Software Design & Analysis

Lecture # 3

Software Supplier (SS/SC)

Software clients

Use Case Diagram

Chunk of systems functionality

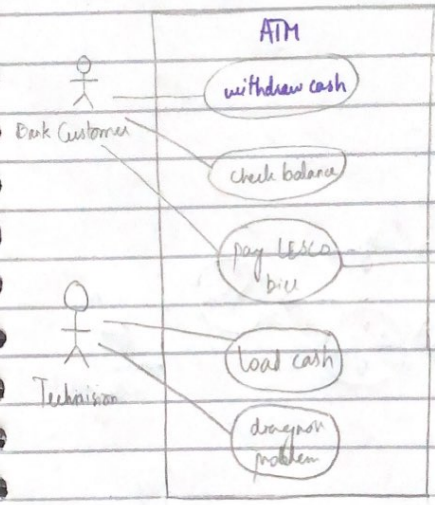
→ use case

→ actor (human) → for non human actors

——/ → participation

→ systems boundry

* ATM Software Use Case Diagram (Understanding Problem (Requirements part))



- All use case names must starts with verbs and use small case letters except special case words. Use spaces instead of under.
- Actor names must be in upper camel case.
- It should never have a dangling actors or use case
- Many to many relationship between actors and use cases.
- An actor always interacts directly with system
- Scenario is specific instant of use case

Software Design & Analysis

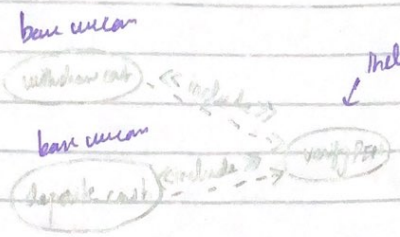
Lecture # 4

Use cases are used to document requirements / system requirements

Use case:

* Inclusion is including a use case in other use cases

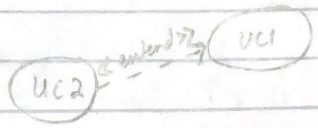
* UC3 is part of the behaviour sequence of UC2.



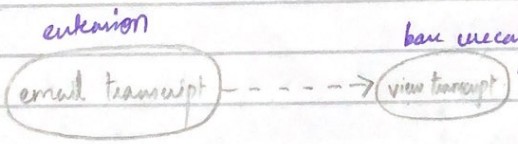
withdraw cash
Include UC1 here

Use Case

* Extention



UC2 is optional which means when you execute UC1 it is not necessary for UC2 to execute everytime



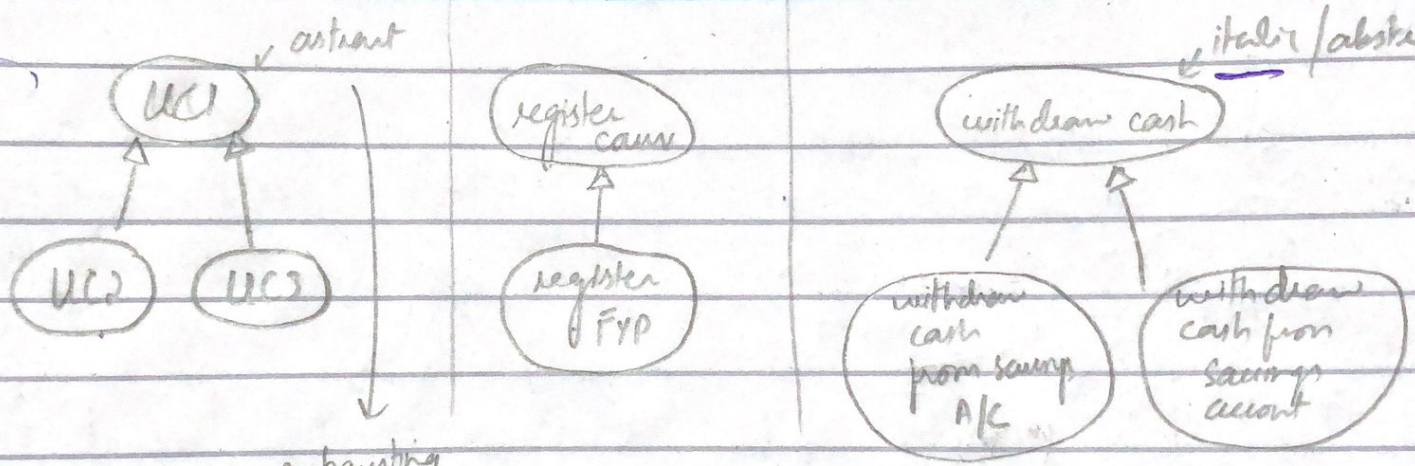
Difference between extension and inclusion

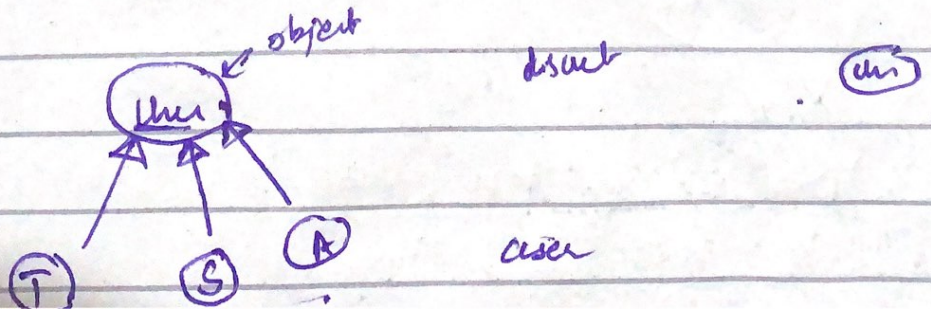
- Inclusion main mandatory to execute the included usecase while in extention it is optional
- Arrow heads are pointed towards the included use case from the base case.

Arrow heads are pointed to base case in extension

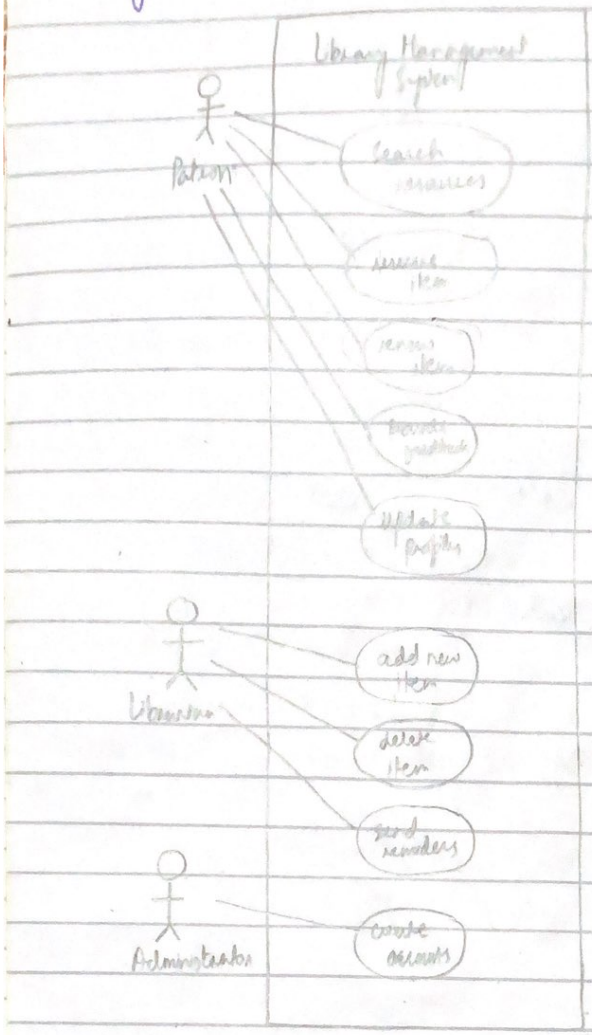
ITALICS

Inheritance : Concrete / Abstract



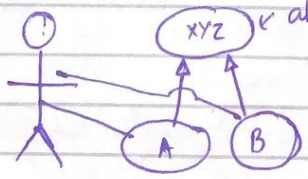


Activity:

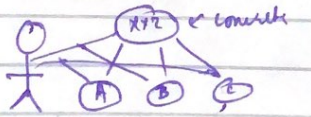


- Multiple Extension is allowed
- Multiple Inheritance is not allowed

In abstract we cannot we do not participate in abstract oval



In concrete we make all participation



Software Design & Analysis

Lecture # 5

Activity Diagrams ★

Activity Diagrams are very similar to flowcharts.

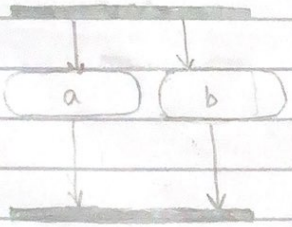
→ use case ✗ Naming conventions are same as usecases.

→ All activity diagrams start here there can only be on start

→ Ending symbol which can be one or more

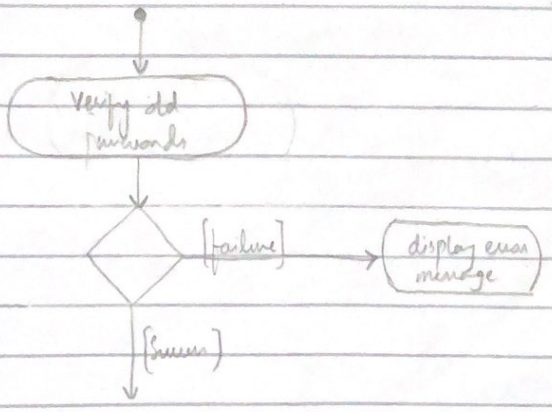
→ flow of control

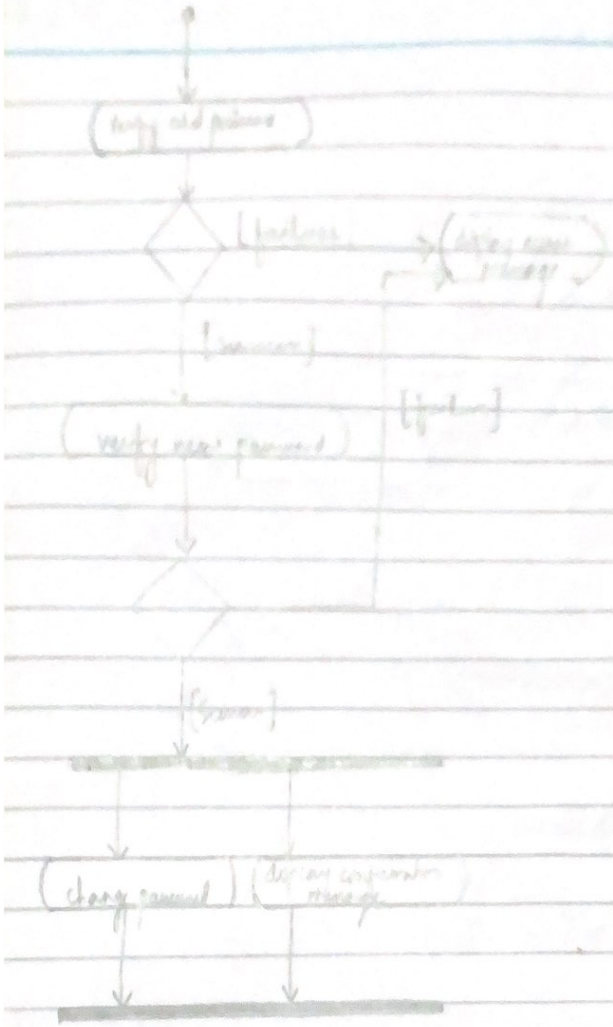
→ Decisions (True, False, if else, switch case)



→ show concurrency
a, b run at same time

Chang Tenward

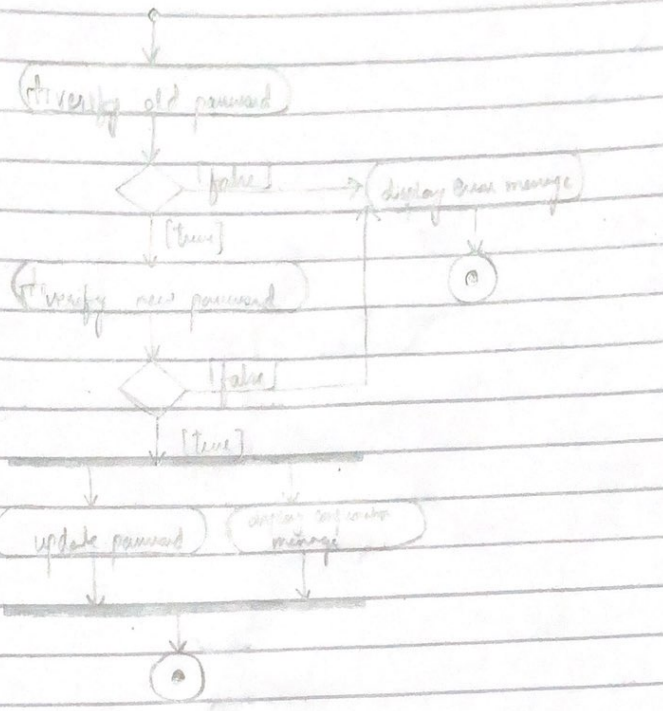


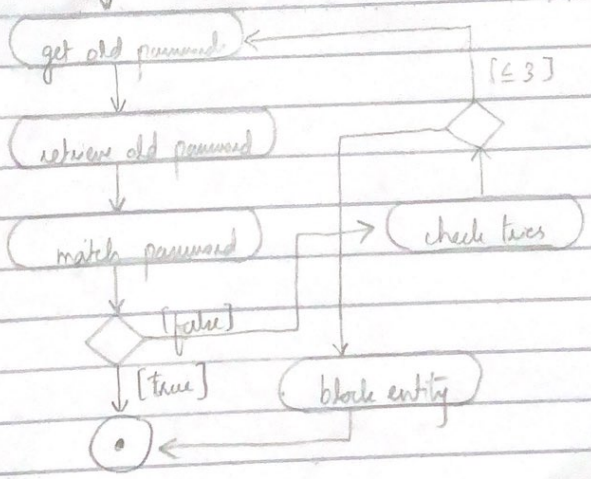


Software Design & Analysis

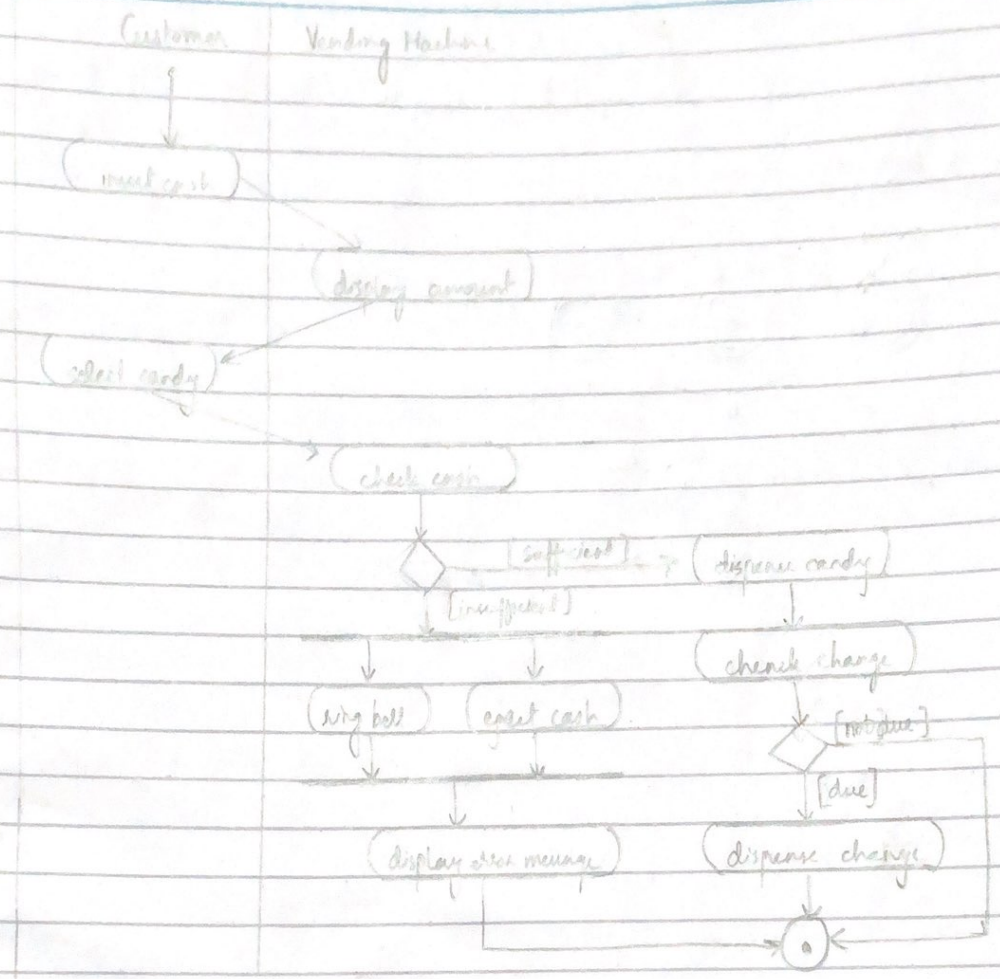
Lecture # 6

Starting Symbol has exactly one outgoing arrow.
A diamond should always be preceded with some activity
There can be more than one incoming arrows to the ending symbol





Expansion of verify old password


Customer
Vending Machine
insert cash
display amount
select candy
check cash
[sufficient]
[insufficient]
dispense candy
check change
ring bell
eject cash
[not due]
[due]
display error message
dispense change

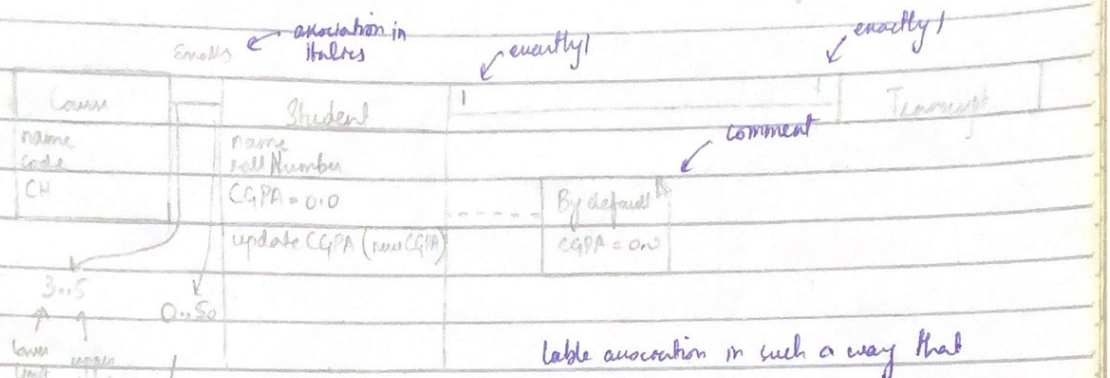
[due]

Software Design & Analysis

Lecture # 7

Analysis Class Diagram → static structure
↳ Structure in problem phase

class name { ← written in upper camel case center align (noun whose first is singular)
list of attributes { ← lower camel case left align (no all caps for abbreviations and acronyms)
list of operations { ← lower camel case left case



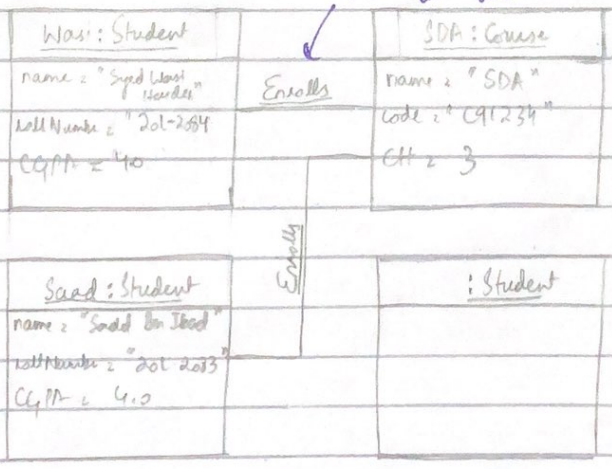
Multiplicity read from left to right

Lable association in such a way that it is easy to read from left to right or top to bottom.

Object diagram for above example

Association maps to links which is underlined.

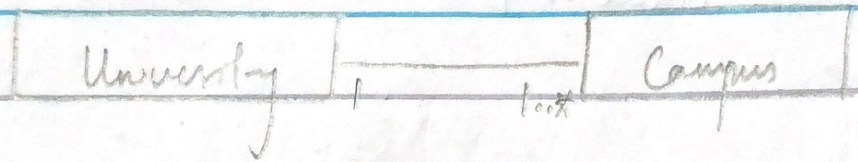
Every object has uniqueness even with the same attributes.



Anonymous object

Single Point

* f*(a



l..u
5
=0..
1..*
3..*

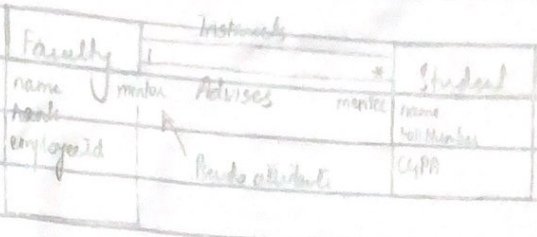
Multiplicity constraints the cardinality

↓ general lower / upper limit.

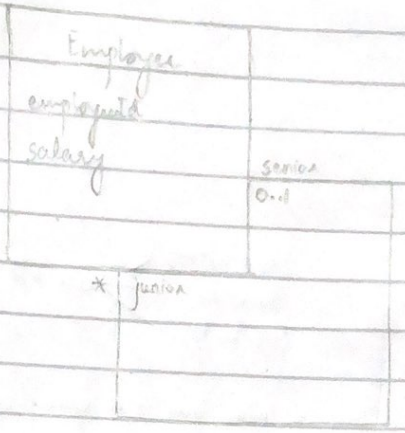
↓ actual count

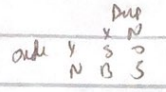
Software Design & Analysis

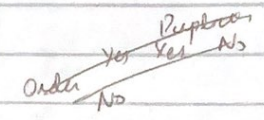
Lecture # 8

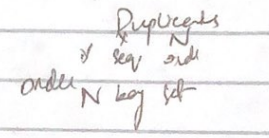


Self Association:

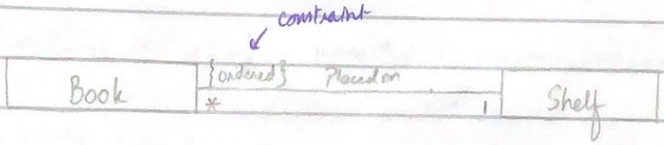


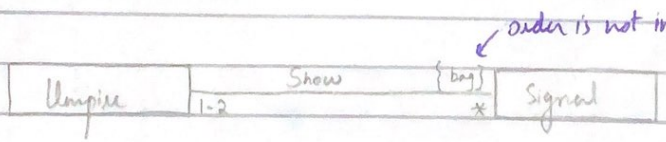




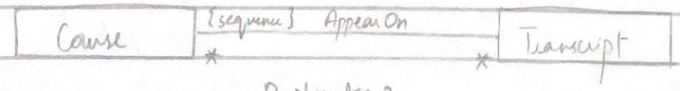


Sometimes the order is important





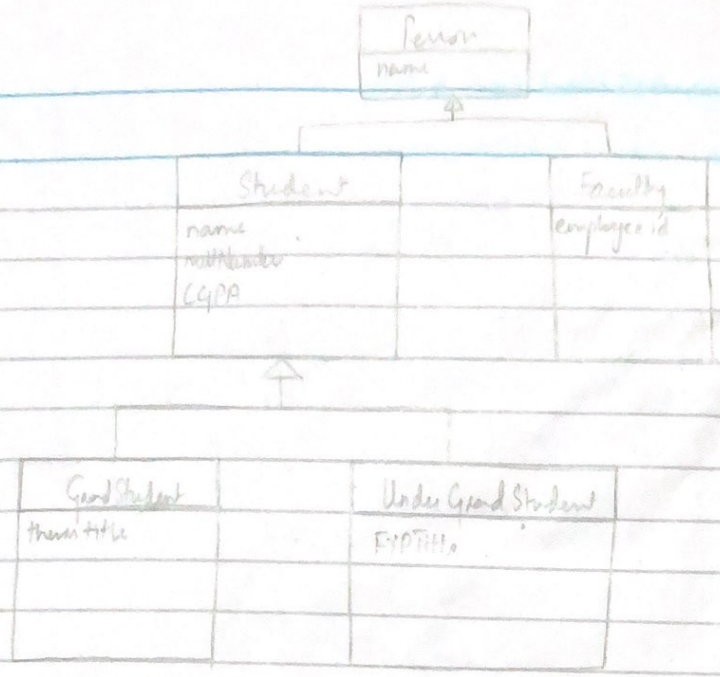
seq order
bag set

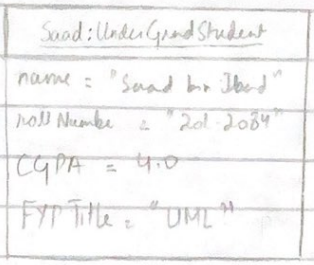


Order?	Duplicates? Yes	Duplicates? No
Yes	sequence	ordered
No	bag	set

Order	Dup: Yes	Dup: No
	Seq	ordered
	bag	set

Section 3.5



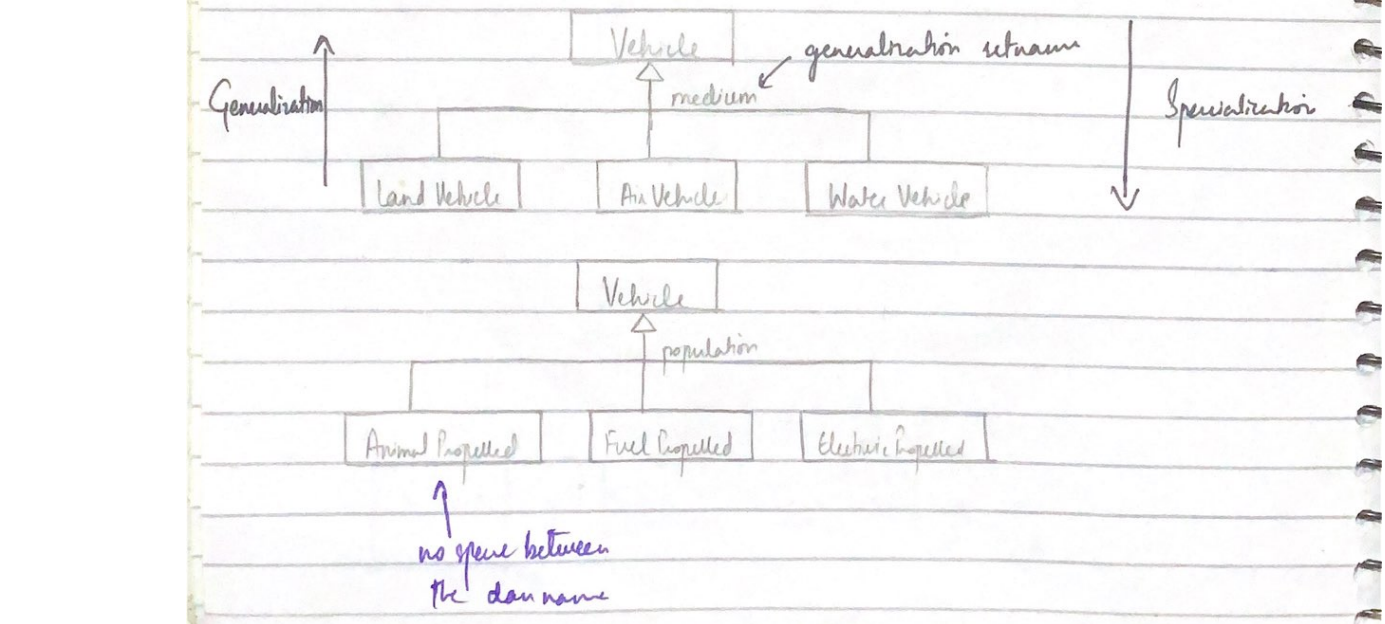


Annotations, Operation, Attributes → are all inherited

- Inheritance is a binary association

Uml allows to generalization according to set name

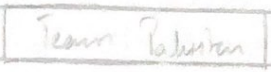
- No plurals in class names

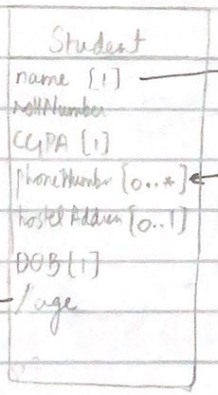


Software Design & Analyses

lecture 9

Qualified Association



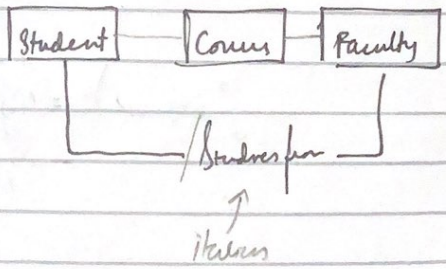


By default

optional multivalued

derived attributes

Derived associations can also be there

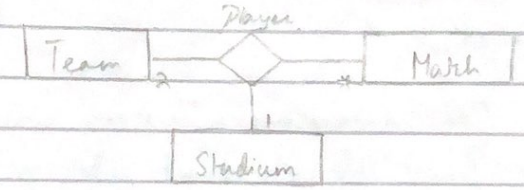


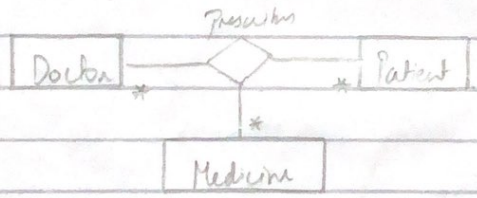
italics

Course	
maxSeats } class scope	what means this two same value for every object
code. }	
name } object scope	
CH	
currentSeats	{currentSeats ≤ maxSeats} ← write on left of class
updateMaxSeats(newMaxSeats)	
updateCurrentSeats(newCurrentSeats)	

Tertiary Associations:

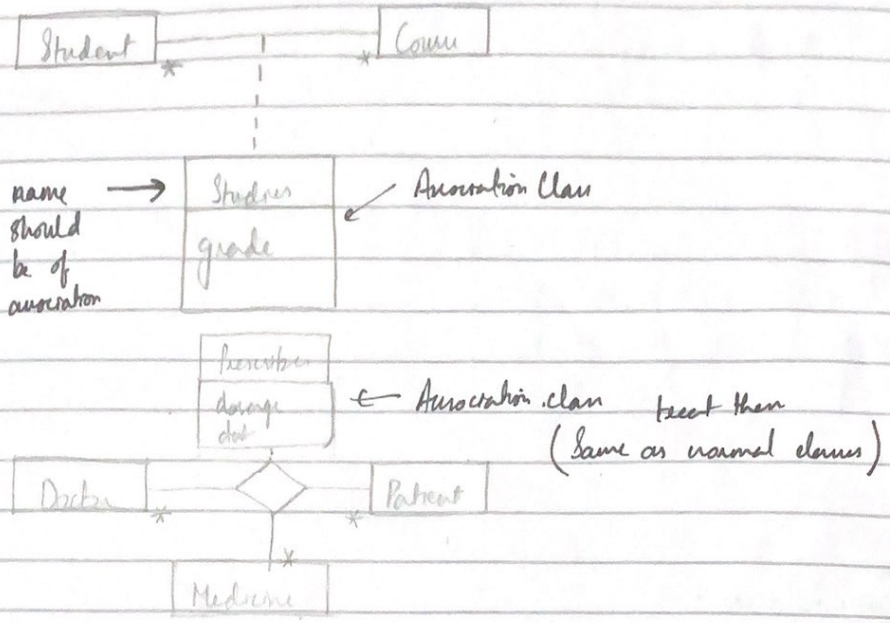
Keep two ends fixed at 1 and ask the question from the third one



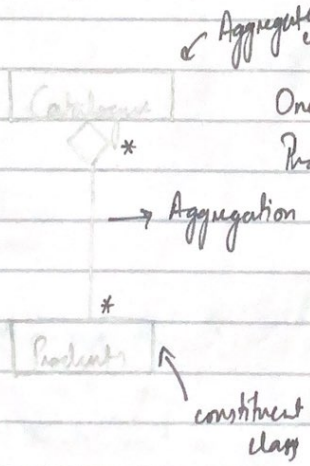


← True generic ternary association.
Means it cannot be decomposed further without losing information

* Binary Association (Association)



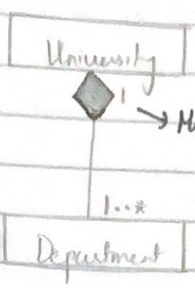
* Aggregation (Special Type of Association)



Two properties of aggregation

① Anti Symmetric ✓
② Transitive in nature ✓

Composition · (more restrictive than aggregation)



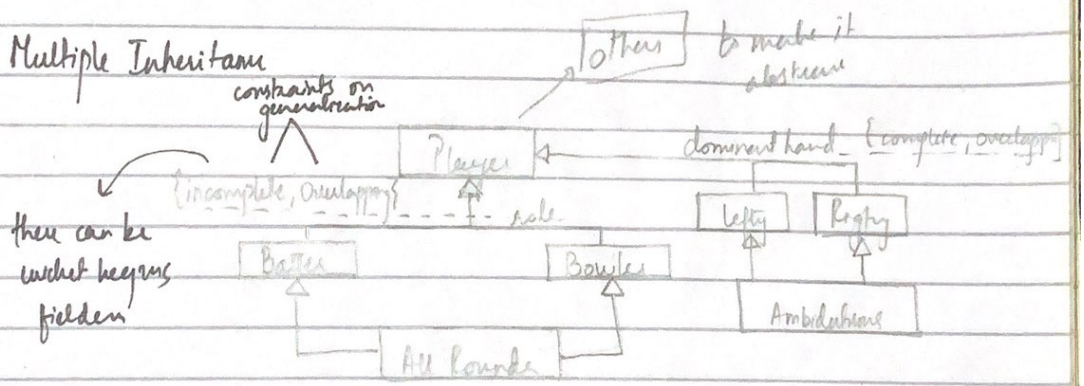
Co incident life time if University gets destroyed the departments also get destroyed.

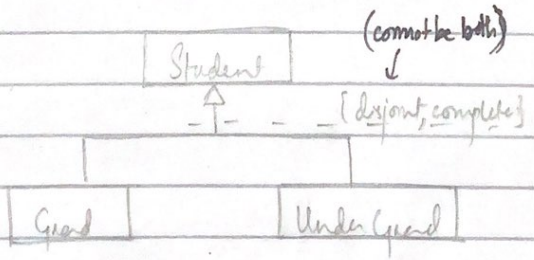
Properties

① Multiplicity max 1 on composite side

② Coincident life time.

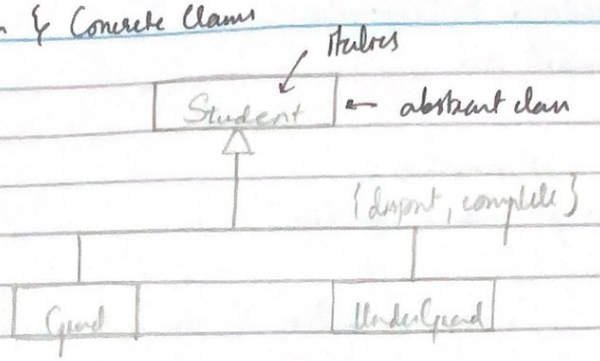
★ Multiple Inheritance





cannot be instantiated

* Abstract Class & Concrete Class

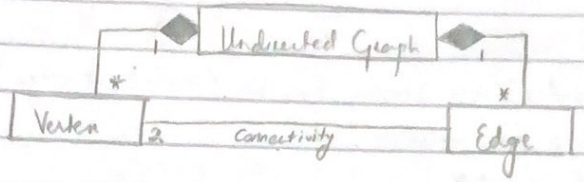


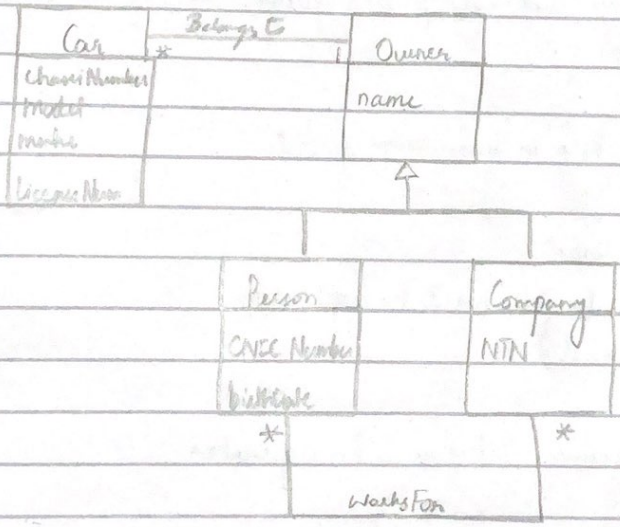
* Good practice to make parent class abstract

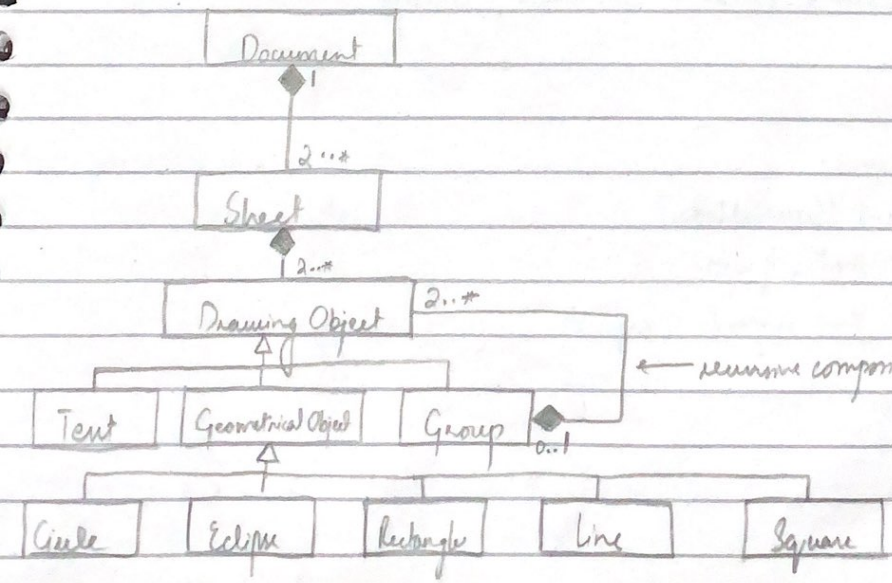
Software Design & Analysis

Lecture 10

Practice Case Studies







Software Design And Analysis

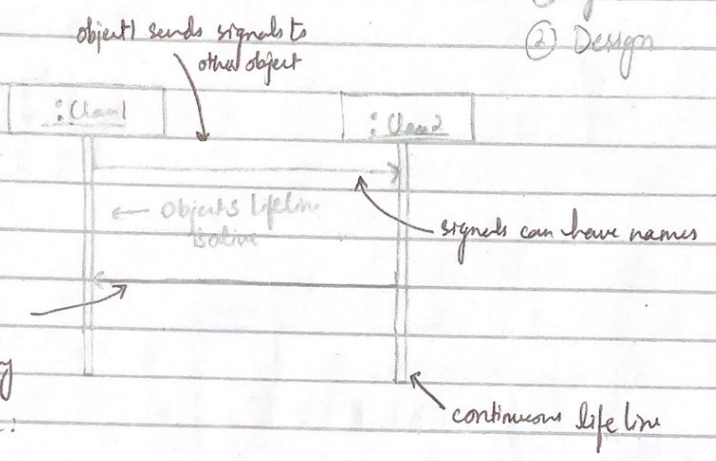
Lecture # 11

MODELS

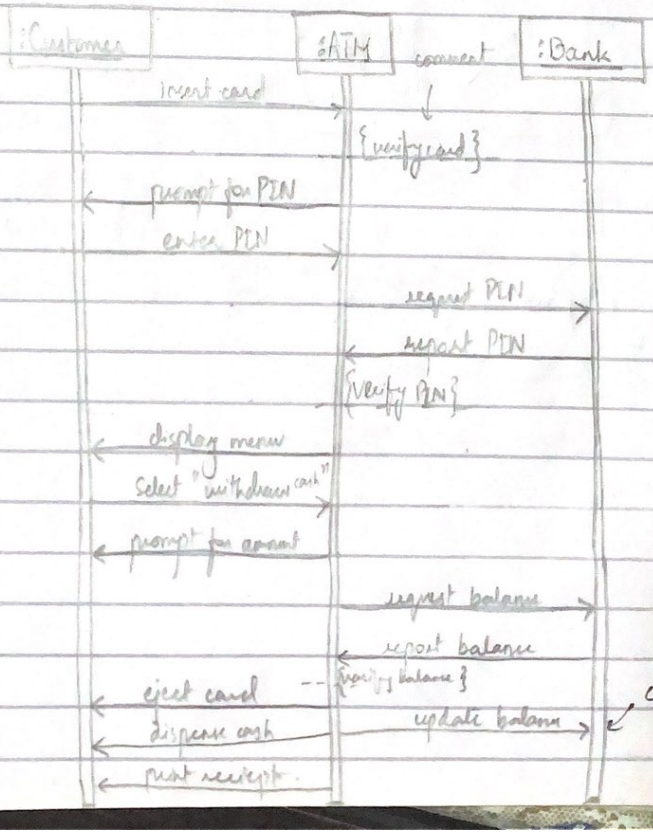
- class — Static in nature
- State
- interaction — Activity
  - Use case
  - Sequence diagram
    - dynamic in nature

System level diagrams

Two types
1. System level
2. Design

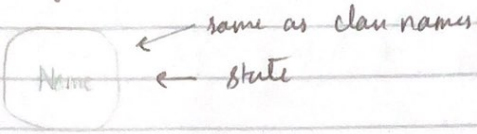


Example:

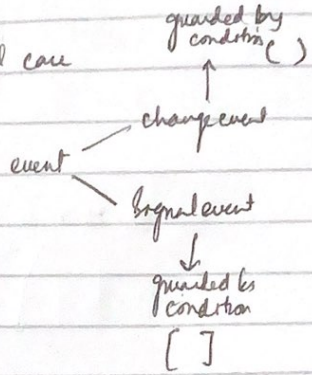


UC	Represents	SISD
1	1	1

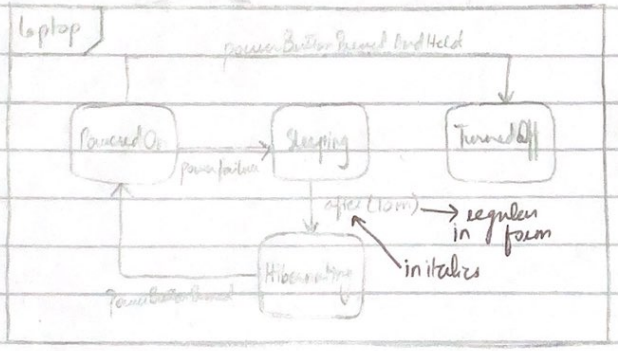
* State Diagram

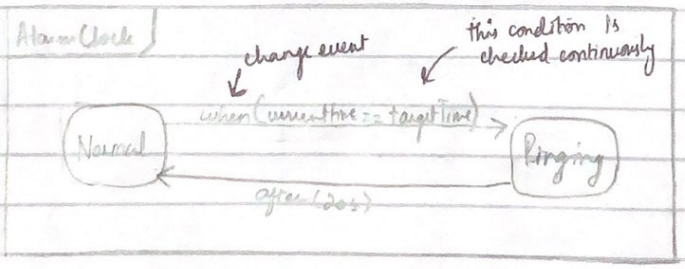


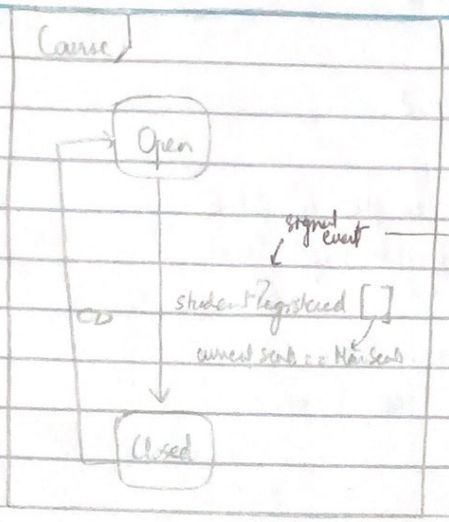
event → Event name in italics in lower camel case



Class	





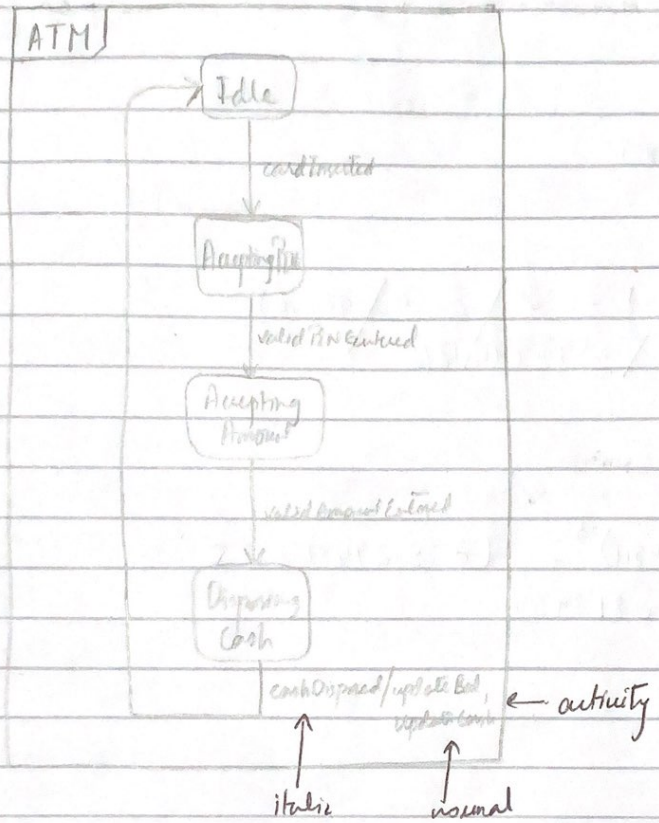


→ it ~~happens once~~ is guarded by a condition that runs when event happens

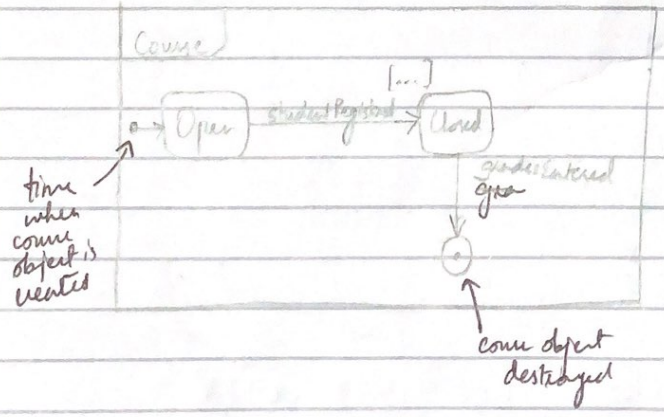
Software Design & Analysis

lecture # 12

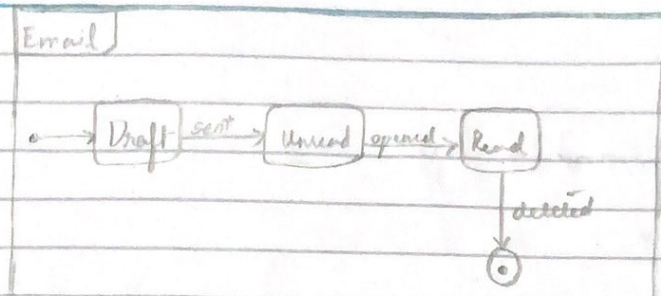
* Continuous loop state diagram

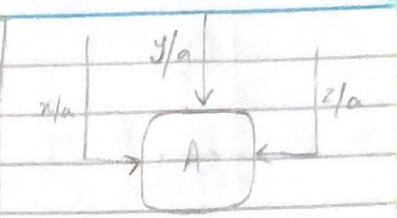


* 1 shot state diagram



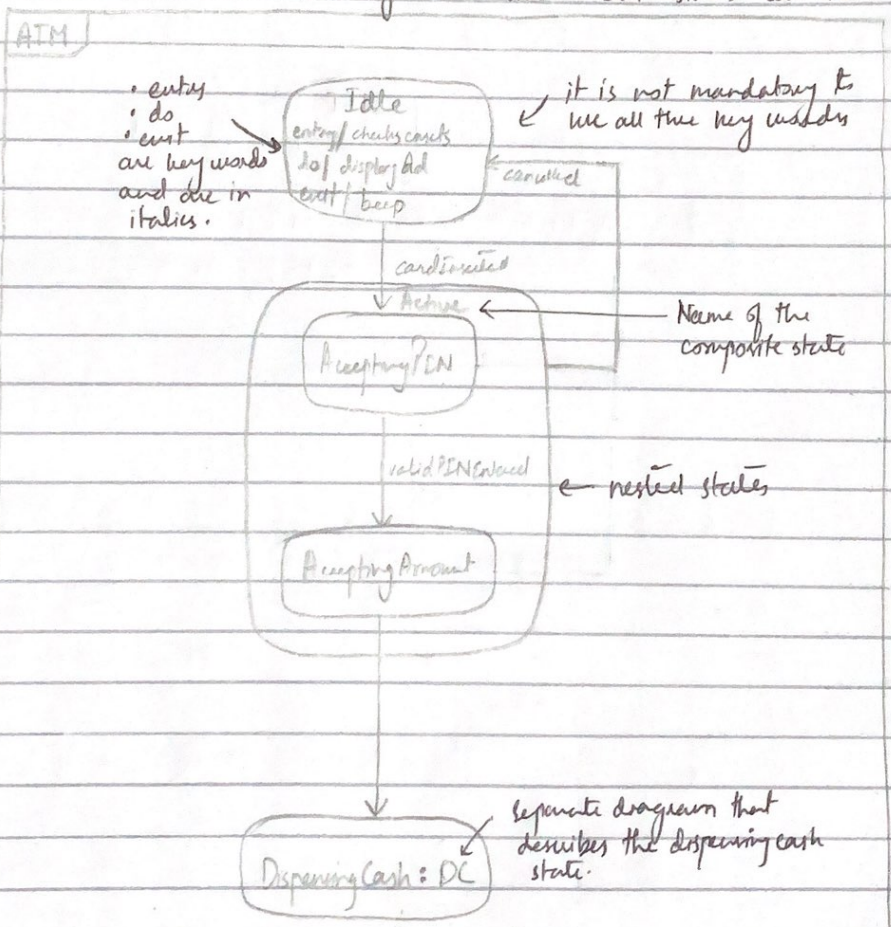
All states should be at same abstraction level.





If there is a common event in incoming arrows then you write the activity 'a' the entry activity of state a and vice versa in case of outgoing arrows

Till now we have seen activity outside states but states can have activities inside.



Ideal.