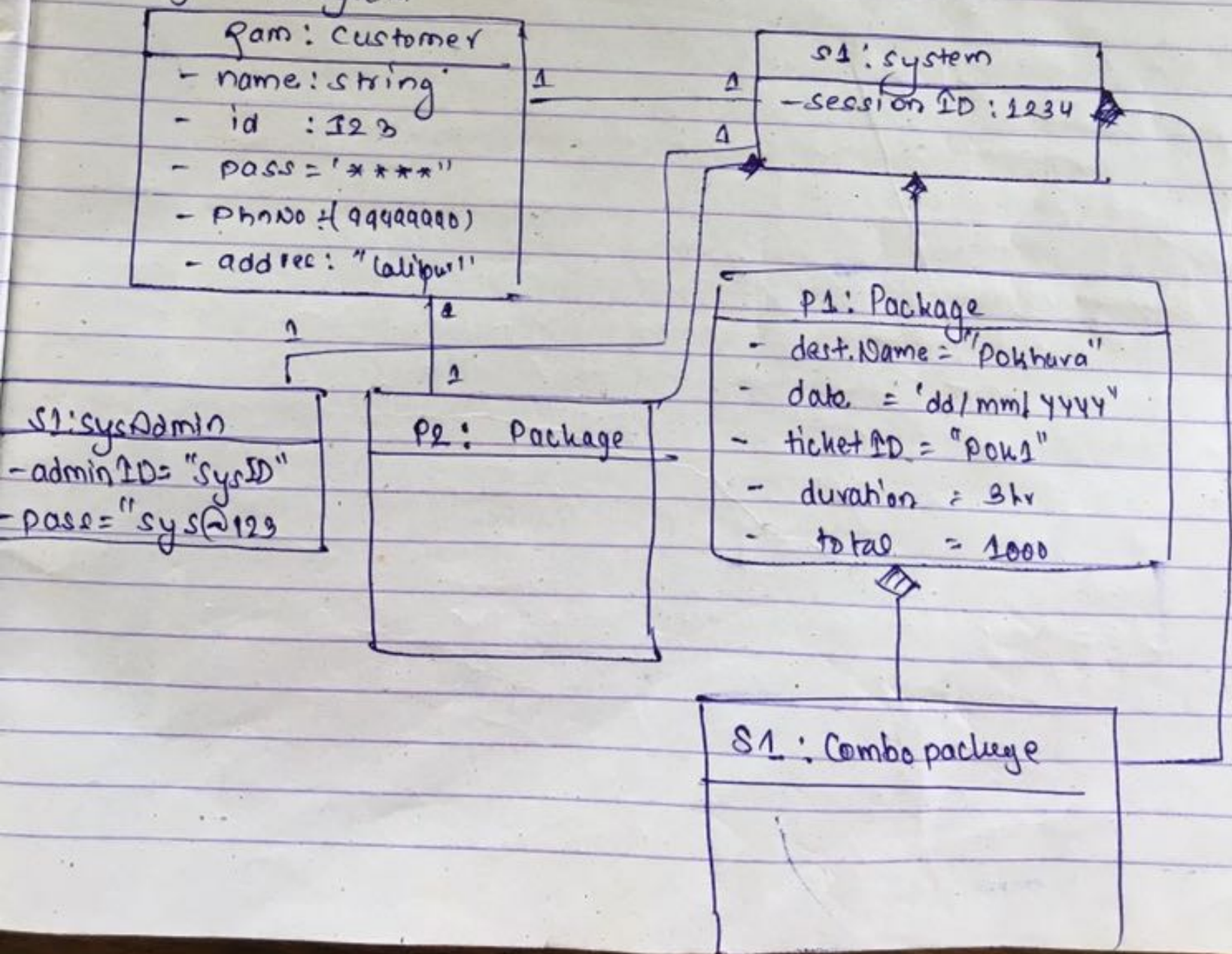


Object Diagram

- Structural UML diagram that shows various objects & their interaction in the system
- Instance of class diagram shows data & its corrasponding values



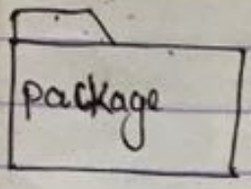
Object Diagram



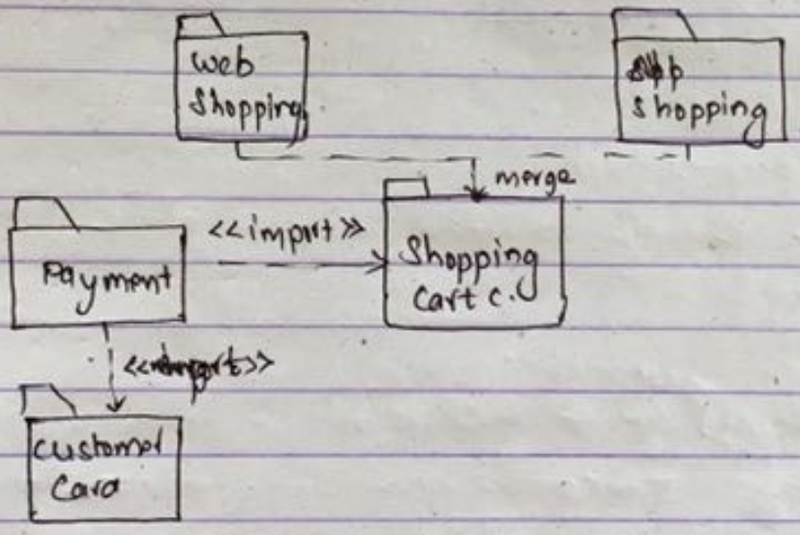
21st Aug 2019

Package Diagram

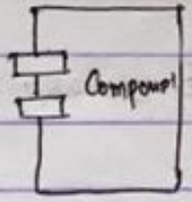
- diagram that shows packages and their dependencies.
- Package : collection of classes
- Two types of dependencies:-
  - <<import>> : Importing functionalities of a package
  - <<merge>> : Combining two or more package.
- Simplify complex close diagram.



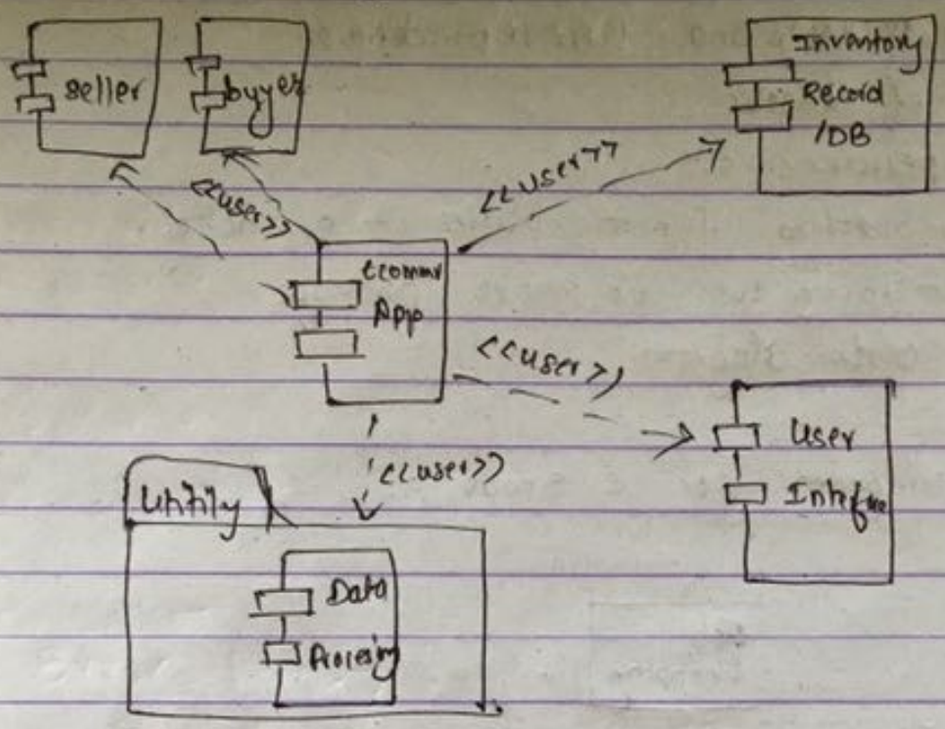
Package Diagram for SE Bhodo



- Component Diagram
  - Physical module of a system.
  -



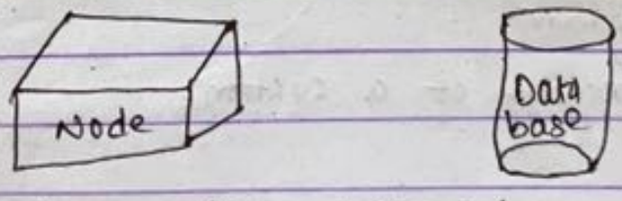
Component Diagram for seenado



wfest

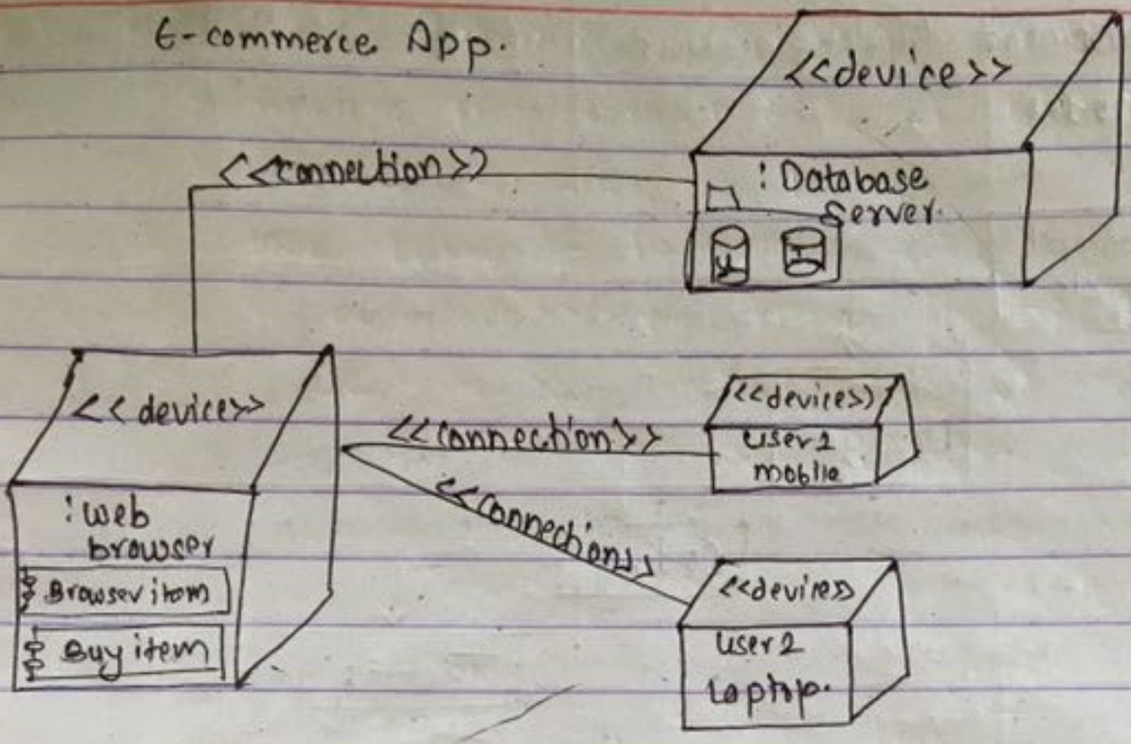
Deployment Diagram

- diagram that shows nodes and their relationship
- Node Notation



- Computing resources with processing capabilities.

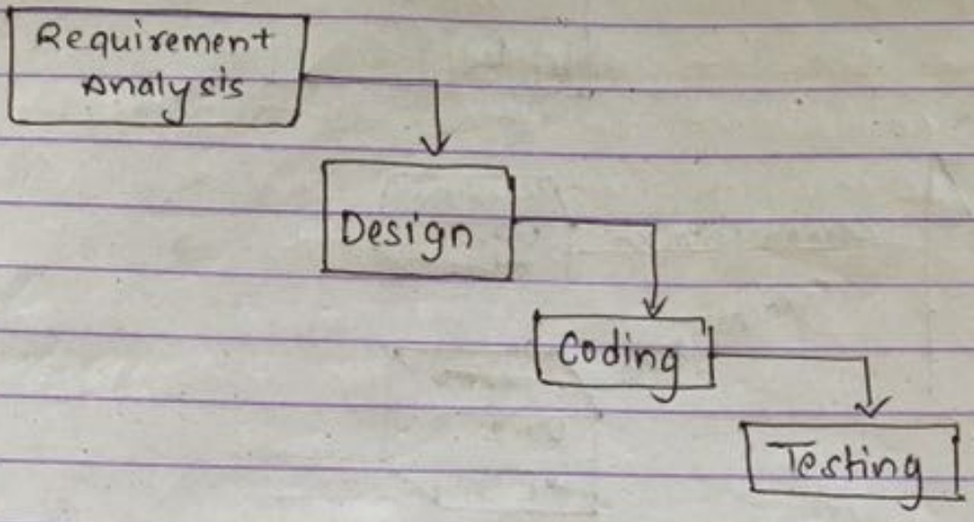
E-commerce App.



Why waterfall model is error prone? How can we address the change in requirements?

00 Software Development Cycle.

1) Waterfall Model



Error Prone?

- Based on false assumption
- Requirement can be clearly defined before hand. (before actually coding)

→ Change are inevitable & must be addressed in every phase (iterati
- Issued addressed by Iterative and Increamental model.

Iterative & Increament model.

- System is divided into sub-systems
  i.e short-fixed length modules. (3 weeks)
- The system grows increamentaly
  iteration by iteration, hence called I & I model.

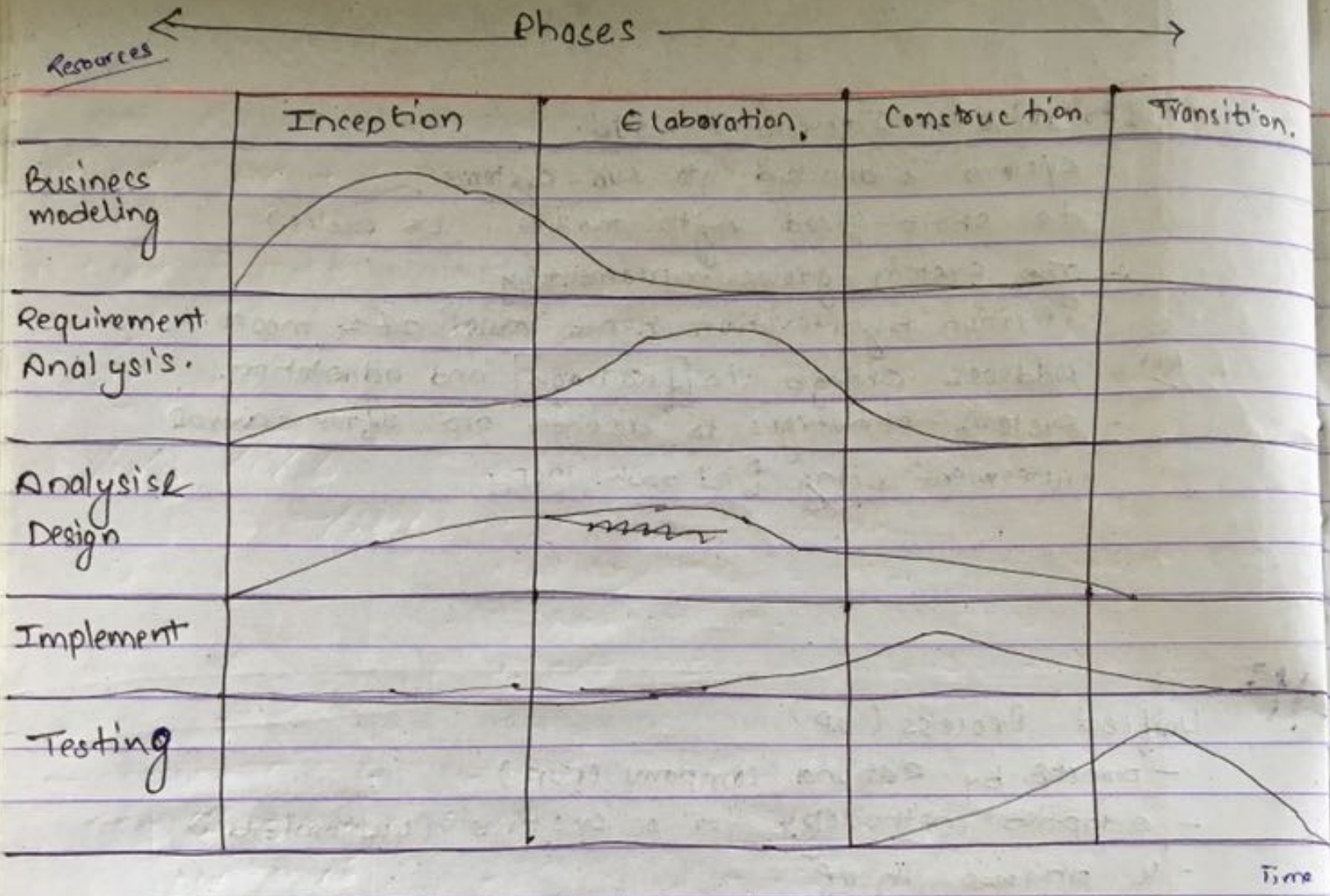
Ans: Address change via feed back and adaptation.

- System converges to desired o/p after several increament using feed back loop.

#

Unified Process (UP)

- process by Rational Company (RUP)
- adaptive methodology for OO systems (customized)
- 4 phases in UP
  1. Intepotion phase: long term vision, market study custo
     feasibility analysis, risk management
  2. Glaboration phase: refinement of pervious phase
     build use case, domain model / OO s
  3. Construction phase: coding / building the system
  4. Transition phase: Testing



Relⁿ between & workflow.

Chapter - 2

22nd Aug 2019

OOA — Object Oriented Analysis

1st → Uses Case
- Domain model
- SSD + Contract
(System Sequence Diagram)

OOA
→ 'Why?' question asked.
→ Process that examines requirement in the problem domain rather than focusing on how a solution is achieved / functional / Non-functional
→ Modeling / capturing requirement

Domain - Model

- Class Diagram without operations (idea / entity / thing or object)
- Model that operations Conceptual class / Domain class

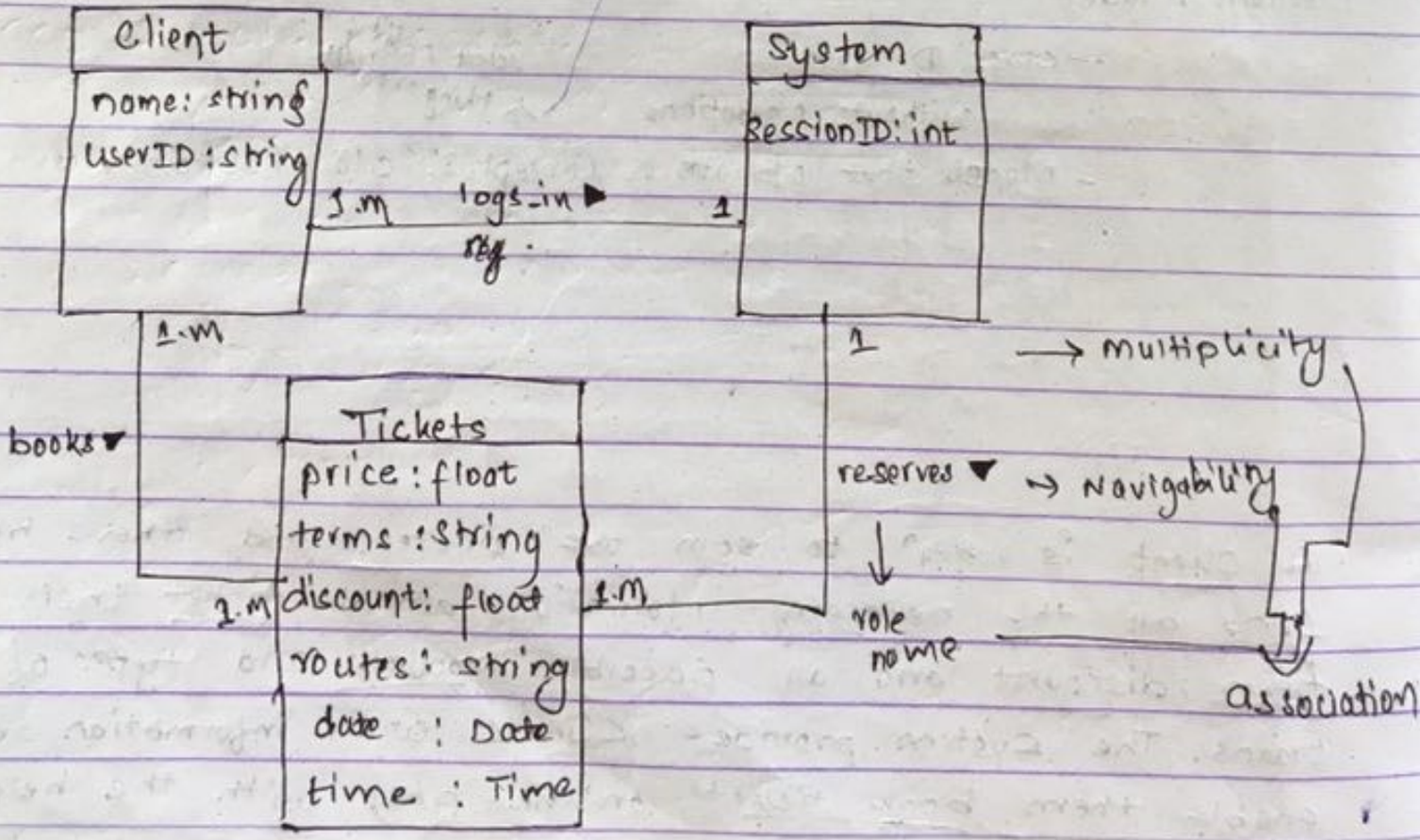
A client is req^n to login the system and there he can find all the necessary information about ticket their price term; discount and all possible router and types of trains. The system provides clients with information and enable them book ticket online pay with the help of a credit card; so there is no need to go to a booking

office any more. A client can also cancel the booking or change date timing of ticket.

i) Draw Domain Model

→ Step to identify conceptual class
- Noun - phase Identification

Domain Model → step to draw Domain Model.
- Class
1. Identify conceptual class
2. Draw them in UML class diagram
3. Add attributes of conceptual class
4. Add association betn conceptual class.

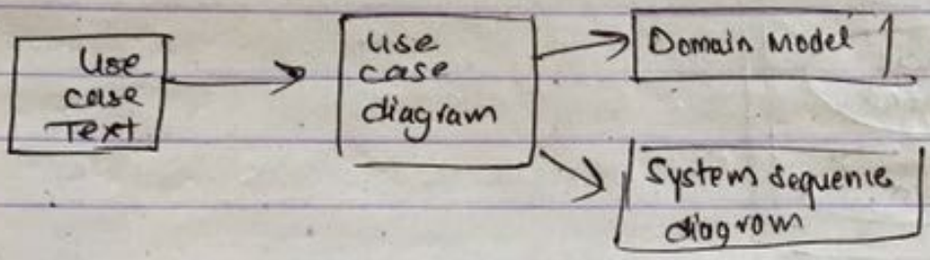


28th Aug 2019

Representation of System Behavior

- functional requirement of system
- description of what system does without explaining how it does it.

(Black-box model)
(Abstraction model)



SSD.

Sequence Diagram: Shows communication / flow of message between objects.

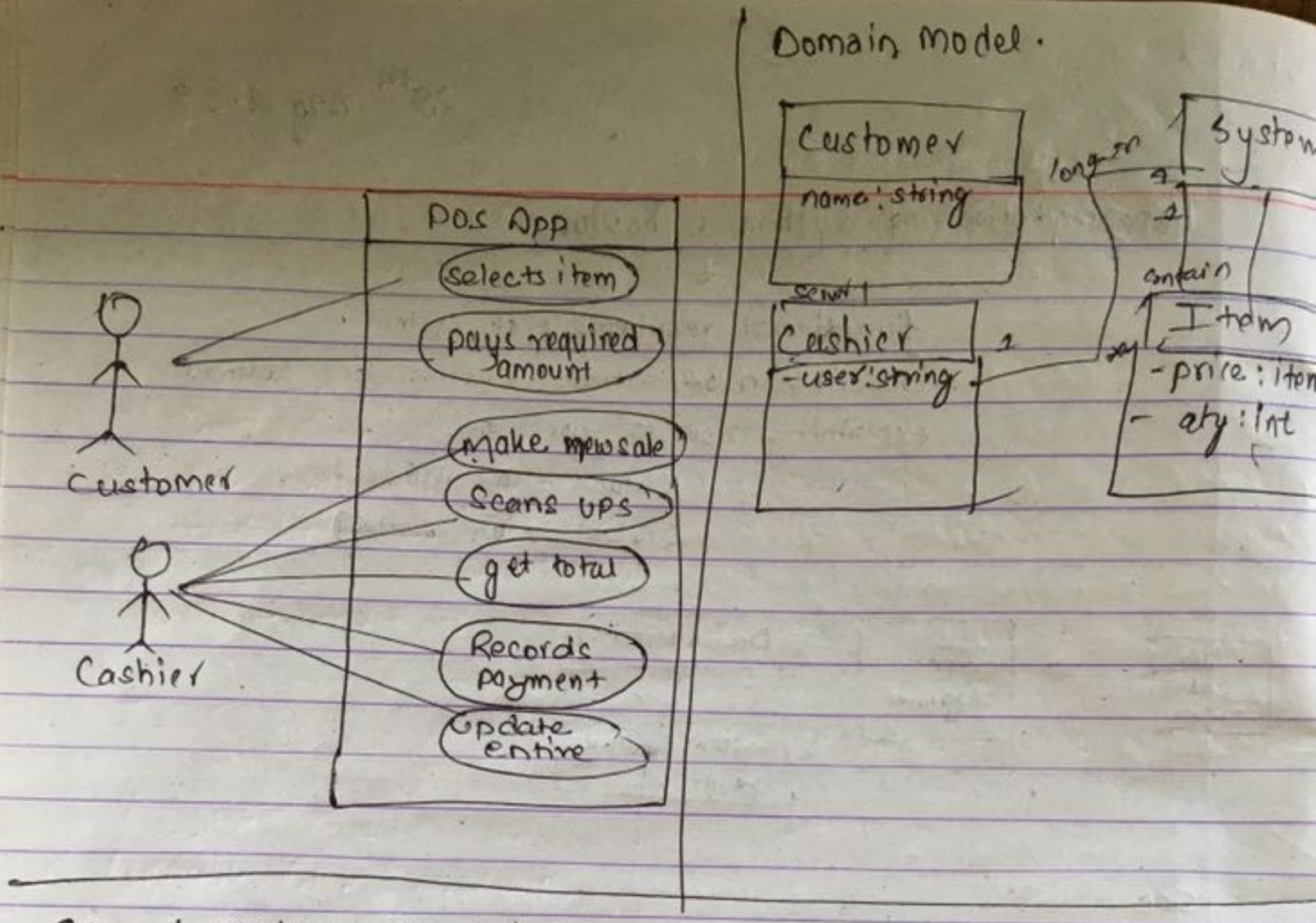
SSD: shows Communication between actor our system.

Case study: POS App

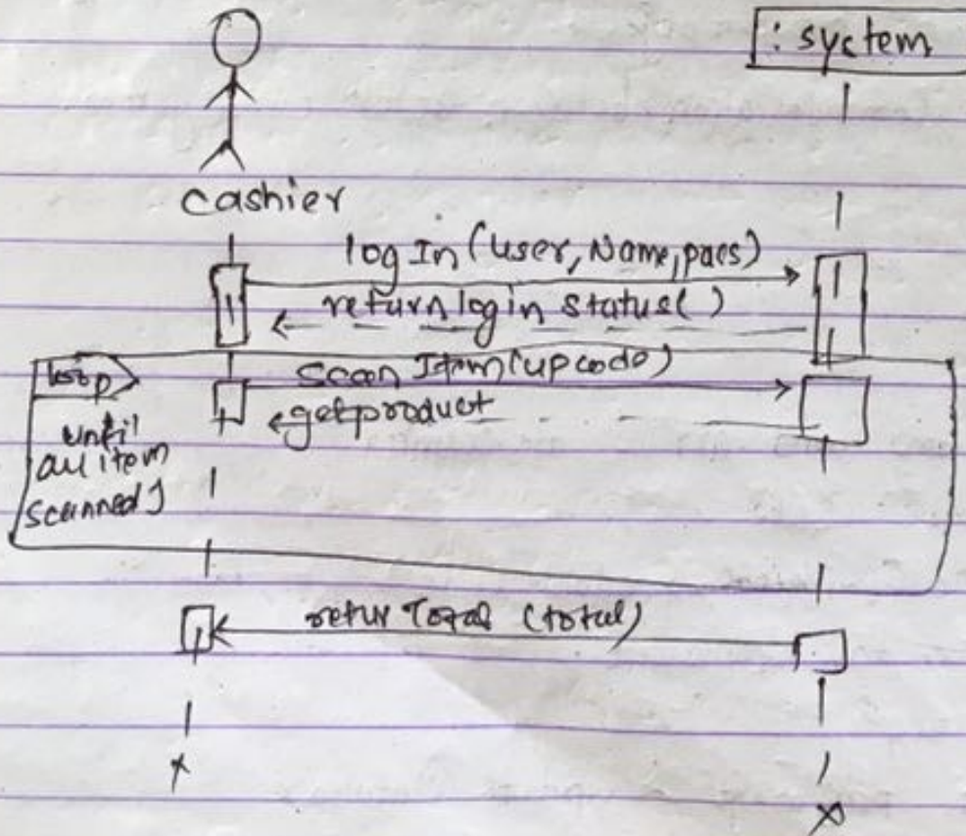
Use case Text

1: Customer select items and arrives at cashier
2. Cashier starts new sale.
3. Cashier scans UPC (Universal Product code.)
4. System calculate total amount.
5. Customer pays req^d.
6. Cashier records payment & update Database

1 2 3 4



SSD : make new sale.



Q. How to represent System Behavior?
- use case
- SSD
- Contract.

* System Behavior:

System event
Ci/p to system
events / actors generated by external actor to the system

for complex course enrolling

Steps / Guidelines to draw SSD.

1. Identify external actors that directly interacts with system
2. Identify system events and system operation.
3. Draw them in Sequence diagram.
4. (optional) include use text to left to SSD.

1. Instance (creation (or deletion)
2. Attribute modification.
3. Association formed.

Contract in UML

Contract CO: name
operations: logIn (userName: string // Name of operation with parameters pass: string)
Cross reference: make new sale.
Pre condition: Cashier has already accounts registered
~~Post condition~~: Cashier credentials are valids.

Post condition: Cashier instance 'c' was created (IC)
- login status was displayed
- 'c' was associated with System (AF).

Contract CO: Scan Item

operation: scan Item (upcode string)

Cross reference: make new sale.
pre-condition: cashier must be logged in
- upc code must be valid.
- Customer has already selected items.

post Conditions: Price and description was returned
Item instance 'i' was created
'i' was associated with System
i upcode was matched to upccode (Am)

Guideline for Contract

1. Identify system events & system operation from s
2. for event that are complex, prepare contracts.
3. Include pre condition and post-conditions.
4. Post condition formed
   past passive (was +v3)
   (1) IC
   (2) VM
   (3) AF

1.) Described detailed system behaviour in term of sta changes. to objects in the Domain model.
2) Decscribe what an operation commits.
3) Emphasizing what will happen
4) Expressed in terms of Pre & Post conditions. as a state

What is expanded use case?

29th Aug 2019

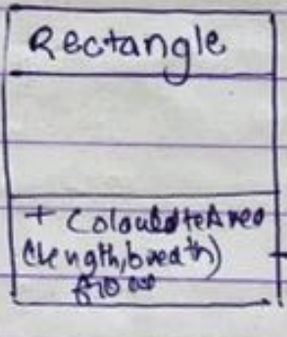
Chapter-3.

OOD

OOA ends with
- use case Text
- use case Diagram
- ~~DS~~ main model
- ~~contract~~ SSD
- Contract.

- OOD focous on fulfilling requirements
  Collects in OOA.
- OOD
  - Interaction Diagram - sequence - force format, timely flow of msg
    - communication - graph format, space flexibility
  - class Diagram / Design Class Diagram (DCD)
  - Patterns.

Responsibilites, Operations & Methods.



Responcibility : calculate Area of a rectangle

operation : float calculateArea
(float length, float breadth);
// Declaration

methods : float calculateArea (float l, float b) {
return (l*b);
}

✓ Responsibility — agreement

"duty or obligation of a system / classifer.

Operation

Declaration of function without implementation.

method :- Implemention of a function
- Method are implemented to fulfill responsibities.

Types of Recponsibilities.
→ Doing responsibility
→ knowing Responsibility

Responsibilities is related to the agreement of an object in term of behavior.

Cashier

Doing . cashier scans UPC code.
co cashier update Item DB
cashier receives payment.

knowing :- Cashier must flow the requirement total of a customer

- Doing Responsibilities of an object includes performing an action, itself such a calculation or Creating an object.

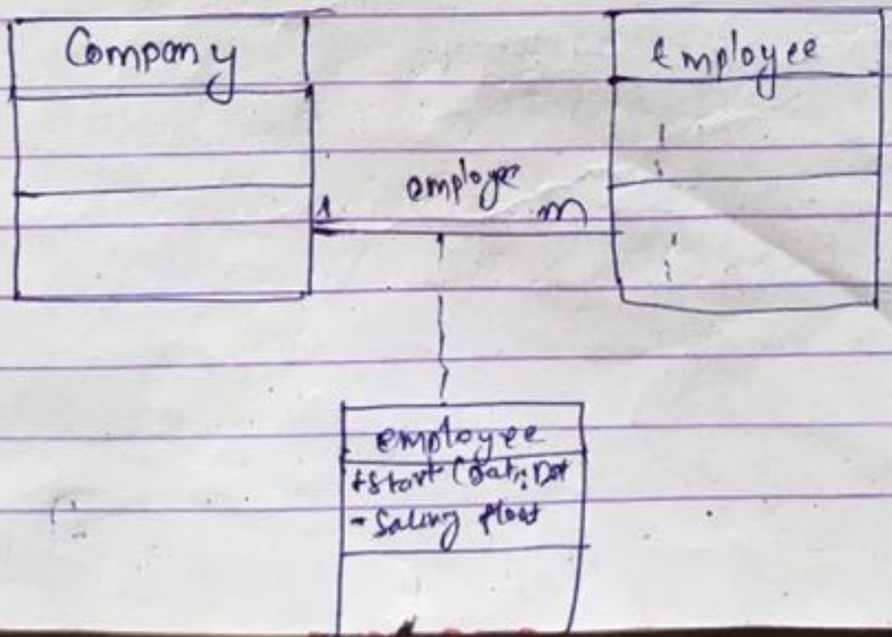
- Knowing responsibilities of an object include knowing about private encapsulated data related objects.

Constraint
- Condition or restriction on uml element
  length >= 10
  breadth >= 20

Rectangle.
- length float {length >= 10}
- breadth float {breadth >= 20}
+ calculateArea.
.(length, breadth) float | operation

Singleton class
- only one object / instance of the class can be created / instantiate
- '1' in top-right corner of class_name.

Association class



Imp

Visibility bet^n objects.

Visibility of a data member / function / class

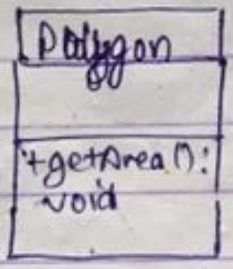
+ Public - outside class also has acess ('+')
- private - only containing class has acess (-)
protected - Chid class also has acess (#)

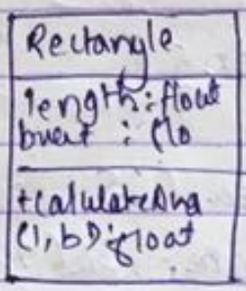
↳

- for sender object to send message to receiver object receiver object must be visible to sender object.
- 'visibility' : ability to see or have reference to another orbject.

Types of visibility

(permanent) - Attribute : B is an attribute of A.
(Temporary) - Parameter visibility : B is a parameter of a methods of A.
Tempory - locally declared : B is locally declared in Method of A.
parament. - Global : B is global.





1.

Class

```
Public ^Polygons
float area;
public void getArea() {
area = CalculateArea();
}
}
```

```
Public class Rectangle {
float length;
float breadth;
public float CalculateArea (float length, float breadth);
return (length * breadth);
}
}
```

2.

```
Public class Polygon {
float area;
public void getArea
(Rectangle R) {
area = R.calculateArea();
}
}
```

3.

```
Public class Polygon {
float area.
public void getArea() {
Rectangle rect;
area = rect.CalculateArea();
}
}
```

4.

```
Rectangle R;
Public class Polygon {
float Area;
public void getarea () {
area = R.CalculateArea();
}
}
```

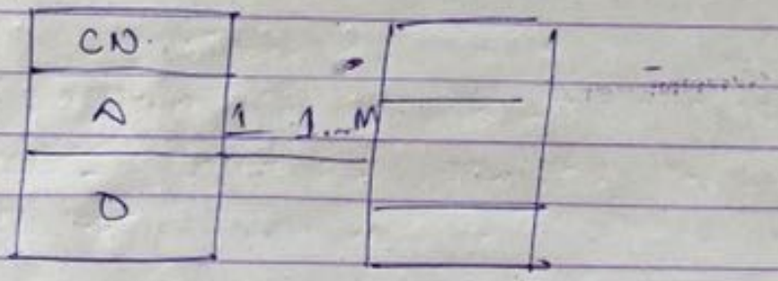
8

DCD (Design Class Diagram)

- DCD is a UML Diagram that shows various software class in the system.

Shows:

a) Software classes
b) Attributes of class
c) Operations of classes.
d) Relationship and cardinality between classes.



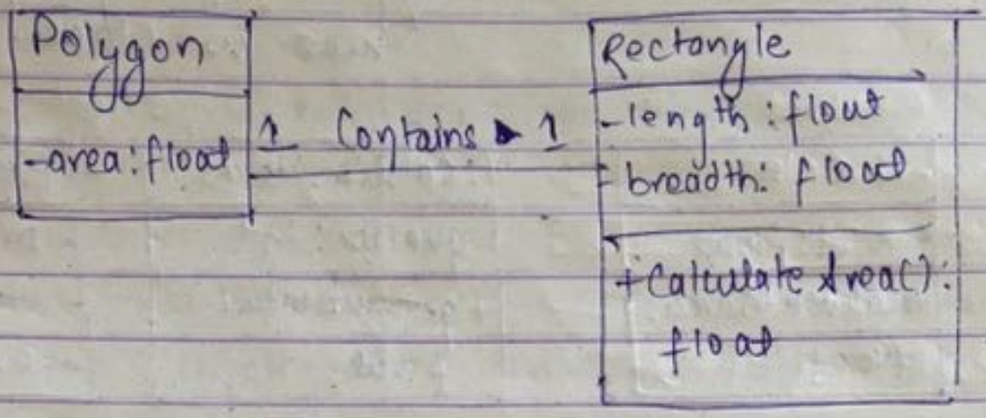
Qualified Association

Color Catalog | 1 ——— 1..M | Color Description |

Color Catalog | C-id | ——— 1 | Color Description |
└ qualifier.

- has a qualifier that is used to select a primary key from large set to change that multiplicity from many down to 1.

Pattern



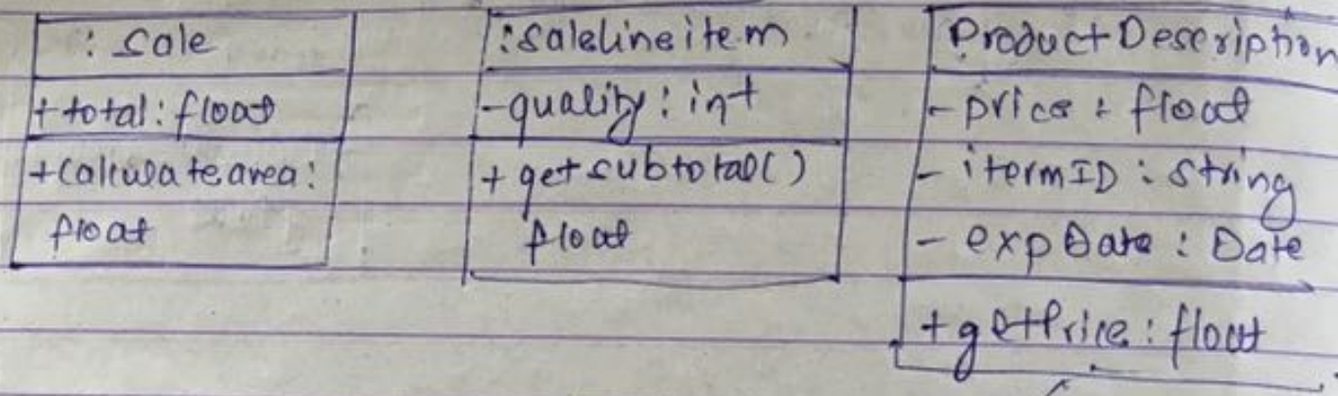
- Pattern provides guideline / principle for assignment of responsibilities to a class / object.
- GRASP (General responsibility assignment software Pattern / principle)

Types of Pattern / GRASP

1. Information Expert
2. Creator
3. High Cohesion
4. low Coupling
5. Controller
6. Puve fabrication
7. Polymorphism
8. Protected Variation
9. Indirection

Information Expert Pattern // give info regarding operation keep in class.

Example.



~~Now~~

- This pattern guides us to assign the responsibility to that class that has all the necessary information required to fullfill it.

Pattern Name: Information Expert

Problem: who is responsible for doing certain things?

Solution: Assign responsibility to that class having all the necessary information.

Pattern is a named description of problem-solution pair.

Problem: who is responsible for getPrice?

Solution: Product Description is an Information Expert for getPrice since it has info about Price.

2. Creator

Patter Name: Creator

Problem: who is responsible for creating an object / instead?

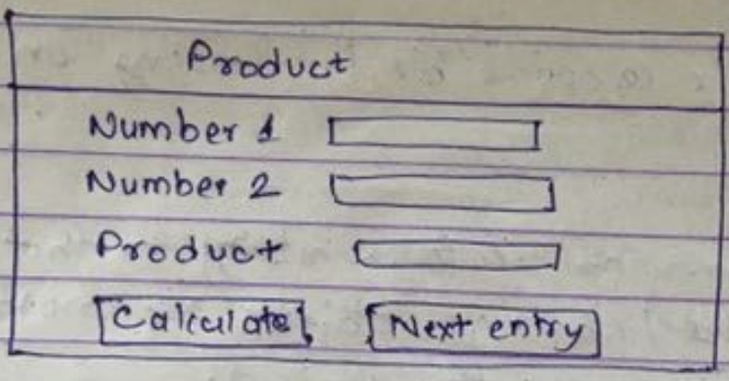
Solution: Assign the responsibility to that class which aggregates / contains objects of another class for closely uses the object.

Problem: who is responsible for creating instance of SaleLine Item?

Solution: Assign the responsibility to Sale for creating instance of Sale Item since it aggregates / contains SaleLine Item.

5th Sep 2019

Controller Pattern



1) System (facade Controller) Calculate
2) Use case Handeler (use Case Controller)

```
Private JFrame f;
Private JLabel n1 = new JLabel ("Number 1");
Private JTextfield j1, j2, j3;
  ---- n2 = ---- ("No 2");
       n3 = ---- ("Product");
JButton b = new JButton ("calculate");
f.add (b);
b.addActionListener (this);
;
!
```

```
void calculate (action Event e) {
    n1 = j1.getText().toInt();
    n2 = j2.getText().toInt();
```

4th Sep 2019

Patterns.

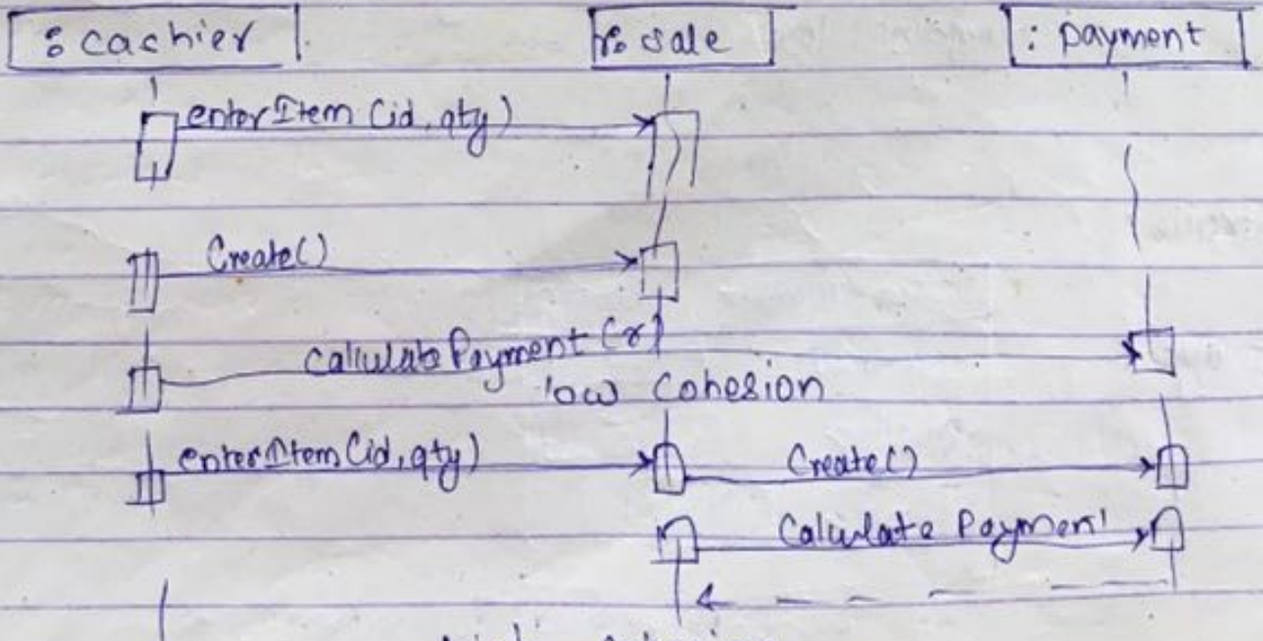
3. High Cohesion

Cohesion is a measure of how strongly related & focoused the responsibilities of an element (class) are:-

- A class with high cohesion only does specific work/responsibilities.
- A class with low cohesion does mendous amount of work.
- A class with low cohesion faces following problems:
  - difficult to understand (comprehend)
  - decreased code reusability
  - Dependency increases / Coupling increases
  - Delicate / unstable.

Problem: How to make objects focoused, understandable managed & support low coupling?

Solution: Assign the responsibility so that Cohesion always remain high.



4. Low Coupling

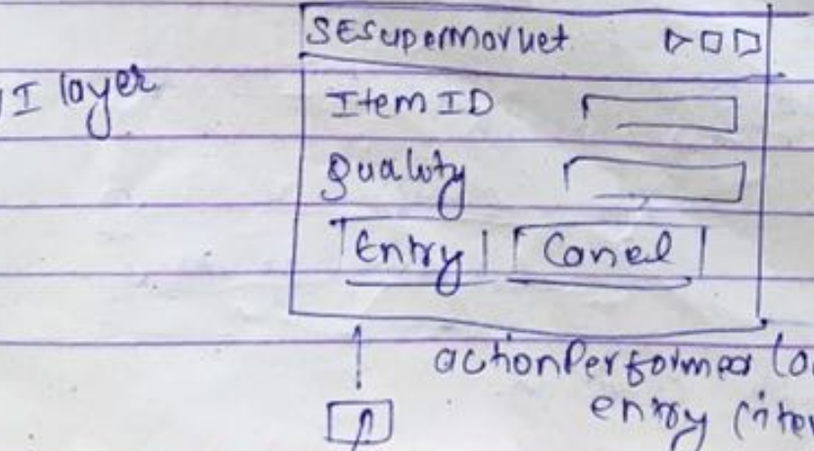
- Coupling is a measure of how strogly connected one element is one another or relies on another.
- A class with low coupling (weak coupling) means that class is less dependent an another.
- A class with high coupling (strong coupling) means that class is too much dependent on another.

- high Coupling problem
  - if bug aries in a related class the dependent classes gets affected.
  - forced local changes
  - Decreased Reusability
- ,

Problem :- How to support high cohesion, decrease dependency and increase reuse?

Solution : Assign the responsibility so that coupling remains low.

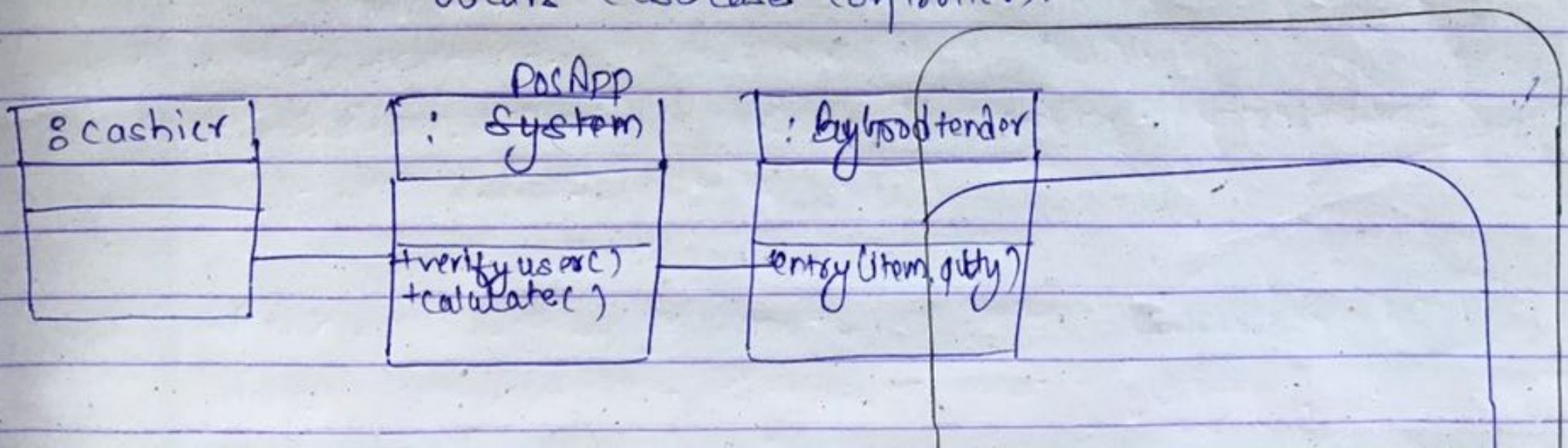
5. Controller



Problem : Which object beyond UI layer is responsible for receiving and controlling a coordinate system operation?

Solution :- Assign the responsibility to one of the following choices

1. System (root object, or major subsystem) (facade controller)
2. Use case scenario where the system operation occurs (use case controller).



Chapter-4. OOI

Object-oriented Implementation.

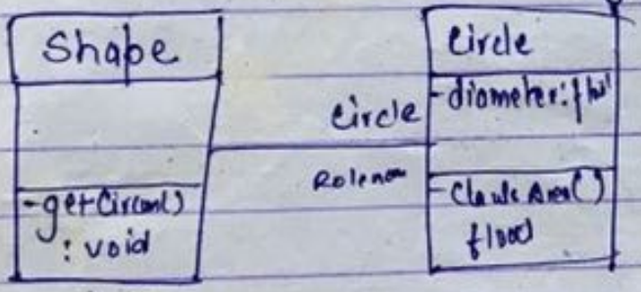
- Implementing the system in an object object-orlented language
- OOI is process of mapping design to code.

Steps for mapping design to code

1) Creating class definitions from DCD (Design class Diagram)
2) Creating method definitions from Interaction Diagram.

Creating class definitions from DCD.

1. ~~Creating~~ DCD provides specification for:
   - i) class Name,
   - ii) Attribute.
   - iii) operations.
   - iv) Association.
   - v) multiplicity



Shape.java.

```
public class shape {

    public void getCircum() {


    }
}
```

Circle.java.

```
public class Circle {
~~public class~~ C
    float diameter;
public float calculateCircum() {

    return (3.14*dia);
}
}
```

Short notes

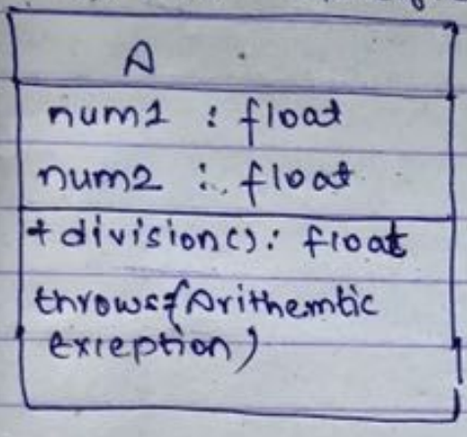
Emp

Exception and Error Handling

- unwanted & unexpected error
- Exception is response to exceptional / unexpected circumstance that might occur while implementing OOD to OOP.

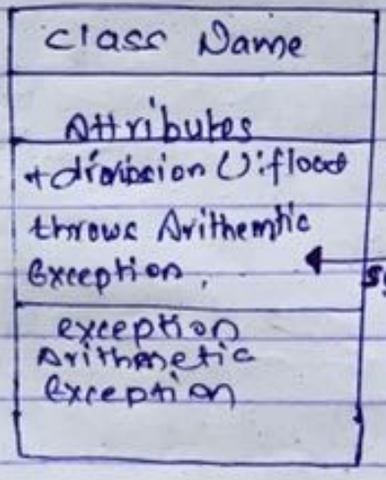
*) Exception handling

1) UML specific exception using OOP system (using throws keyword)



2) UML specific exception using "property string" (key + value pairs)

3)



✓ UML Specifies exception in a separate compartment in OOD.

When