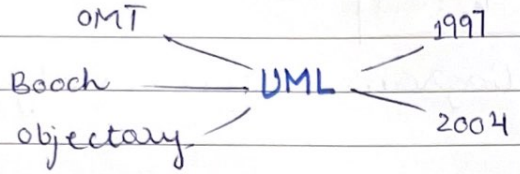
Date 01/02/2023.

Software Design and Architecture.

OOM by Micheal Blaha.



→ UML.x latest version.

OO Paradigm:

→ Abstraction.
→ Encapsulation
→ Inheritence
→ Polymorphism.

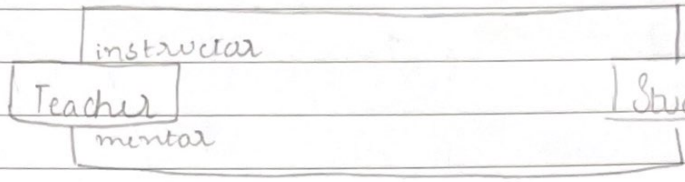
Modelling types.

→ Class – static in nature.
→ * State – dynamic in nature, transitions from state to state.
→ Interaction :– dynamic in nature; different objs exchange info.

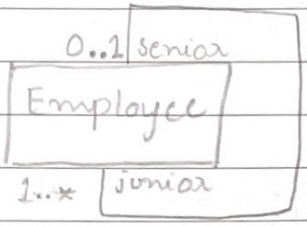
Class → Class diagrams and object diagrams.
Interaction → UCD, AD
SD → SSD, PSD.

Date

no visibility for OD



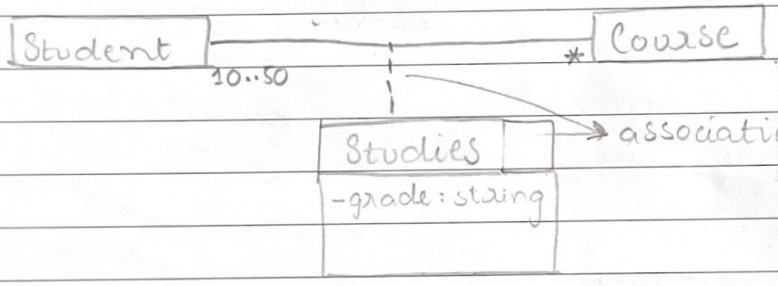
Binary association.

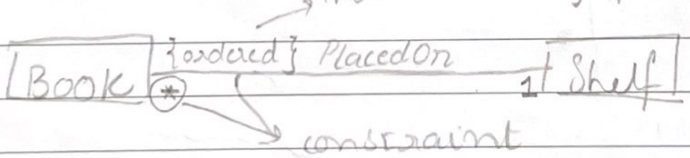


Top - bottom
left to right

() parenthesis
[] Brackets
{} Braces / Curly Braces

Self association = unitary association.





Constraints :

{ordered} → ordered set, duplicates not allowed.

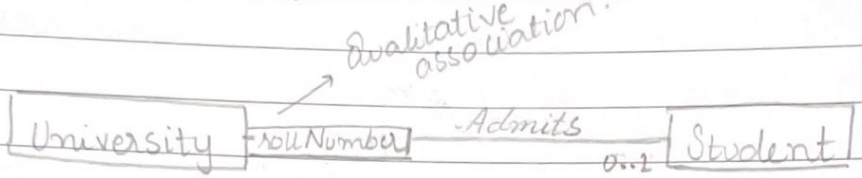
{sequence} → order/sequence is imp and duplicates are allowed.

{bag}

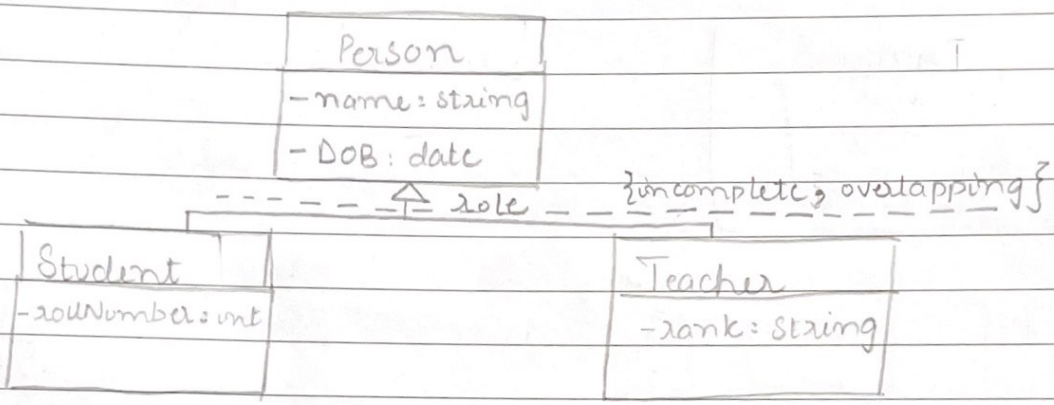
→ duplicates are allowed and order is not important.

Date

Qualitative association:



Binary Inheritence.



Constraints

{complete} {overlapping}
{incomplete} {disjoint}

Uzair : Student
rollNumber = "21L-5803"
name = "Muhammad Uzair"
DOB = 1-1-2000

Date 8/02/2023

UnderGradStudent		Course
-name : String [1]		- code : String
-rollNumber : String [1]	Registers	- name : String
- CGPA : double [1]	* *	- maxSeats : int → static class attribute.
- phoneNumber : String [*]		
- hostelAddress : String [0..1]		+ getMaxSeats() : int
- year StudyYear		
+ updateCGPA(n double) : void		

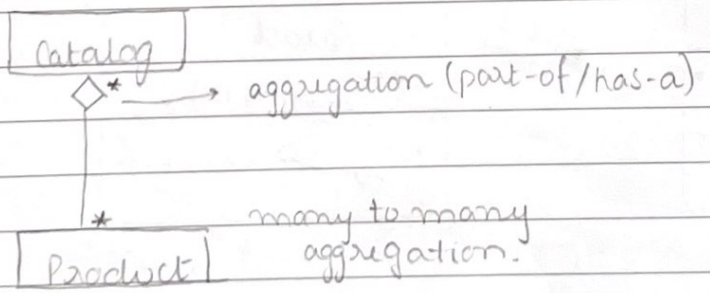
features

Implementation of operations is called method.

<<enumeration>>
StudyYear
Freshman
Sophomore
Junior
Senior

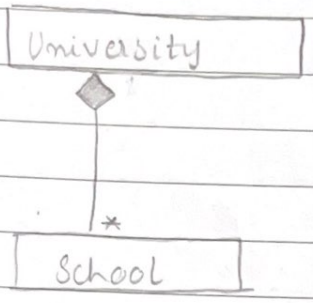
* List all enumerations together.

Aggregation:



Date

Composition: (Part-of) (stronger)

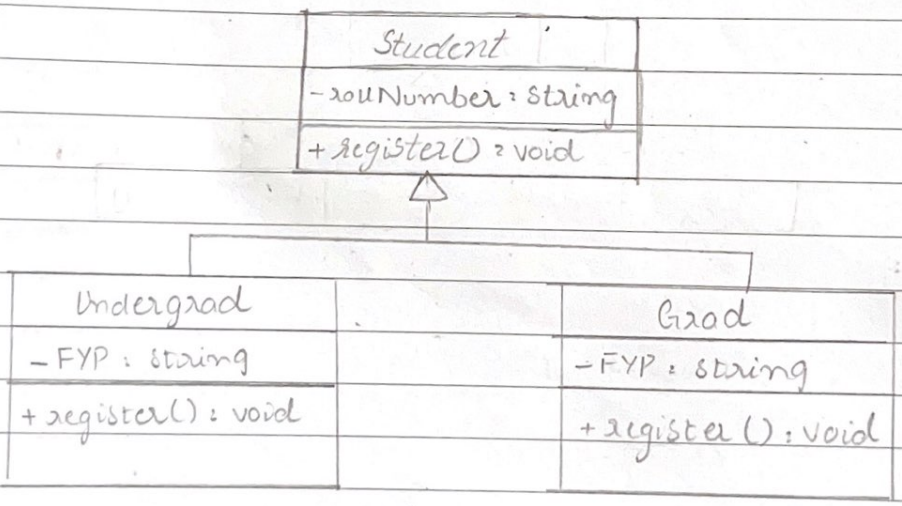


already known multiplicity (no need to write)

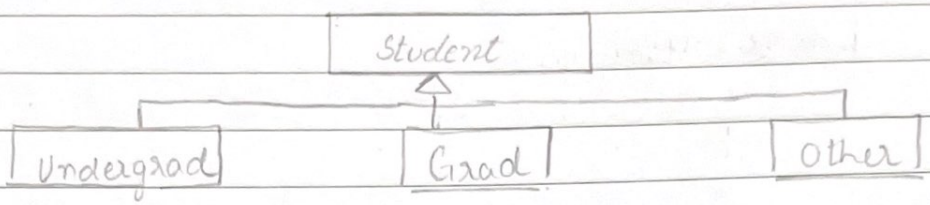
co-existent lifetime, they exist together.

- Binary association are transitive.
- Aggregation and composition are transitive.

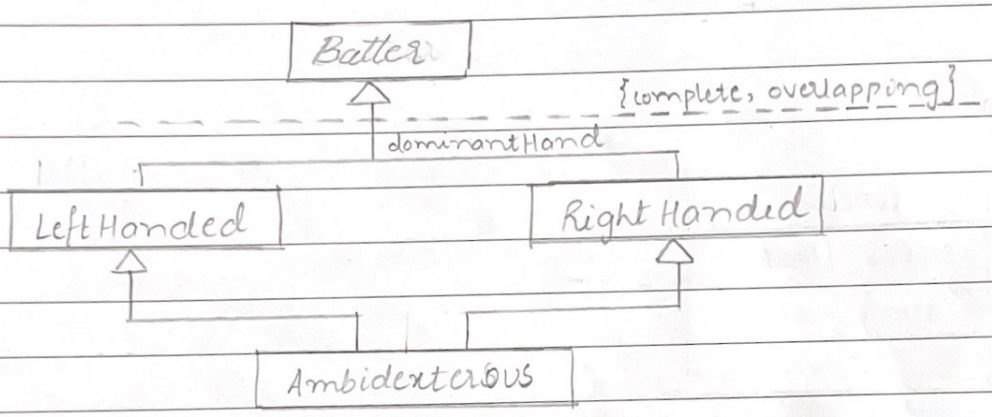
* Abstract class — write name in italics.



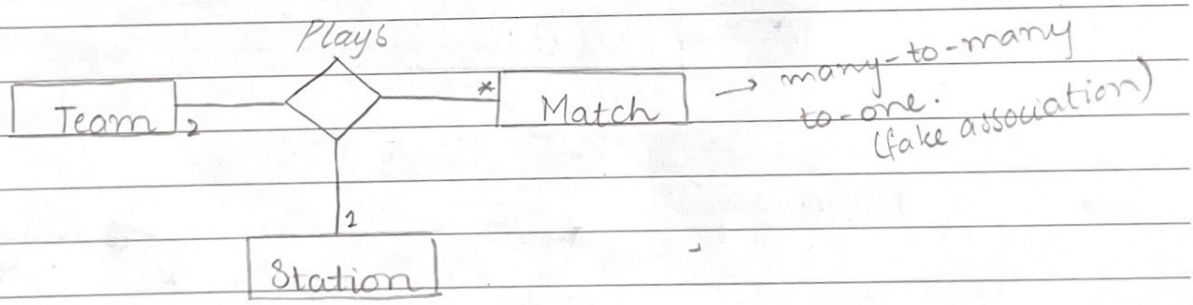
Date



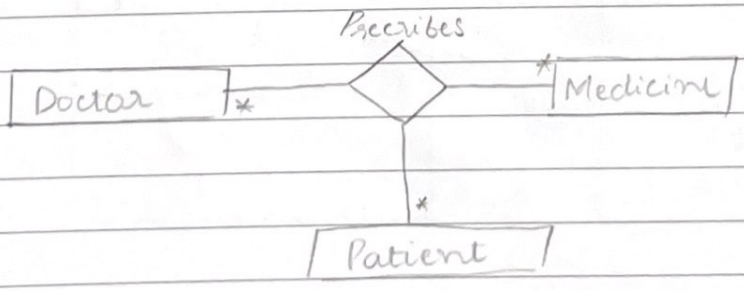
Multiple inheritence:



Terneraxy association:

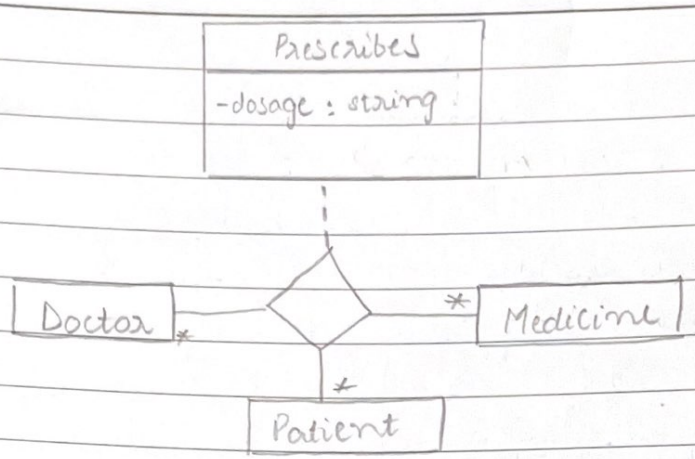


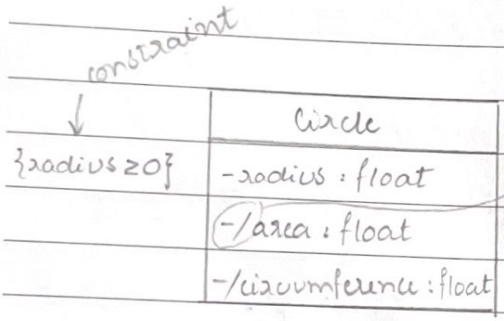
→ many-to-many to-one. (fake association)

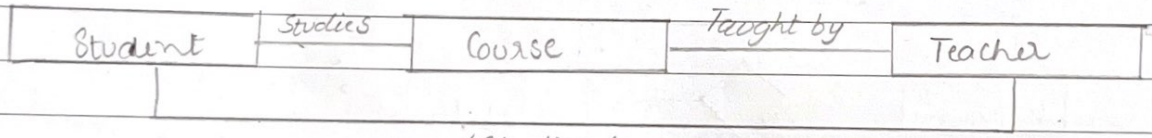


* Genuine association is many to many to many.

Date



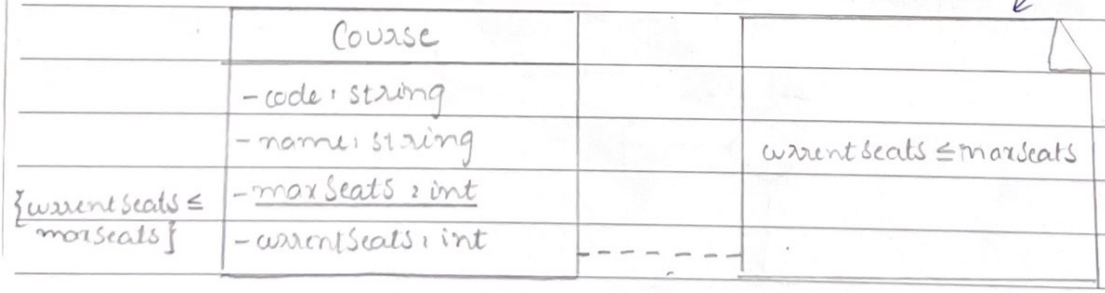




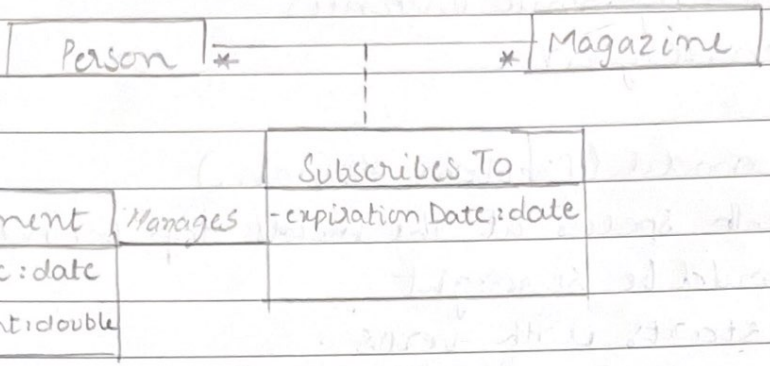
/Studies from

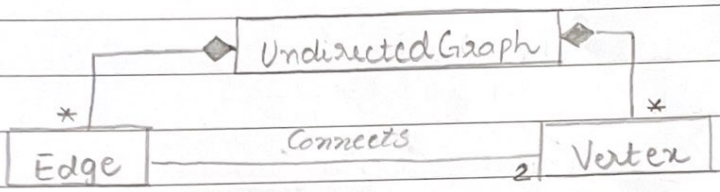
derived association.

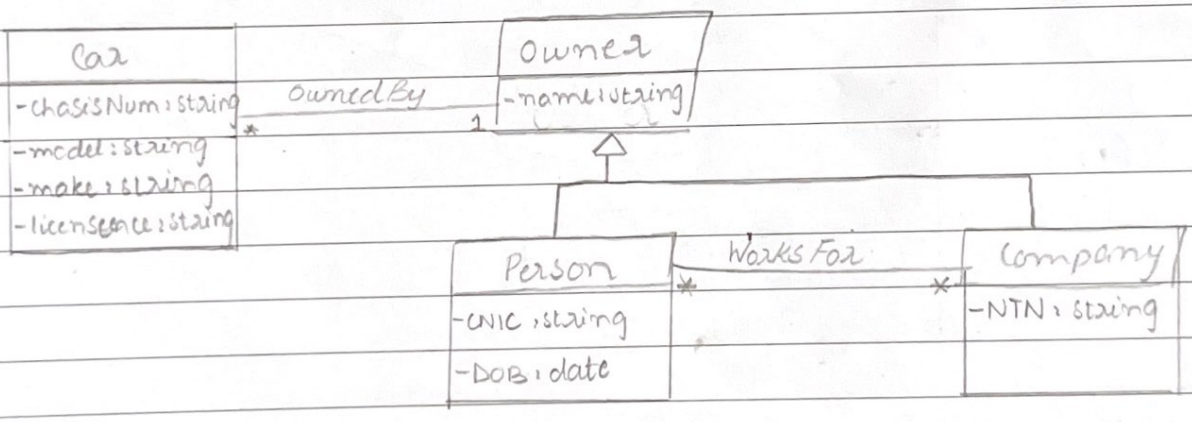
comment box syntax



Date


Person
*
*
Magazine
Subscribes To
-expirationDate:date
Payment
-gDate:date
-amount:double
Manages

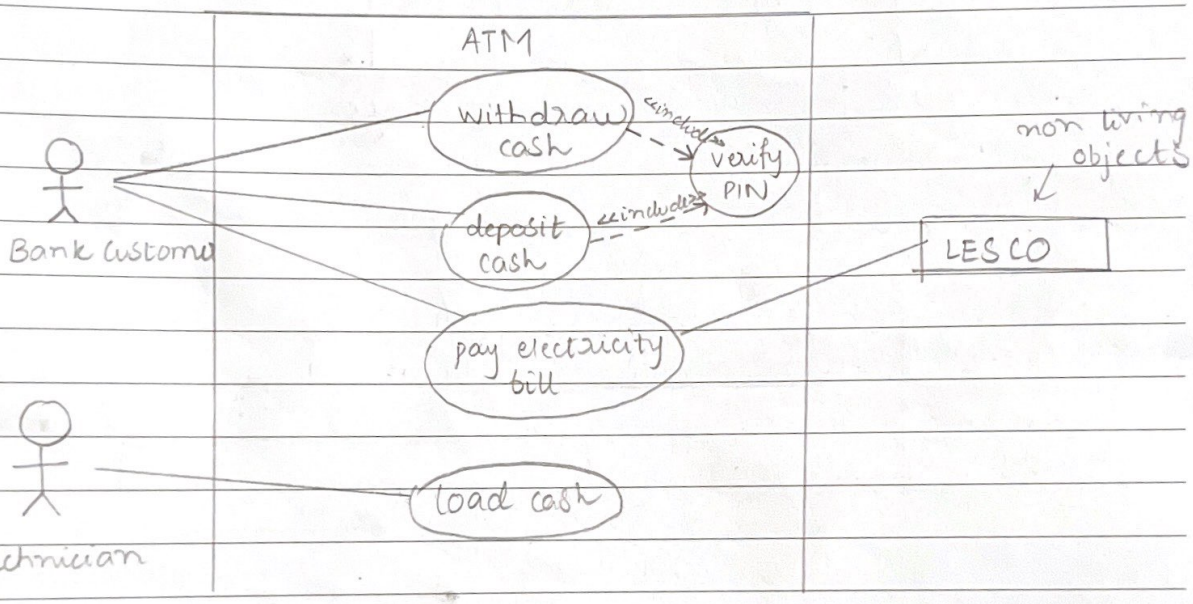

UndirectedGraph
*
*
Edge
Connects
2
Vertex


Car
-chasisNum:string
-model:string
-make:string
-licensence:string
OwnedBy
*
1
Owner
-name:string
Person
-CNIC:string
-DOB:date
Works For
*
*
Company
-NTN:string

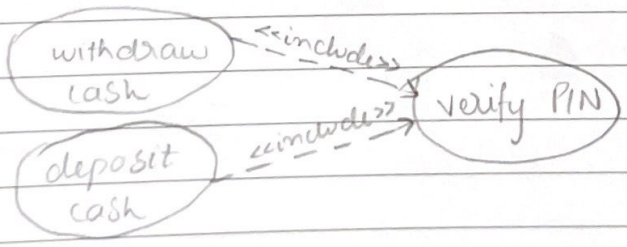
Date

Interaction diagrams – dynamic instances.
Use case – interaction diagram.

Use case: Analysis model (Problem domain)
- upper camble case with spaces at the inside top boundry.
- Stickman-arm should be straight
- All use case name starts with verbs
- No dangling actors or use cases.

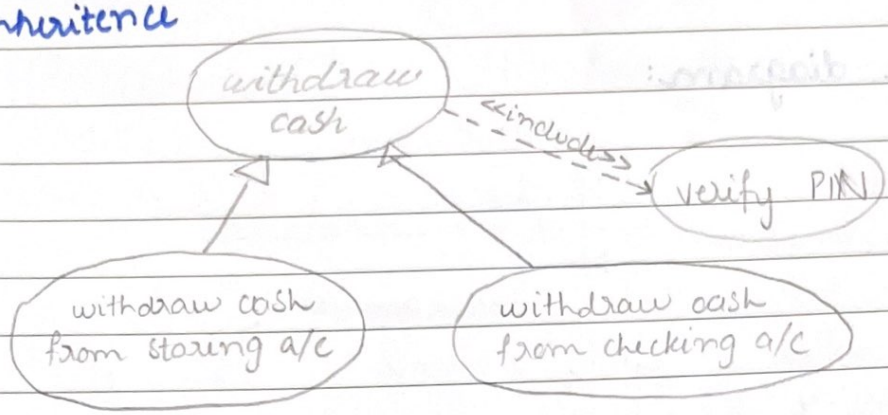


Include:

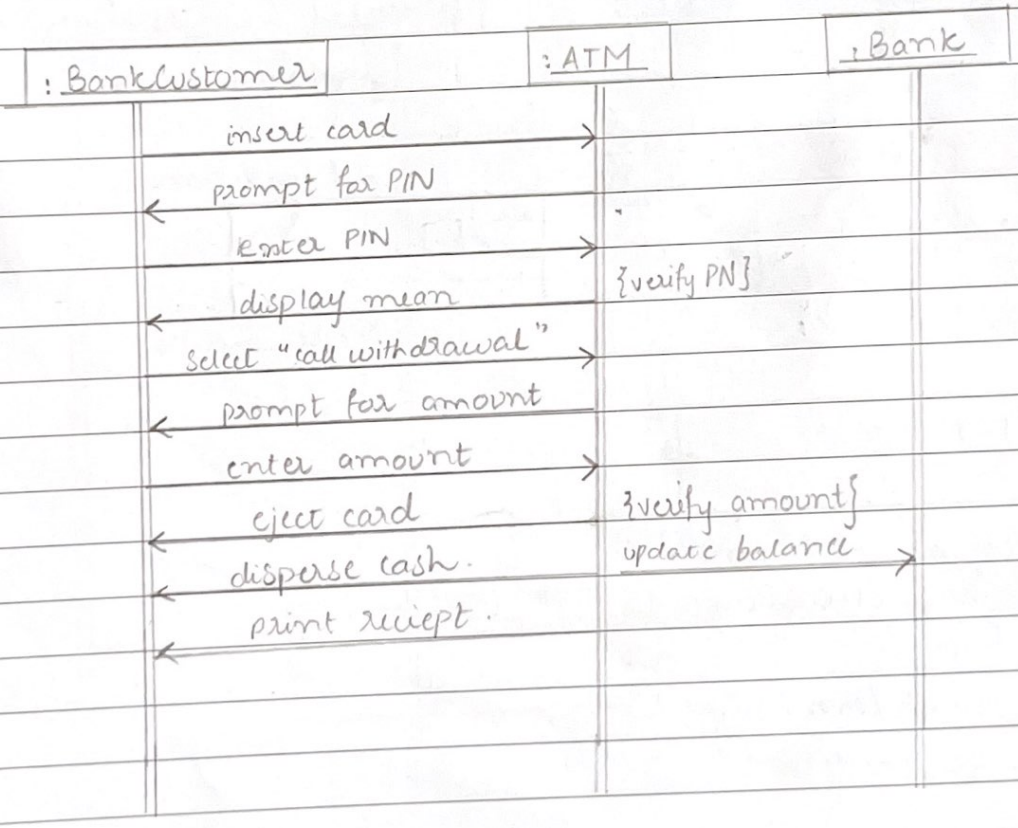


Date

Inheritence

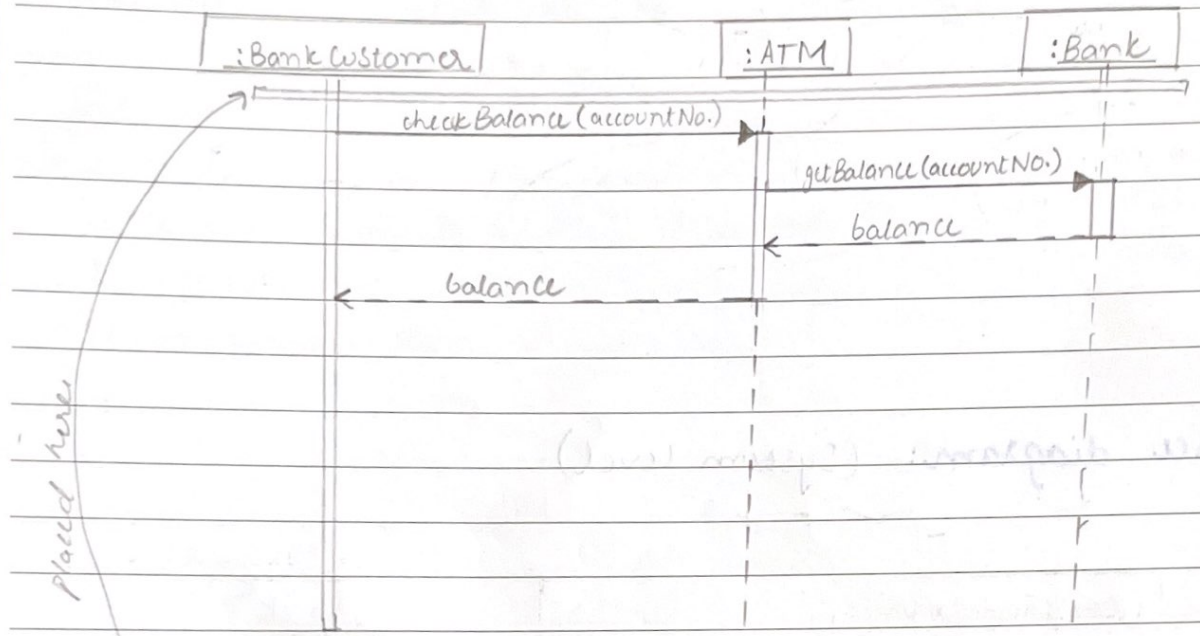


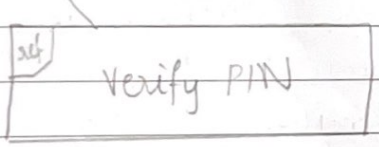
Sequence diagram: (System level)



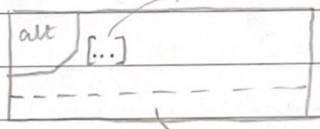
Date

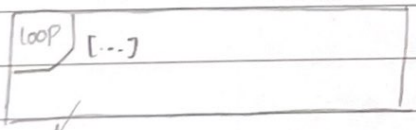
Design level Sequence diagram:





reference.

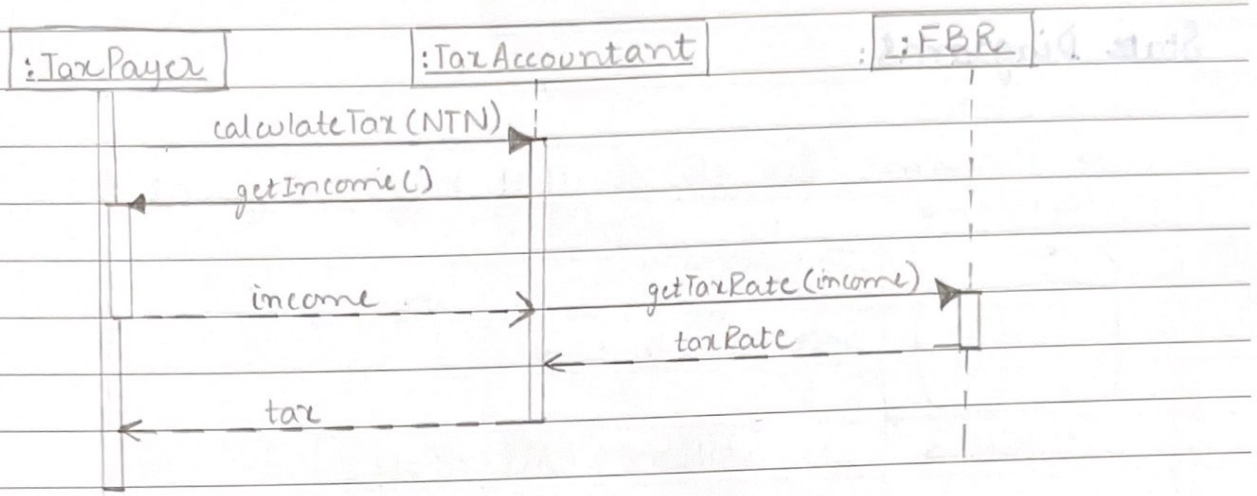


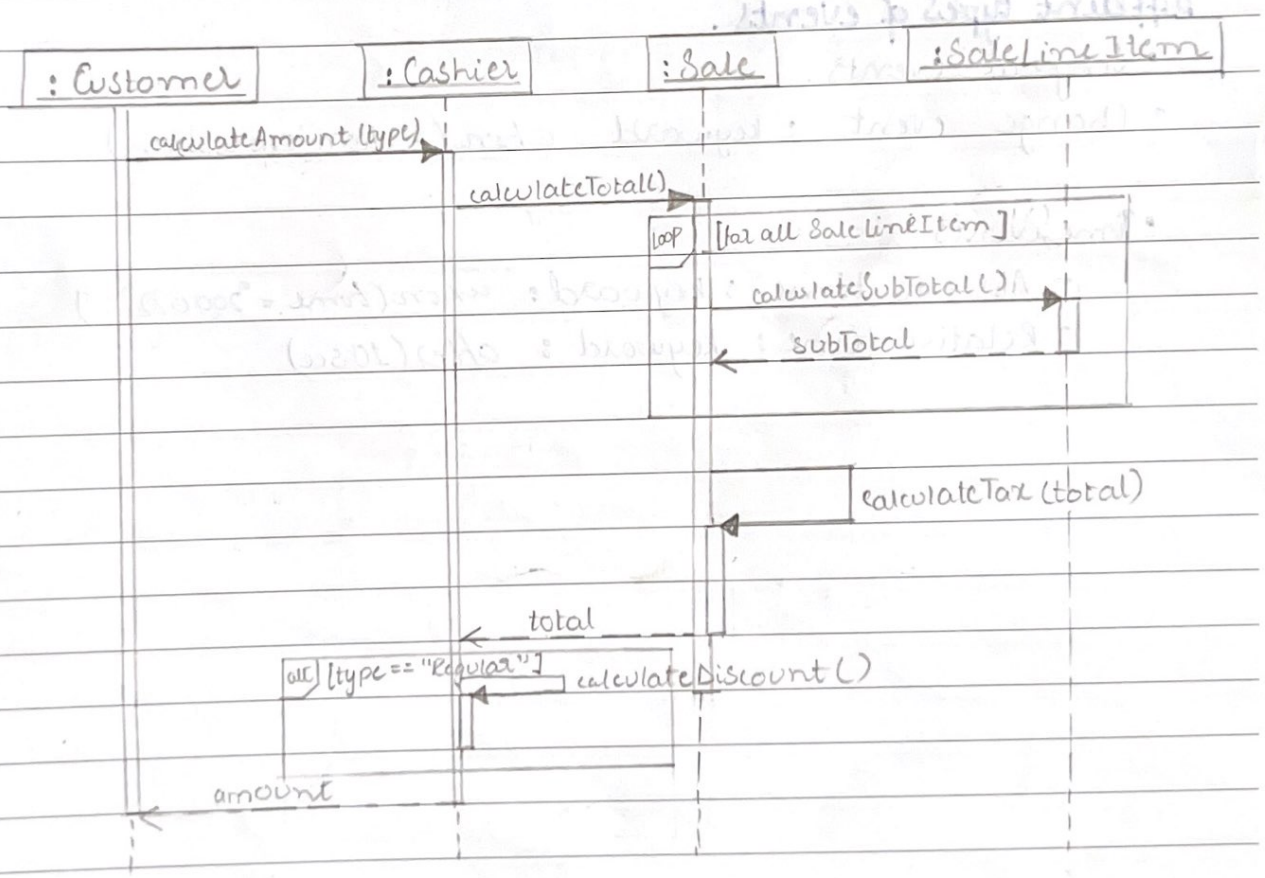


as long as condition true keep executing the loop.

- write math/eng/any language condition.

Date

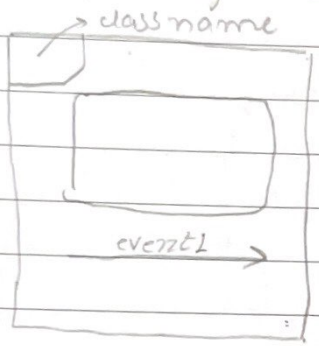

:TaxPayer
:TaxAccountant
:FBR
calculateTax(NTN)
getIncome()
income
getTaxRate(income)
taxRate
tax


:Customer
:Cashier
:Sale
:SaleLineItem
calculateAmount(type)
calculateTotal()
loop
[for all SaleLineItem]
calculateSubTotal()
subTotal
calculateTax(total)
total
alt
[type == "Regular"]
calculateDiscount()
amount

Date

State Diagrams:

- Change States
- State diagrams for classes that have temporal behaviour.



→ boundry.

upper camble case

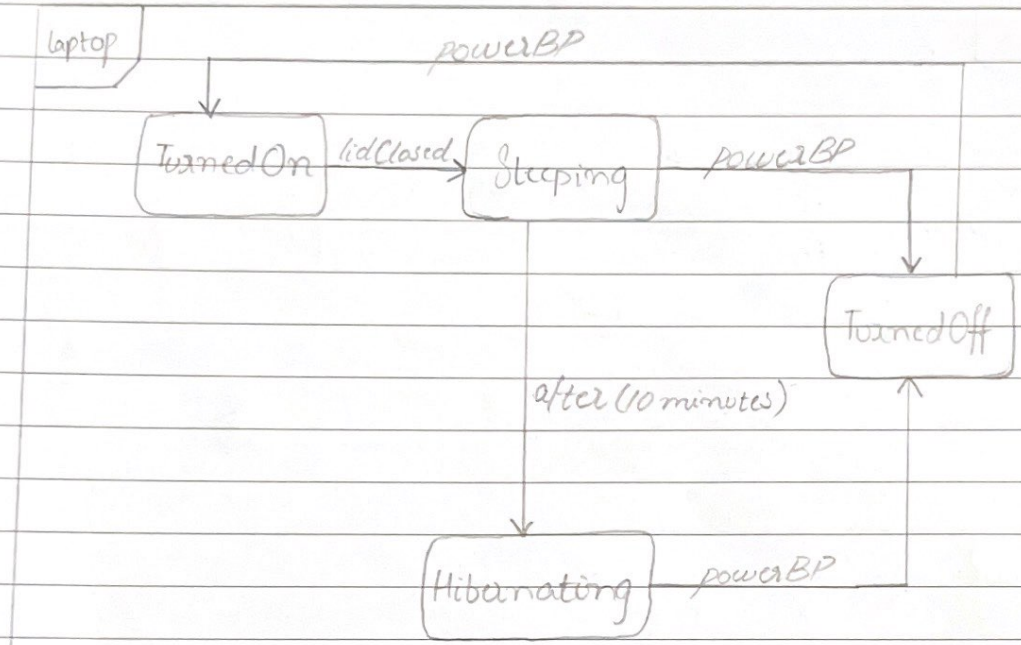
transitions b/w states.
event name - italics, lower camble case.

Different types of events..

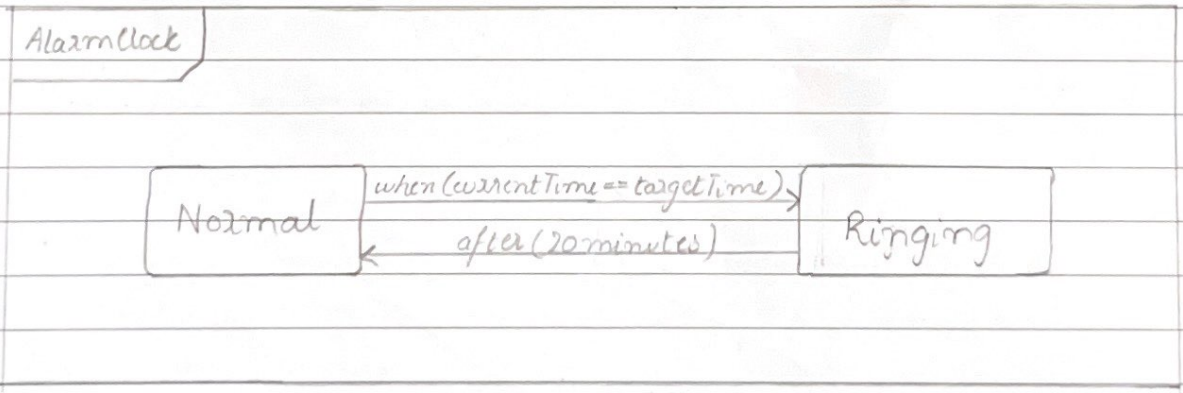
- Singnal events.
- Change event : keyword when (boolean expression)

- Time events :
  - Absolute time : keyword : when (time = "200022")
  - Relative time : keyword : after (10 sec)

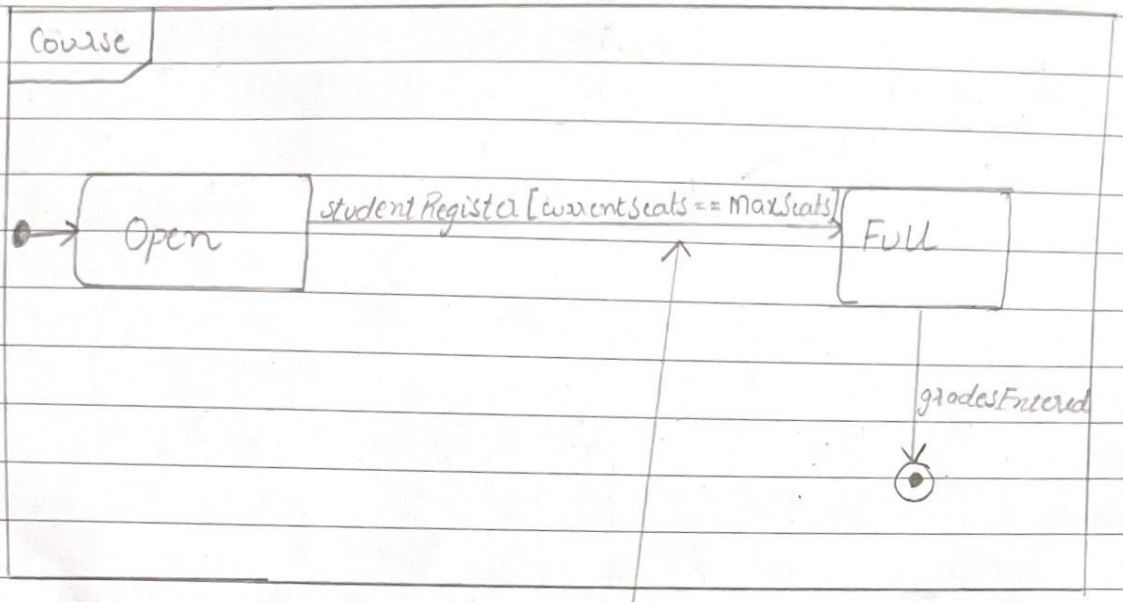
Date


laptop
powerBP
TurnedOn
lidClosed
Sleeping
powerBP
TurnedOff
after (10 minutes)
Hibarating
powerBP

loop SD


AlarmClock
Normal
when (currentTime == targetTime)
after (20 minutes)
Ringing

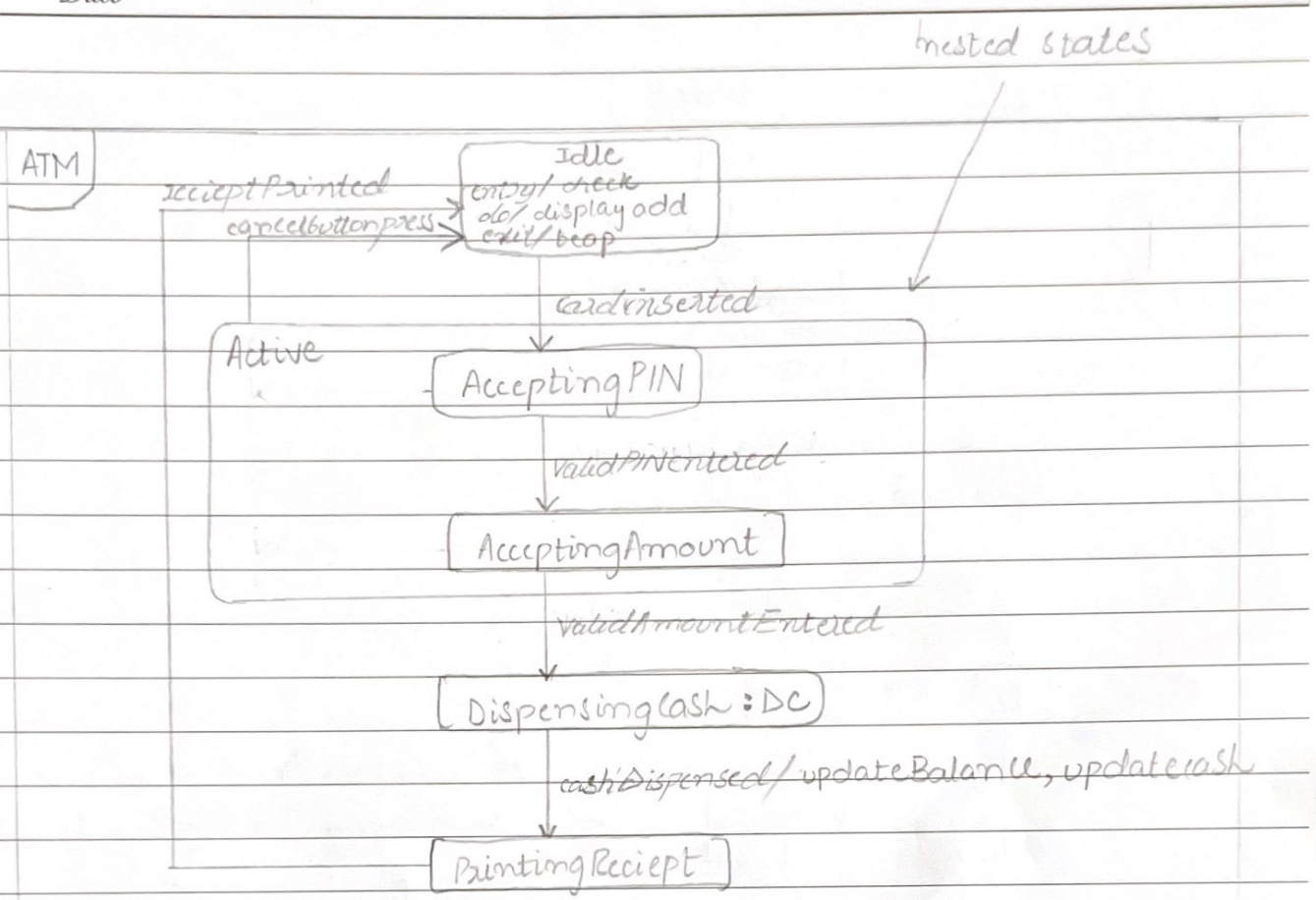
Date

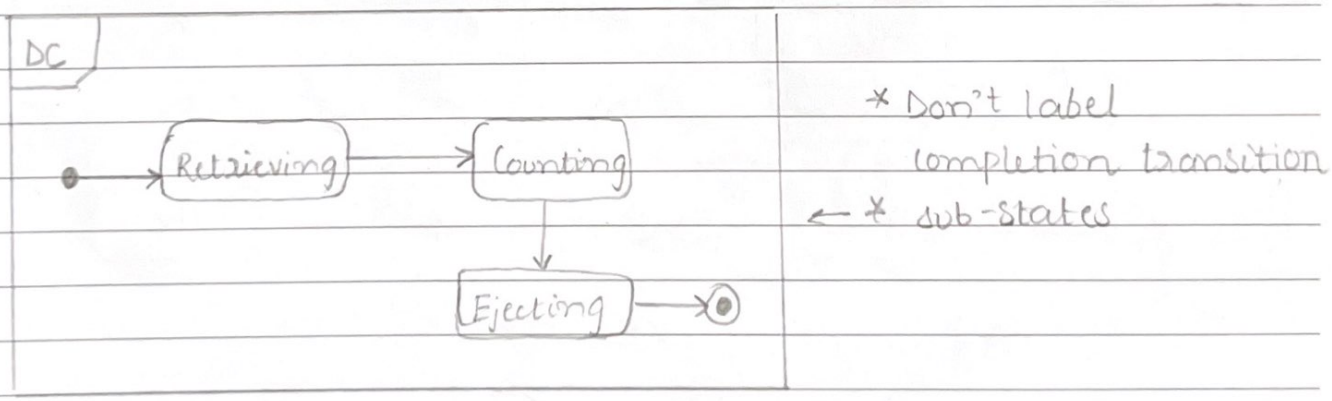


one short SD

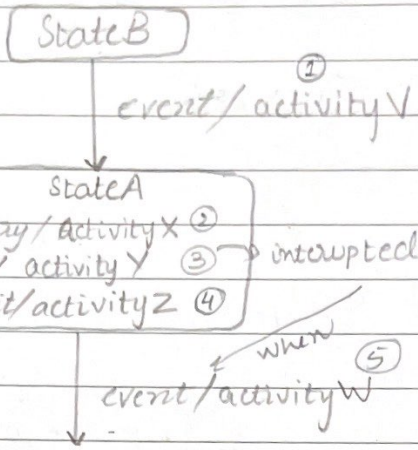
— Guard Condition.
- executes once.

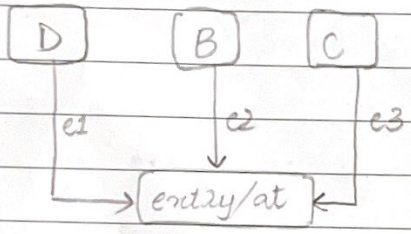
Date


nested states
ATM
Idle
entry/ check
do/ display add
exit/ beep
reciept Printed
cancelbuttonpress
cardinserted
Active
AcceptingPIN
validPINEntered
AcceptingAmount
validAmountEntered
DispensingCash : DC
cashDispensed/ updateBalance, updateCash
PrintingReciept


DC
Retrieving
Counting
Ejecting
* Don't label completion transition
* sub-states

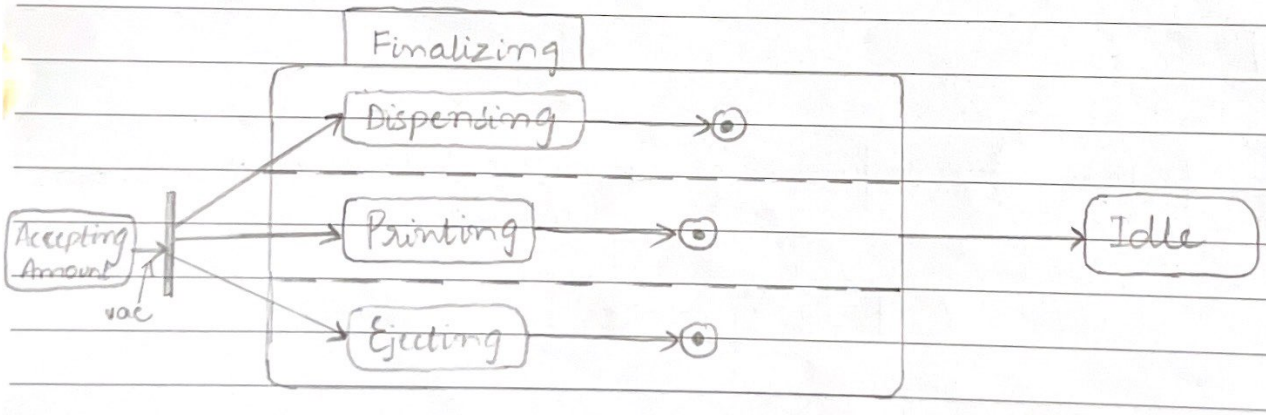
Date





← all activities that are happening at entry state

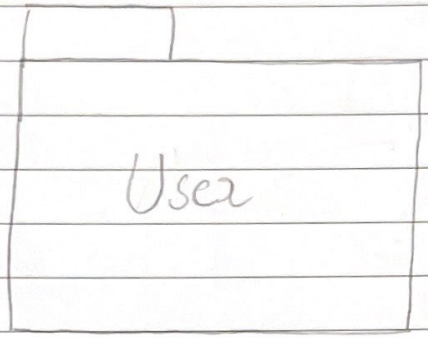
Concurrent States: (Needs to happen at the same time)



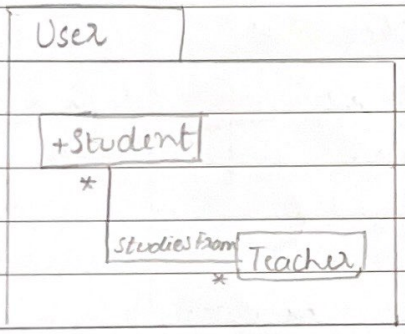
Date

Package diagrams:

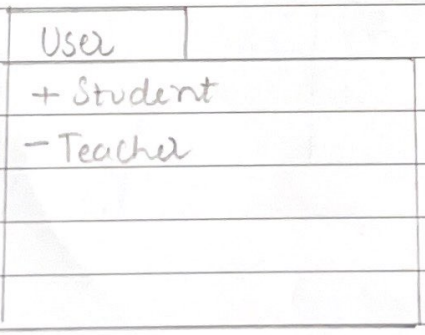
- Structural diagram
- Used to organize diagrams.



If not showing content.



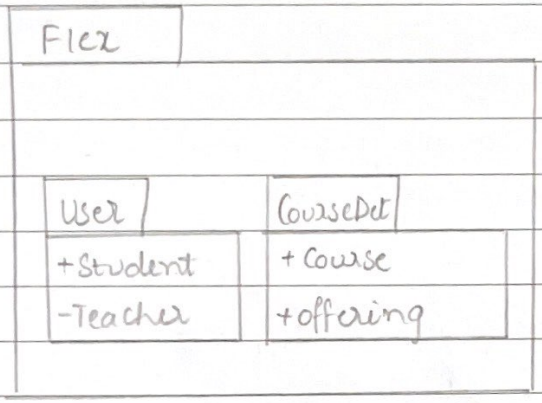
If showing contents



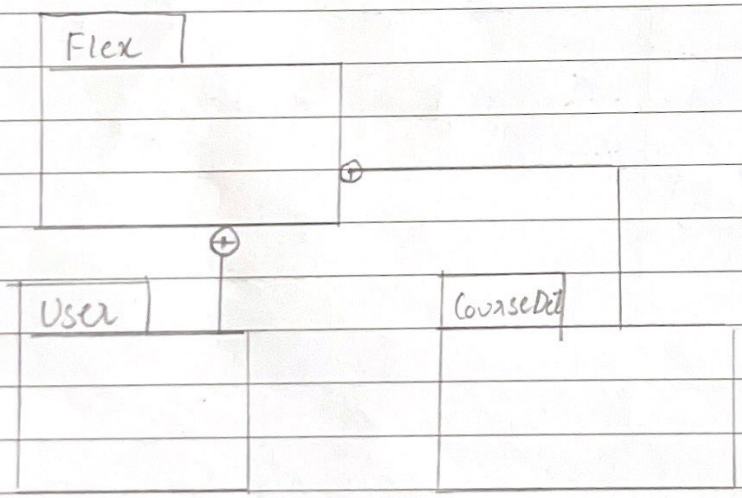
only two visibility available private and protected.

Date

Nested Packages.

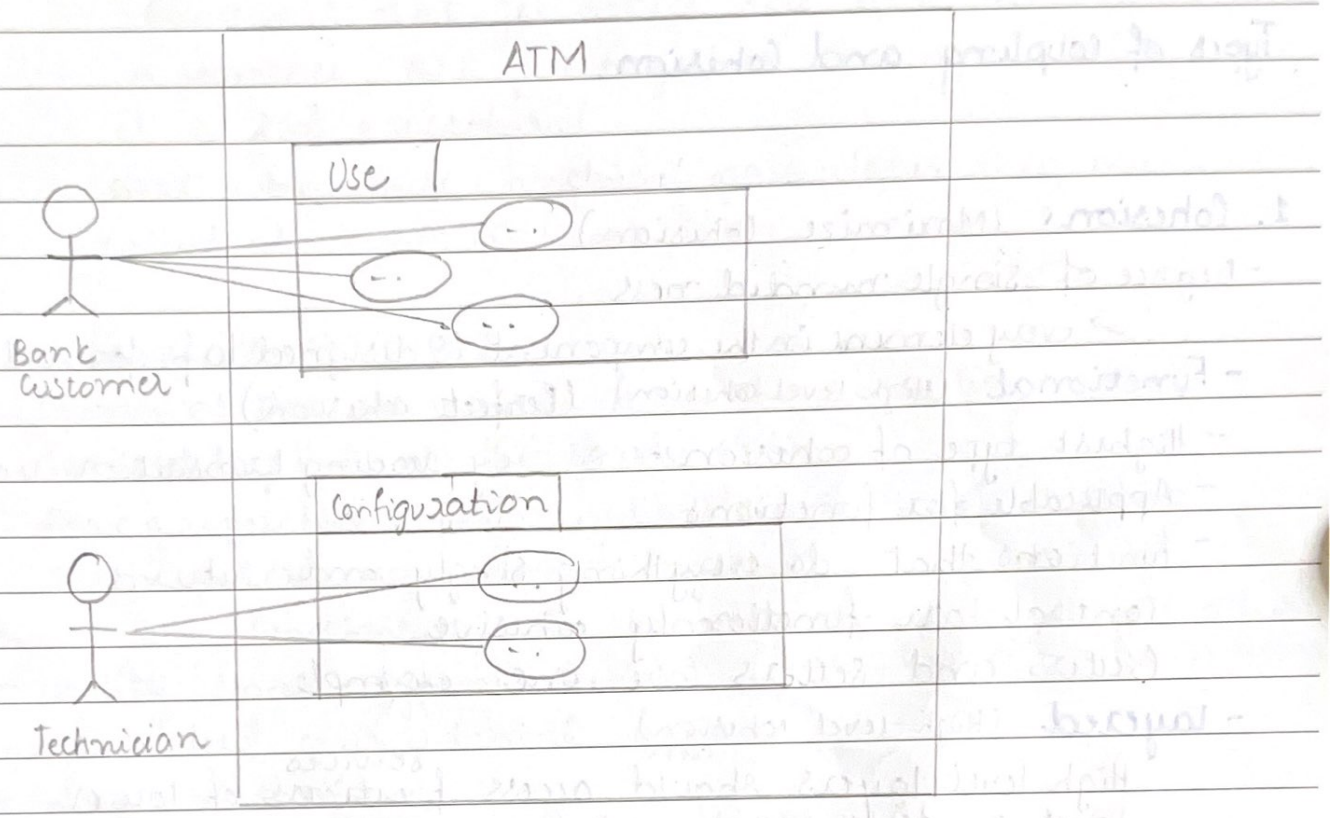


OR



Date

Organizing use cases



Date 1st March, 2023

Lecture after Mid-1:

Component-level Design:

Types of Coupling and Cohesion.

1. Cohesion: (Maximize Cohesion)
- Degree of single minded ness
  → every element in the component is designed to perform 1 task
- Functional (High-level cohesion) (Perfect cohesion)
  - Highest type of cohesion. e.g reading transaction records.
  - Applicable for functions
  - functions that do everything singly and return control are functionaly cohesive.
    Getters and setters are one example.
- layered (High-level cohesion)
  High level layers can should access services functions of lower level layers but it cannot happen vice versa.

- communication (High level) (Informational cohesion)

~~operations that access~~ to
keep the functions in the places where the information is
Two elements operate on the same input data or contribute towards the same output data.

Date

- **procedural** (low-level cohesion)
  - **sequential**: (a special form of procedural cohesion)
    Functions that get called one after the other, in sequence, are put together in same class is called procedural
    and when these functions pass data they are called as sequential cohesion.

- **temporal** (lower-level)
  Operations that are performed at the same time are lumped together in the same class.
  For e.g operations performed at the time of starting of computer are grouped together.

- **utility** (lowest type of cohesion) → Functions that are broadly associated with each other are grouped together in a class
  - also called co-incidental cohesion.
  - weak relation
  - no concept of sharing of data that gets stored.
  - functions can be called in any order.
  - e.g utility functions ~~for~~ like stats class, strings class.
  - elements have no conceptual relationship other than location in source code, e.g ⇒ substring, stringlength

logical:
Elements are logically related and not functionally. operations are related but functions are different.

Date

- **Coupling**
  - degree of connectedness.
  - ~~tom~~ Coupling should be low.

Types of Coupling:

- **content** (worst-type of coupling) — one module can modify data of another module.
  - Making data members public enables content coupling
  - Making friend classes / functions enable content coupling.
  - Inheritence is also risky. (violate information hiding)
  - ~~Shared~~ Everyone can access it and it is shared.
- **Common** (bad-type of coupling)
  - Shared data enables common coupling
  - Global variables " "
- **routine call** : one function is called into another function
- **data** : function call each other and pass data.
- **control** : pass data which controls the flow of a function, we call it as control coupling. (passing control information)

functions communicate by passing only data.

- **Type-use coupling**: (frequently done coupling)
  - When an attribute in one class has the type of another class.
  - aggregation / composition.
  - for e.g `b : B` b is a variable in class A.

iterator

Date

- Stamp coupling:
  When we use another class in one classes' signature.
  For example → complete data structure is passed from one module into another module.
  f1 (B p1 : B);
- external: module depend on other modules, external to
- import Software being developed or to hardware e.g external file, device format etc.

Solid Principles:

S - single Responsibility
O - Open-closed
L - Liskov Substitution
I - Interface Segregation
D - Dependency Inversion

Single Responsibility

Every class should have a single responsibility.
It should do only one thing.
- If the class has too many responsibilities, it should be split.

Open-closed

Classes should be open for extension but closed for modification.
- Don't change existing code, but you can add sub-classes