Course: Object Oriented Analysis & Design

Feasibility study: এক system থেকে আরেক system এ shift করার আগে যে analysis করা হয়।

For software design we use modeling language.

- Model is a simplification of reality.
- It provides the blueprint of a system.
- UMS → University Management System.

BENEFITS OF MODELING

1) Model helps us to visualize a system as it is or as we want it to be.
2) Model permit us to specify the structure or behaviour of a system.
3) Model give us a templet that guide us in constructing a system.
4) Model document the decission we have made.

UML → Unified Modeling Language.

OMG → Object Management Group.

Chapter 1: What We Model?

Building Block of UML

UML include three kinds of building block.

(1) Things (2) Relationship (3) Diagram

Things

There are 4 kinds of things.

1) Structural things (Noun/ static parts of UML)
2) Behavioural " (verbs/dynamic " " ")
3) Grouping " (Organizational " " ")
4) Anotation " (explanatory " " ")

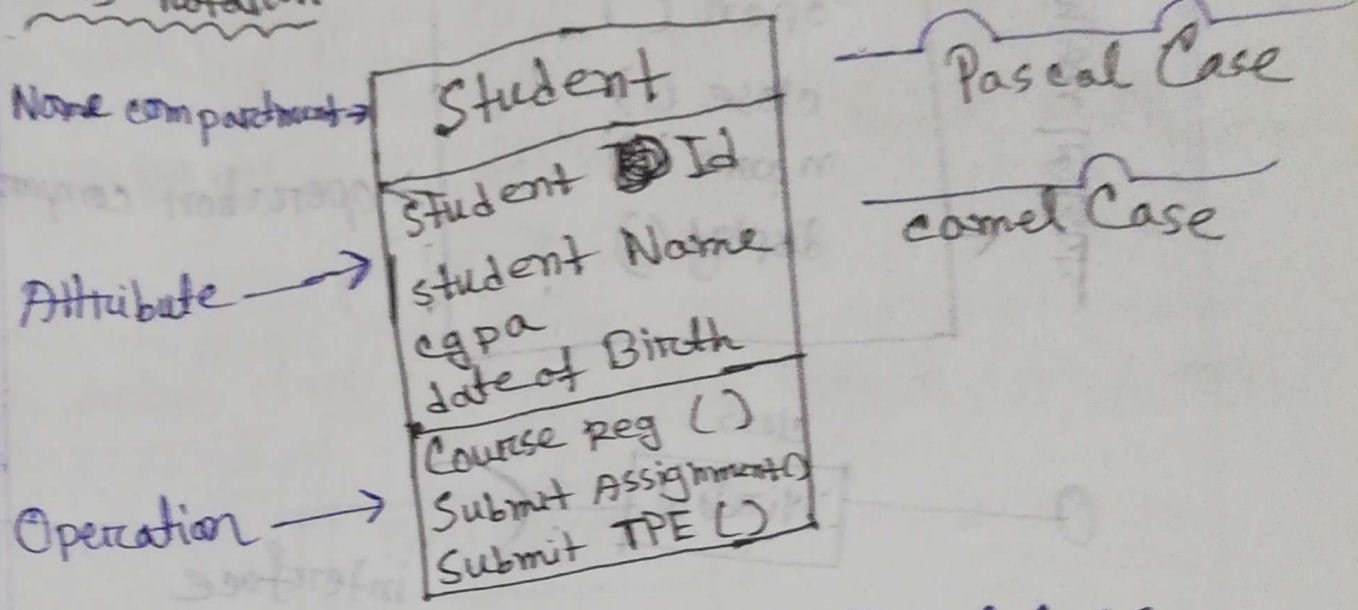
Structural things

7 kinds of structural things.

i) Class ← ch-4&9
ii) Interface ← ch-4
iii) Collaboration
iv) Use case
v) Active class
vi) Component
(vii) Node

1. Class: class is the description of a thing or anything.

class notation

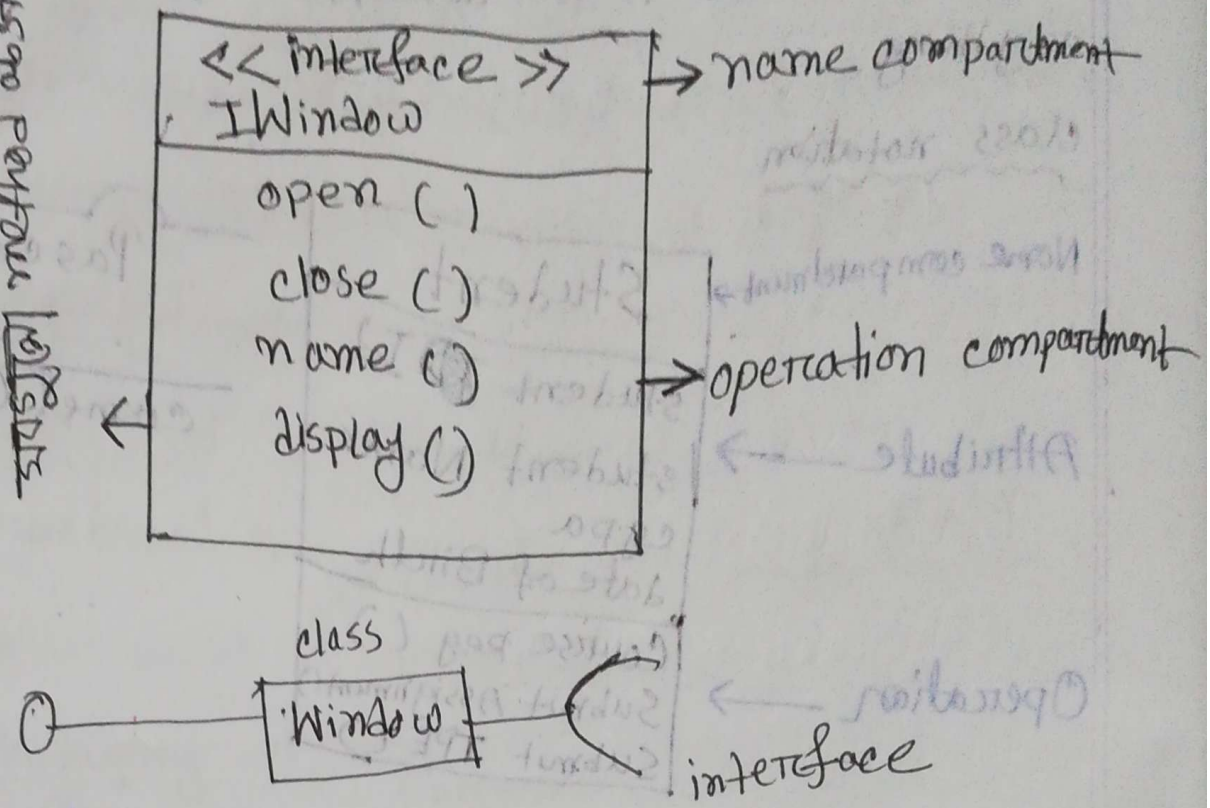


* class name should be in Pascal Case
* Attribute " " " " camel Case
* Operation " " " " Pascal Case

2. An Interface:

```
void RegesterCourse (); //abstract method
//{
  //no body
//}
```

```
interface I student CourseReg
{
    void RegisterCourse ();
}
```



* Interface is fully abstract. You can't instantiate in interface but you can create reference.
* Instanciation is the process of creating object.

Object Creation

Obj ref variable

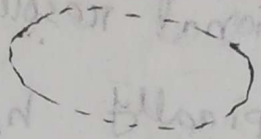
```
Student nadim = new student( );
```

stack

Object (heap ↳ create 2?)

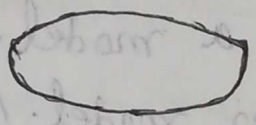
* Interface is not extended by a class, but
* " is implemented by a class.

3. Collaboration:



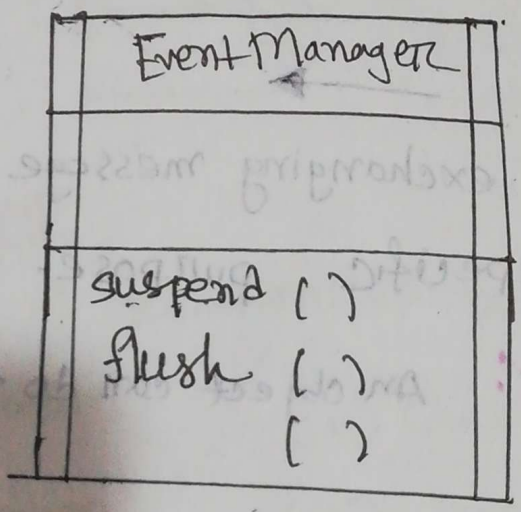
Class, interface work together to provide some functionality within the system.

4. Use case: specify the behaviour of a system.

 → (elipse)

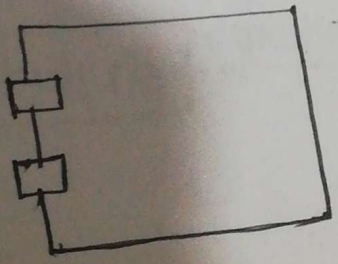
5. Active Class:

Class activity control করে।



6. Component: Package of class, interfaces, and collaboration. Total program is divided into various parts or module.

* Component is a collection of lots of collaboration.



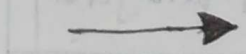
7. Node: computational resource that exists at run time, typically h/w resources.

* যে hardware system এ run করে।

Behavioural Things

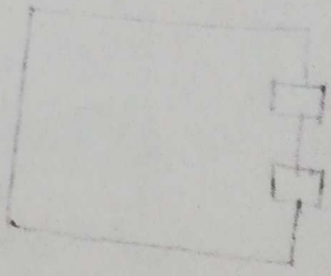
- These are verbs of a model.
- Dynamic parts of UML model. (সময়ের সাথে change হয়)
- Representing behaviour over time and space.

1. Interaction:

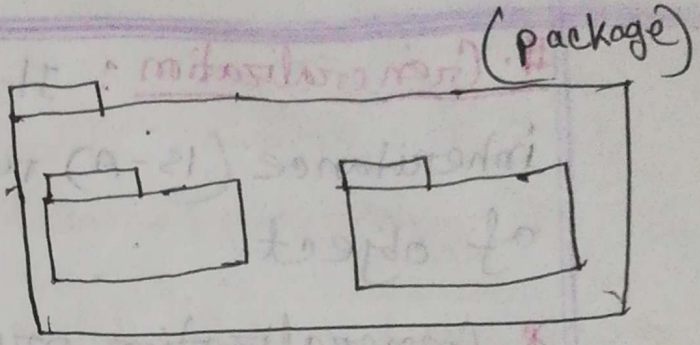


A set of object exchanging message to accomplish a specific purpose.

2. State machine: An object can do multiple behaviour.



Grouping Things



Anotational things

- Anotational things are explanatory parts of UML model.
- There is one primary kind of Anotational thing, called a note.

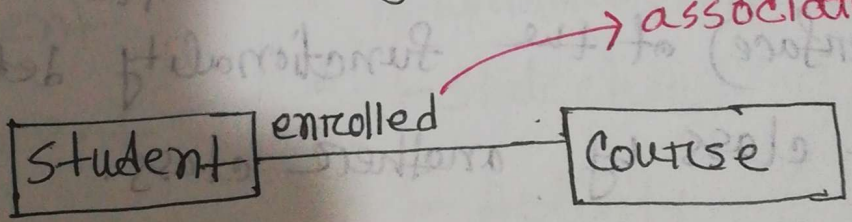
Relationship

4 kinds of relationship in the UML

i) Association (ii) Generalization

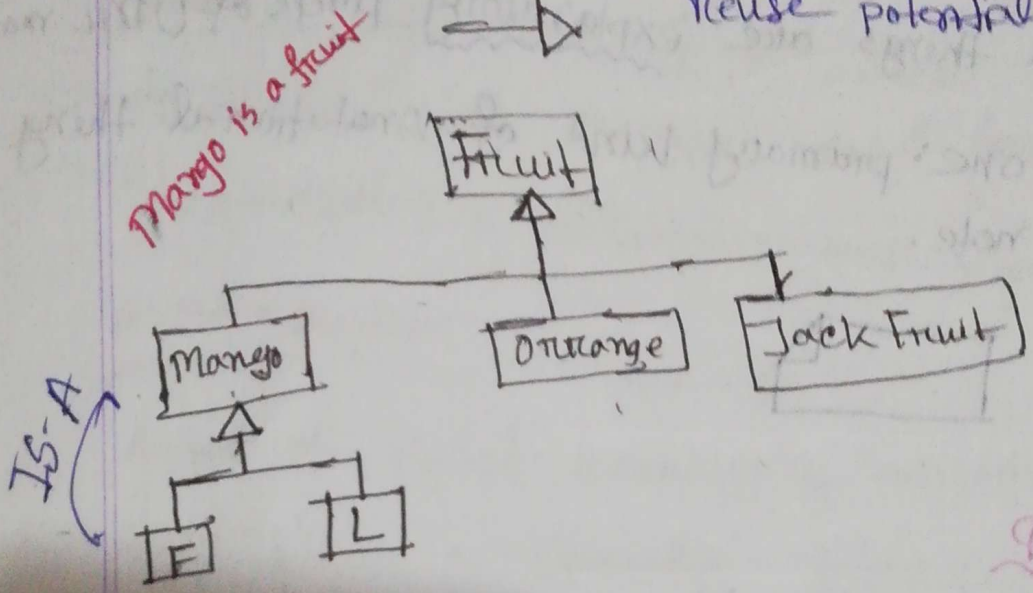
iii) Realization (iv) Dependency

i: Association: Describe a set of link. A link is a connection between object.



II. Generalization : It basically describes inheritance (IS-A) relationship in the world of object.

* Generalization provide code reusability / reuse potentiality.



* The top down approach of a inheritance hierarchi is known as specialization.

* The bottom up approach → generalization.

III. Realization denotes the implementation (interface) of the functionality defind in one class by another class.

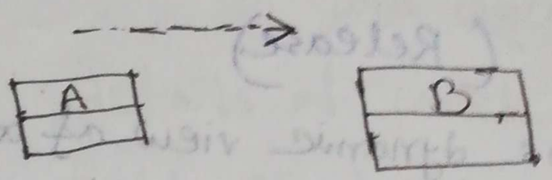
class C implements I

~~IV. Dependency:~~

* Interface এর কোন reality নেই।
* " is an indirect form of inheritance.

IV. Dependency:

is a relationship between two things in which change in one element also affects the another one.



Class A depends on B

Diagram In UML

* graphical presentation of a set of elements.

Structural Diagram Static	Behavioral Diagram Dynamic
- Class	- Use case
- Object	- Sequence (Interaction)
- Component	- Collboration
- Deployment	- Statechart
	- Activity

Class diagram: provides the static design view of a system.

Reality ↑

Object diagram: provide static snapshots.

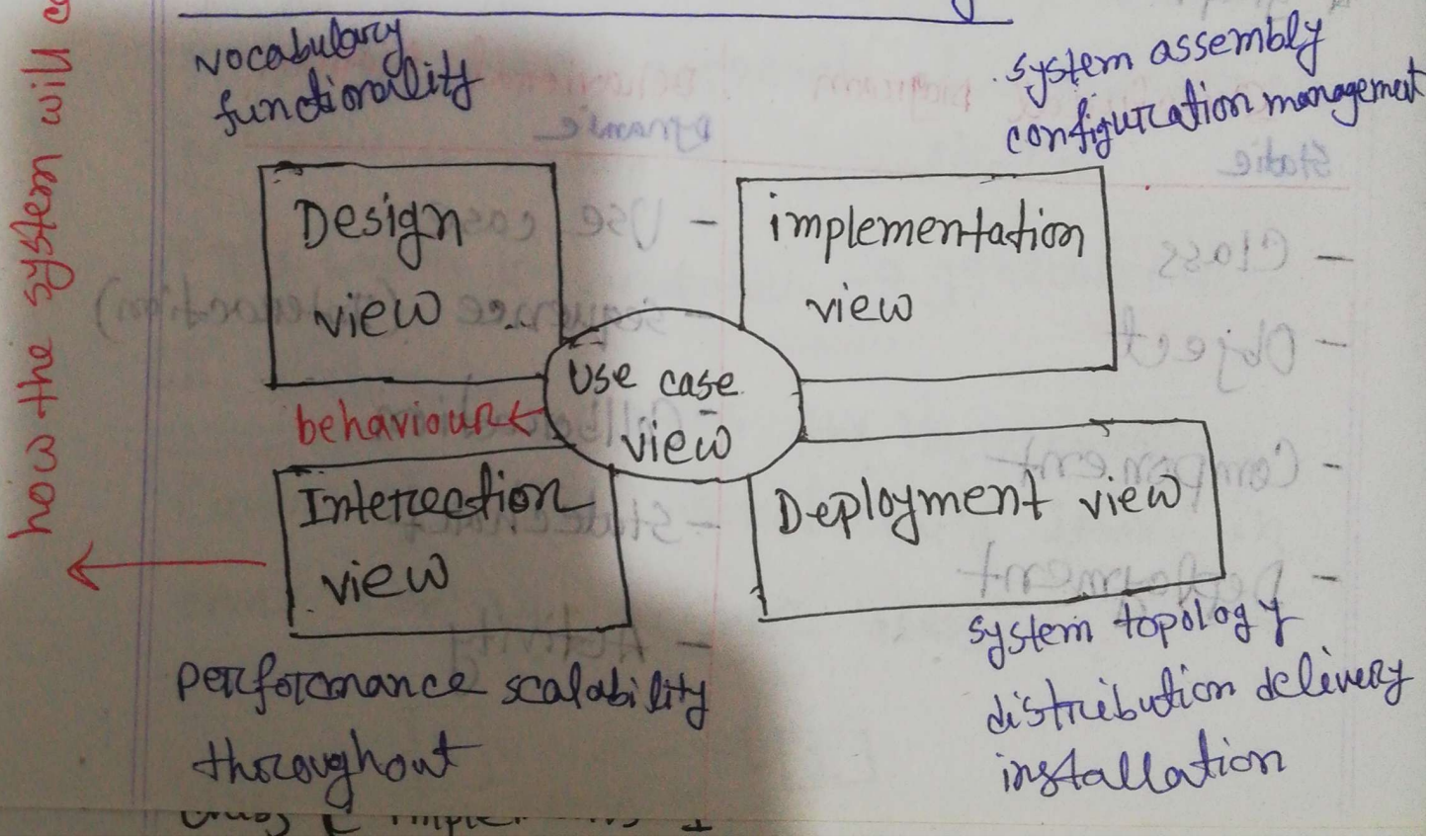
Component diagram: provide static design implementation view.

deployment: (Release)

State: shows dynamic view of an object.

Artifact: physical filesystem.

System Architecture Modeling



SDLC → Software Development Life Cycle
RUP → Rational Unified model

SDLC - RUP

Inception: concept / Idea

business modelling → logic for app? how the system will work.

Elaboration: details of work

Transition: ready to deploy

CH-02

Use case Diagram

A use case specifies the behavior of a system or a part of a system and is a description of set of sequences of actions, including variants, that a system performs to yield an observable result value to an actor.

* represent a functional requirement of your system as a whole.

Book page → 187

* A use case involves the interaction of actors and the system.

* You can apply use case to your whole system. You can also apply use cases to part of your system.

* During requirement analysis we can find the system behaviour from use case diagram.

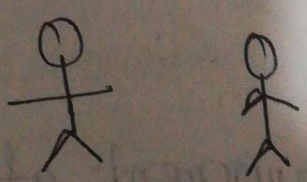
-> communicate করা যায়।

-> use case name verb form এ লিখতে হয়

-> actor system এ যাকে কোন perspective এ ব্যবহার করতে তার verb.

* The name of use cases are always written in the form of verb followed by an object.

Actor

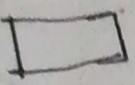


student Teacher

usecase name

You have to provide the name of the actor, under actor symbol.

* Actor can be human or automated system.

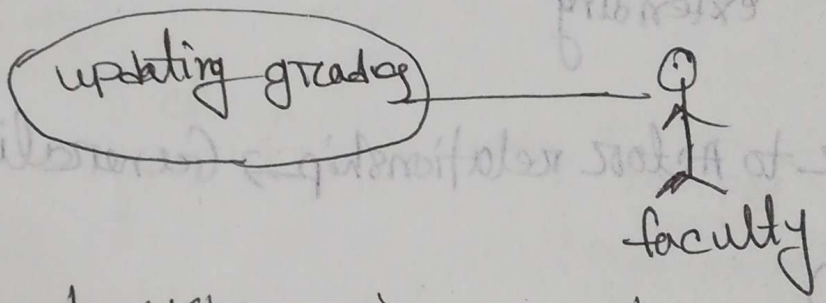


Rectangle box represent a system.

* Actor are not part of the system.
* A system can be an actor of other system.
* A use case describes a what a system does but it does not specify how it does it.

Relationship between system and actor

- Actor can define the environment of a system.



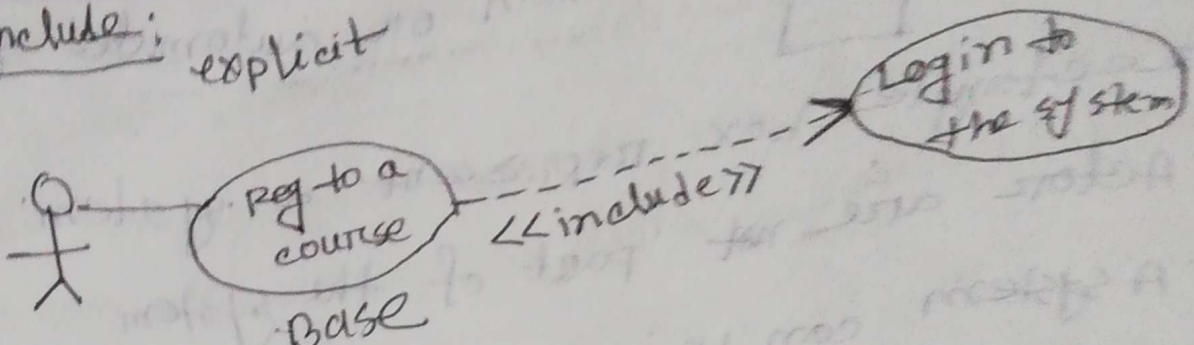
1) Actor to system → Association

2) use case to use case

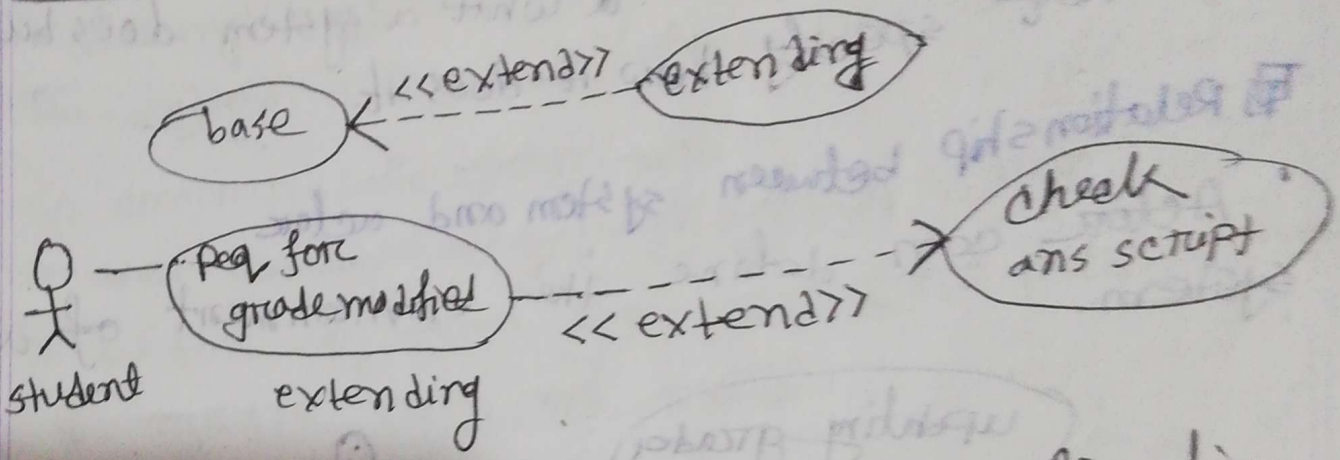
i) include (ii) extend (iii) generalization

They for under dependency

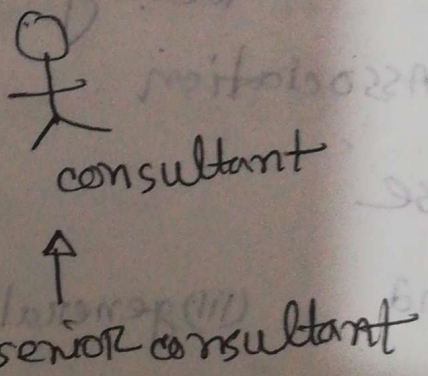
Include: explicit



Extend: optional system behaviour

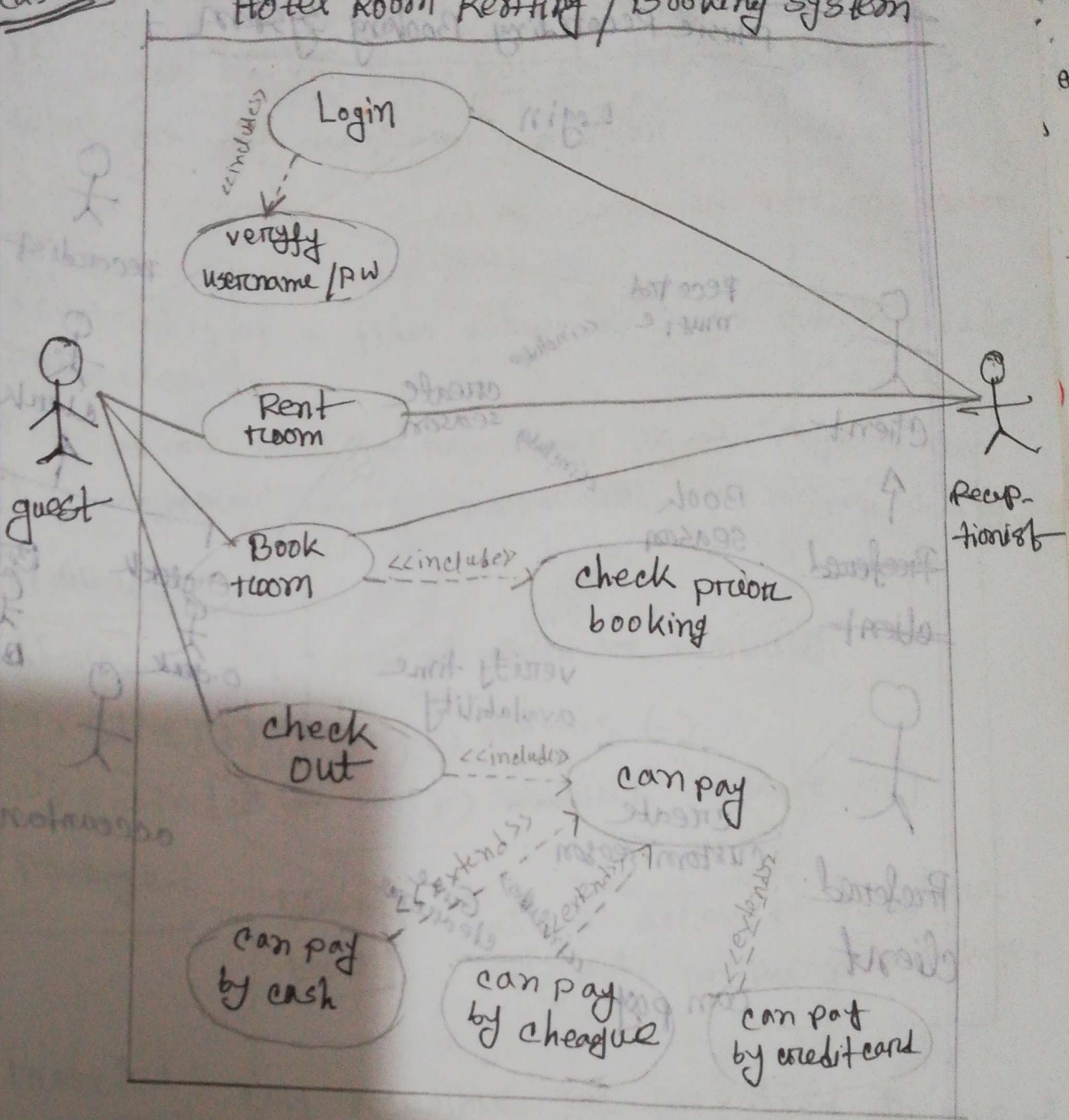


3) Actor to Actor relationship → Generalization



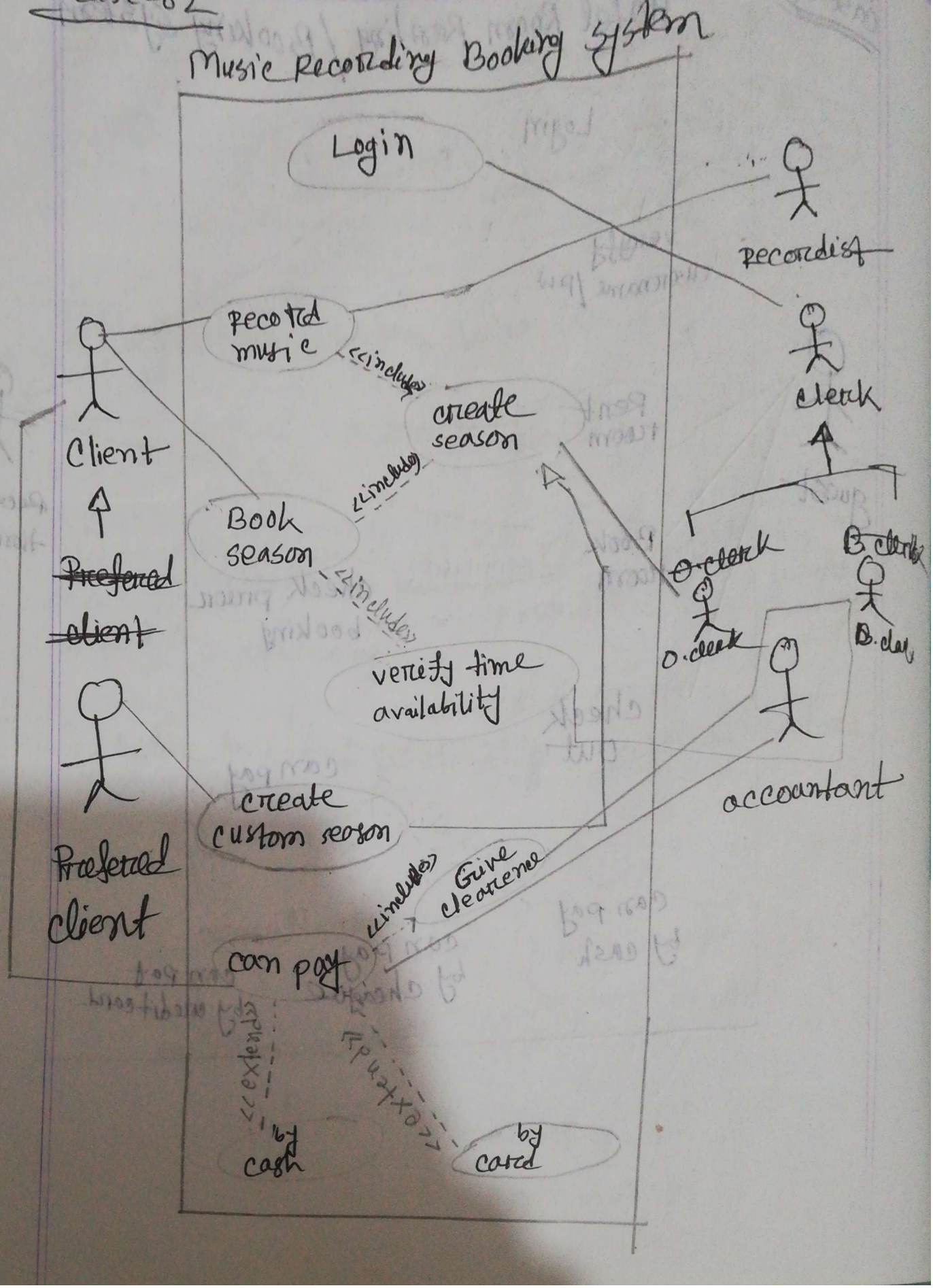
case 01

Hotel Room Renting / Booking System



case-02

Music Recording Booking System



Ref → Stack
Obj → heap

Class Diagram

If I ask the obj what an obj has → attribute
what an obj does → operation

A class diagram shows a set of classes, interfaces and collaborations and their relationships

* Properties of a class attribute can be public, private, protectes.

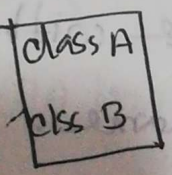
* Classes are used to represent object. Objects can be anything having properties and responsibility.

Attribute

There are 4 types of attribute visibility.

i) public (+)
ii) private (-)
iii) protected (#)
iv) ~~visibility~~ package (~)

Package:



allow access to ~~to~~ other objects in the same package.

Protected: only accesible for the child. In case of generalizations (inheritance), subclass must have access to the attributes and operations of the super-class or they cannot be inherited.

most powerful

private

```
class student
{
    private float cgpa
};
```

Public

Protected visibility is related to inheritance or generalization.

Format

visibility / attribute name : data type = default value {constraints}

visibility — mandatory
attribute name — mandatory (external view)
data type — mandatory
default value — optional
constraints — optional (Limitation)

derived attribute : age, cgpa

attribute: must be unique within the class.
must be a meaningful name.

default value: optional.

constraints:

static: class member. [সব obj তে] common attribute share করে।

Operation

mandatory mandatory mandatory mandatory optional

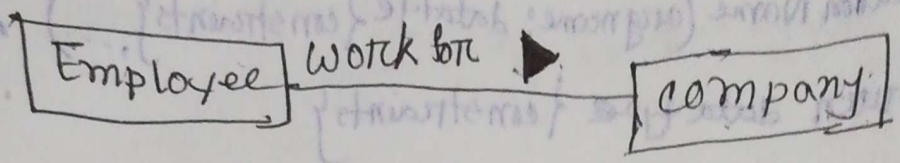
visibility operationName (argname: datatype {constraints}, ...) :

mandatory return data type {constraints}

operation name → camel Case

Student
- studentID : string { Generated by system} - studentName : String - /cgpa : float = 0.0 - dateOfBirth : DateTime = DateTime.now - /age : int = DT.now - dateOfBirth - department : String
+SetStudentName (studentName : string) : void +calculate GPA (courseGrade[] : string, creditTaken : int) : float +showAge ( ) : void

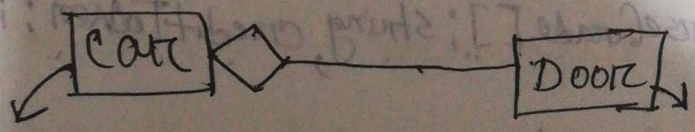
Association: দুটা class এর মধ্যে general relationship.



Multiplicity (ER Diagram cardinality)

Aggregation

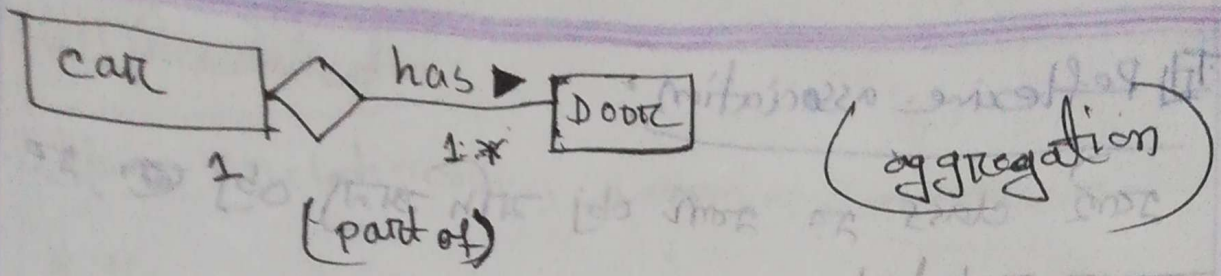
* association provide (has-a) relationship.
* Models a "is a part of" "has-a" relationship.
* special form of association that models a whole-part relationship between an aggregate (the whole) and its parts.



whole

part/participating obj

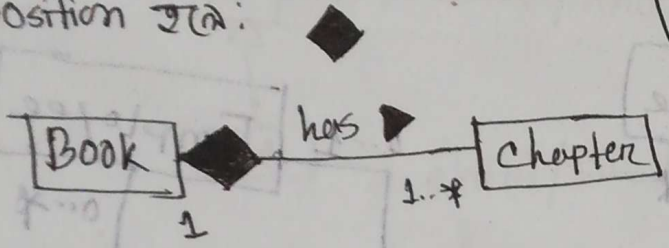
* part যদি whole এর উপর lifetime depend না করে তাহলে এটা aggregation / depend করলে composition



Composition (consist/contain/made of)

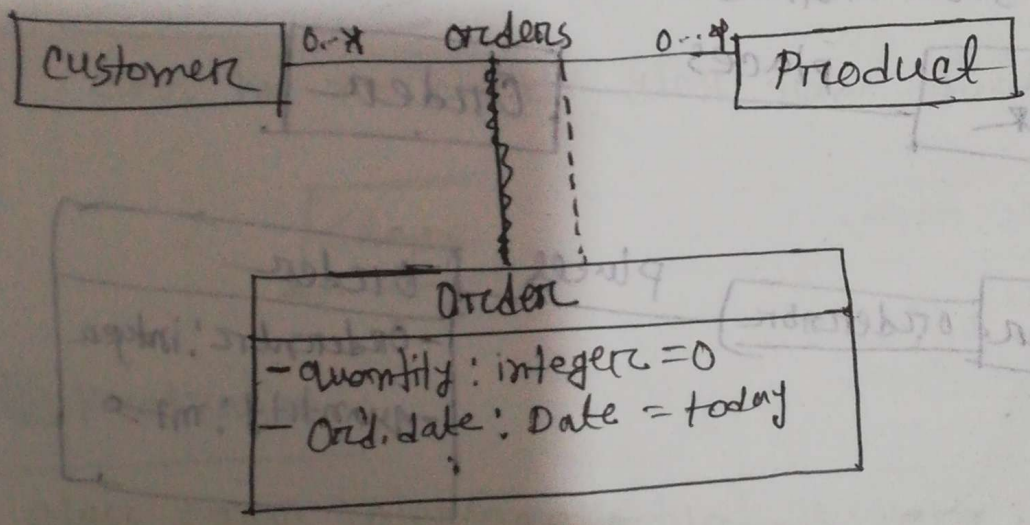
composition হচ্ছে:

whole যে side এ থাকে



whole side cannot be zero in composition.

Association class

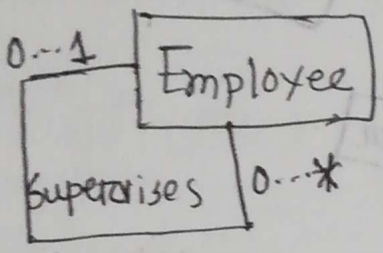


Association class encapsulate information about relationship.

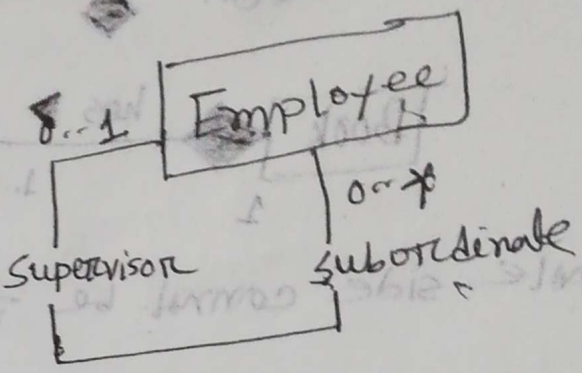
Reflexive association:

একই class এর একটি obj যদি অন্য obj এর সাথে related.

using association name:

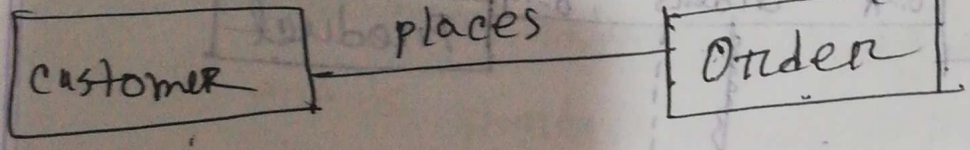


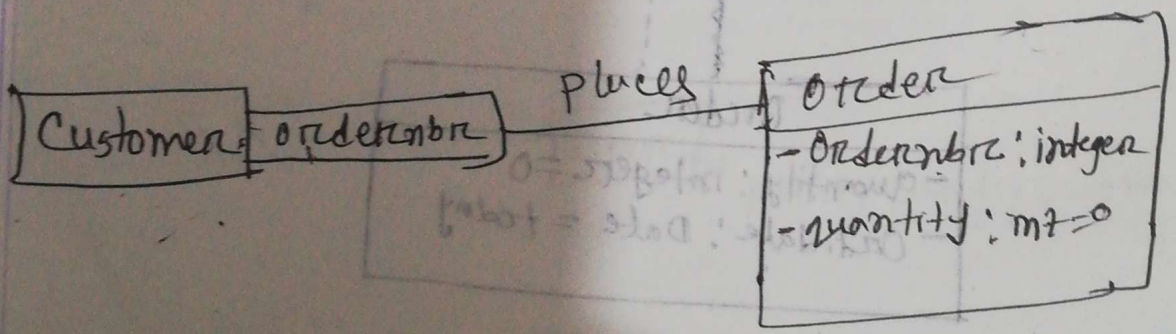
using role name:



Qualified Association

unique from another class (Foreign key এর মত)

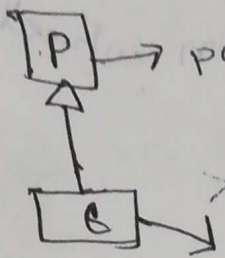




Generalization

(is-a) relationship, bottom up ~~app~~ process

* "is kind of" or "is type of" relationship



P → parent / Ancestor / superclass, base class

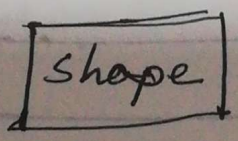
C → child / subclass / descendant / Leaf / Derived

↓

can inherit ~~Attributes~~, operations, relationships

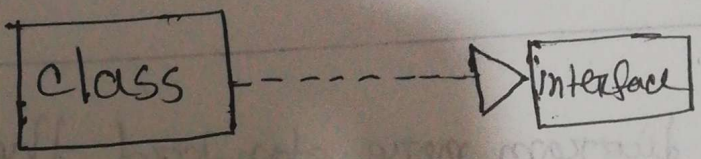
Abstract class

class has no object. class name should be italic



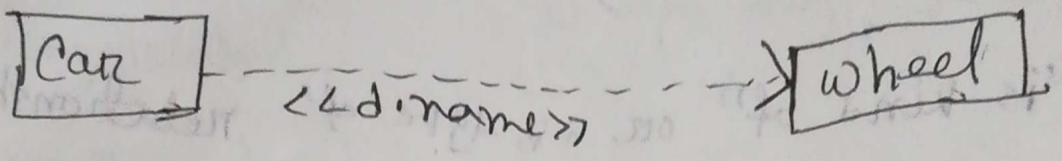
Realization

class থেকে interface implement করাকে indirect inheritance.



Father of class diagram

Dependency



car depends on wheel

CRC CARD

CRC → Class Responsibility Collaboration.

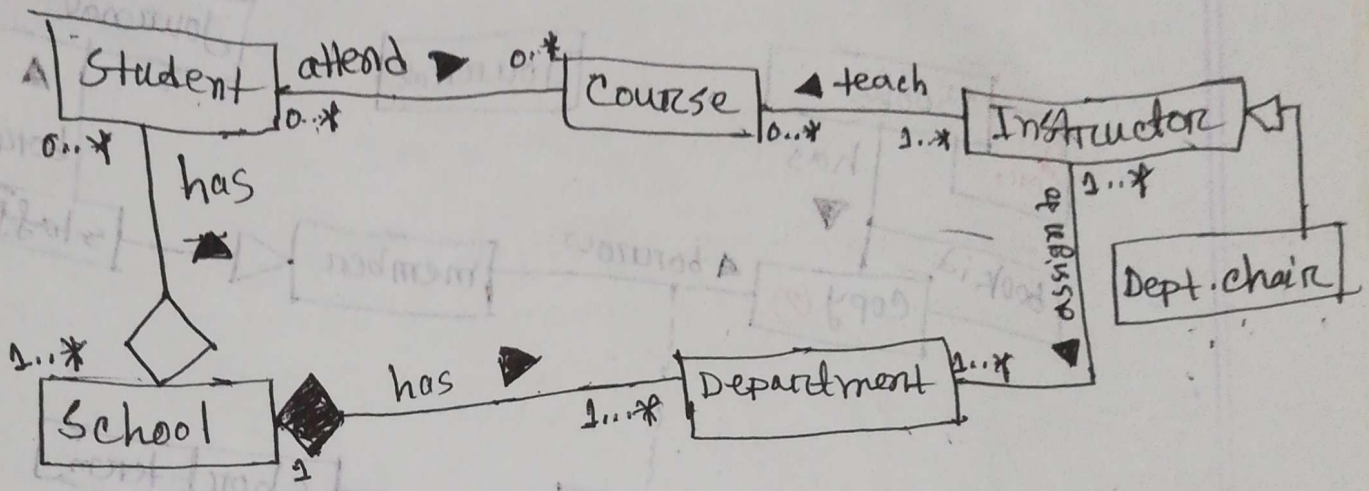
card size: 3x5 / 4x6 inchs

goals: provide the simplest possible conceptual introduction to OO. design

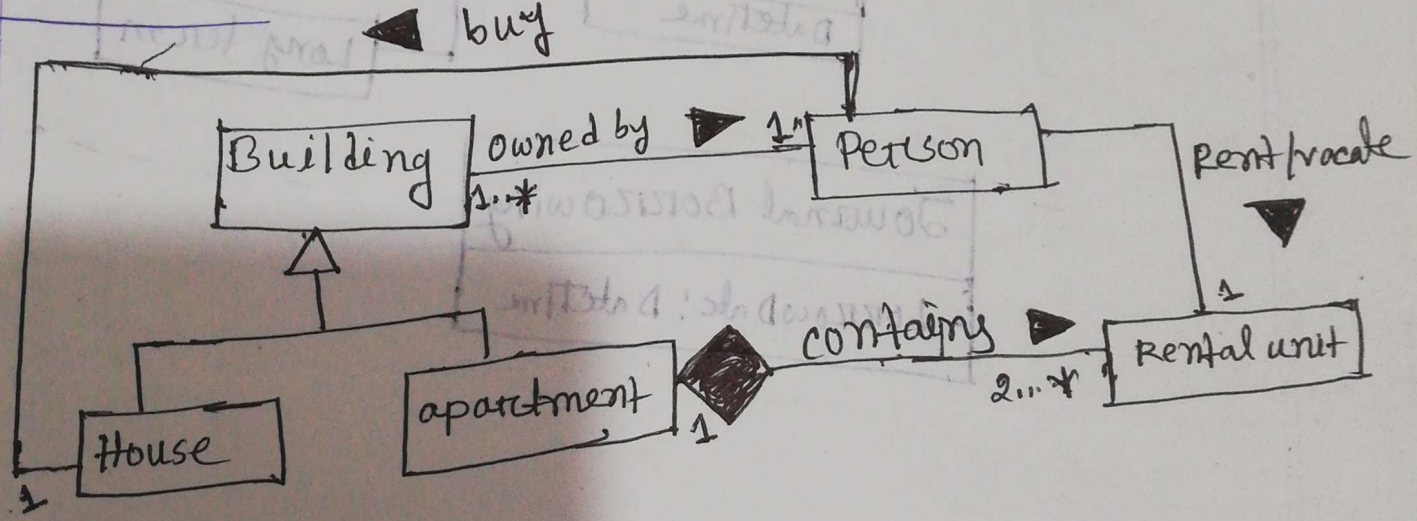
Class name:	
Subclasses:	
Superclasses:	
Responsibilities	Collaborators

*Class diagram more standard than CRC card

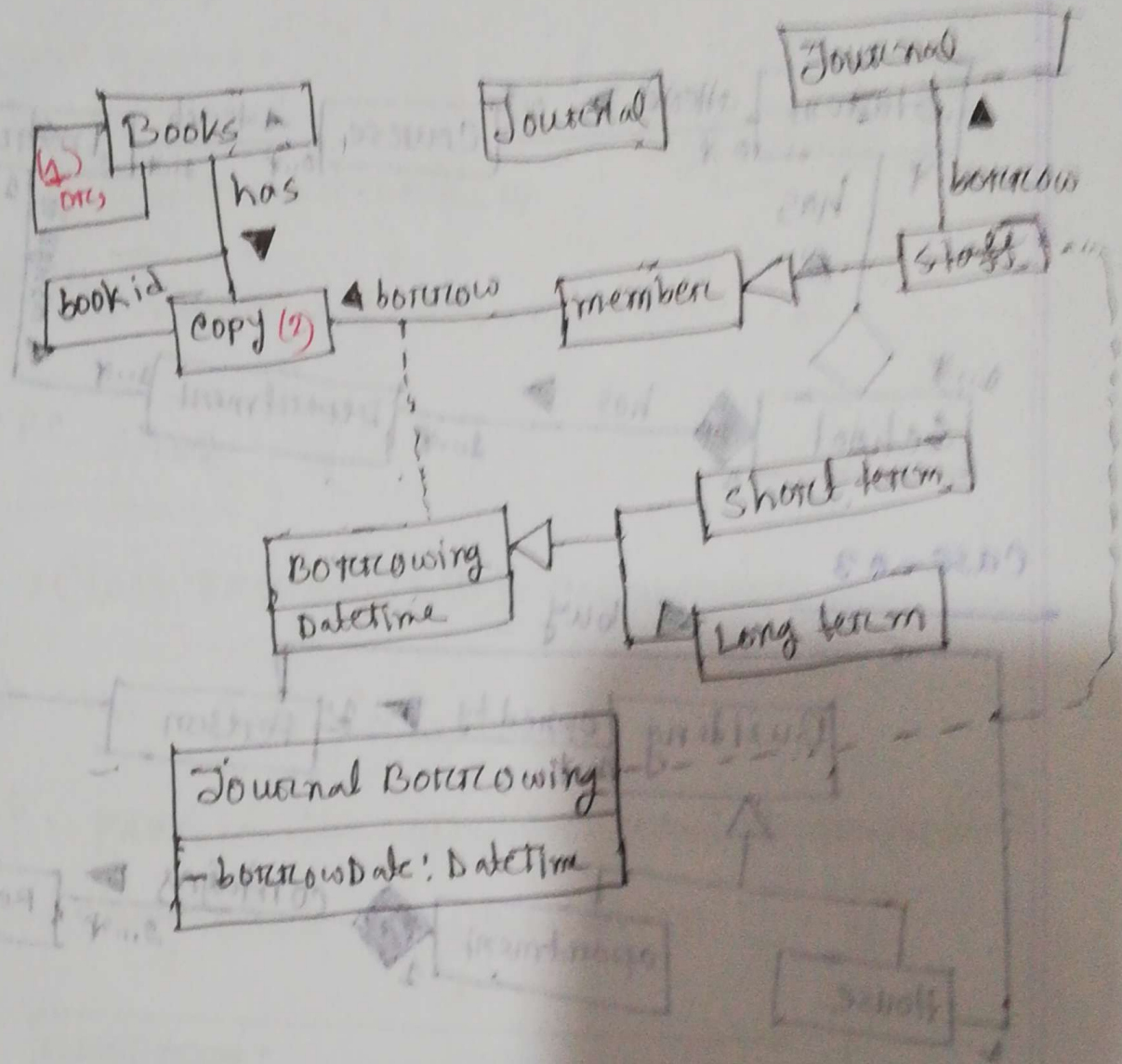
Case-2



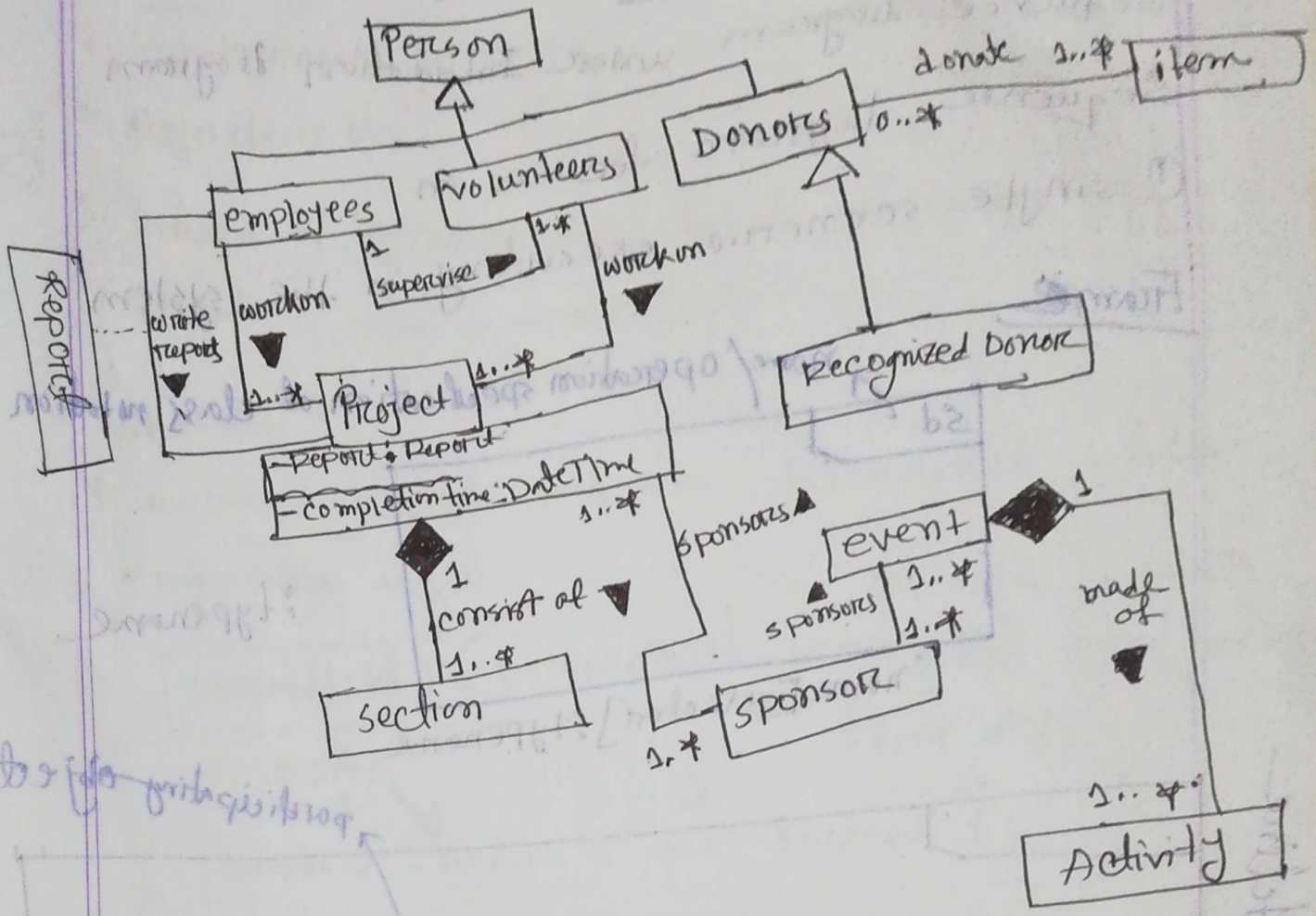
Case-03



case – 5


Books
(1)
has
book id
copy (1)
borrow
member
Journal
Journal
borrows
staff
short term
Borrowing
Datetime
Long term
Journal Borrowing
-borrowDate: Datetim

case-6


Person
Donors
donate
1..*
item
0..*
employees
volunteers
1
supervise
1..*
workon
workon
write reports
Reports
1..*
Project
1..*
Recognized Donor
-Report : Report
-completion time : DateTime
1..*
1
consist of
1..*
sponsors
event
1
1..*
1..*
sponsors
made of
section
sponsor
1..*
1..*
Activity

FINAL TERM

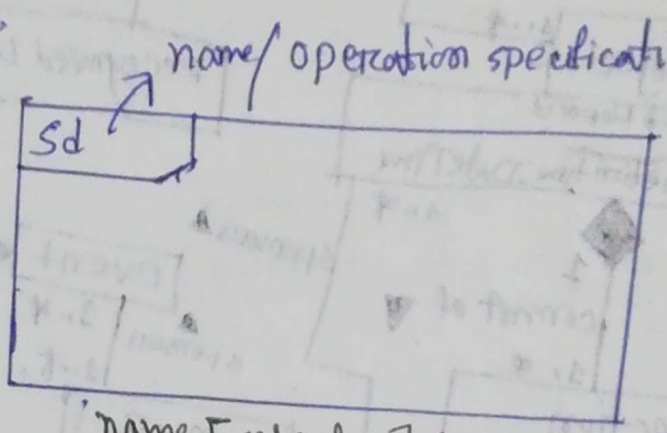
Sequence Diagram / Event Diagram

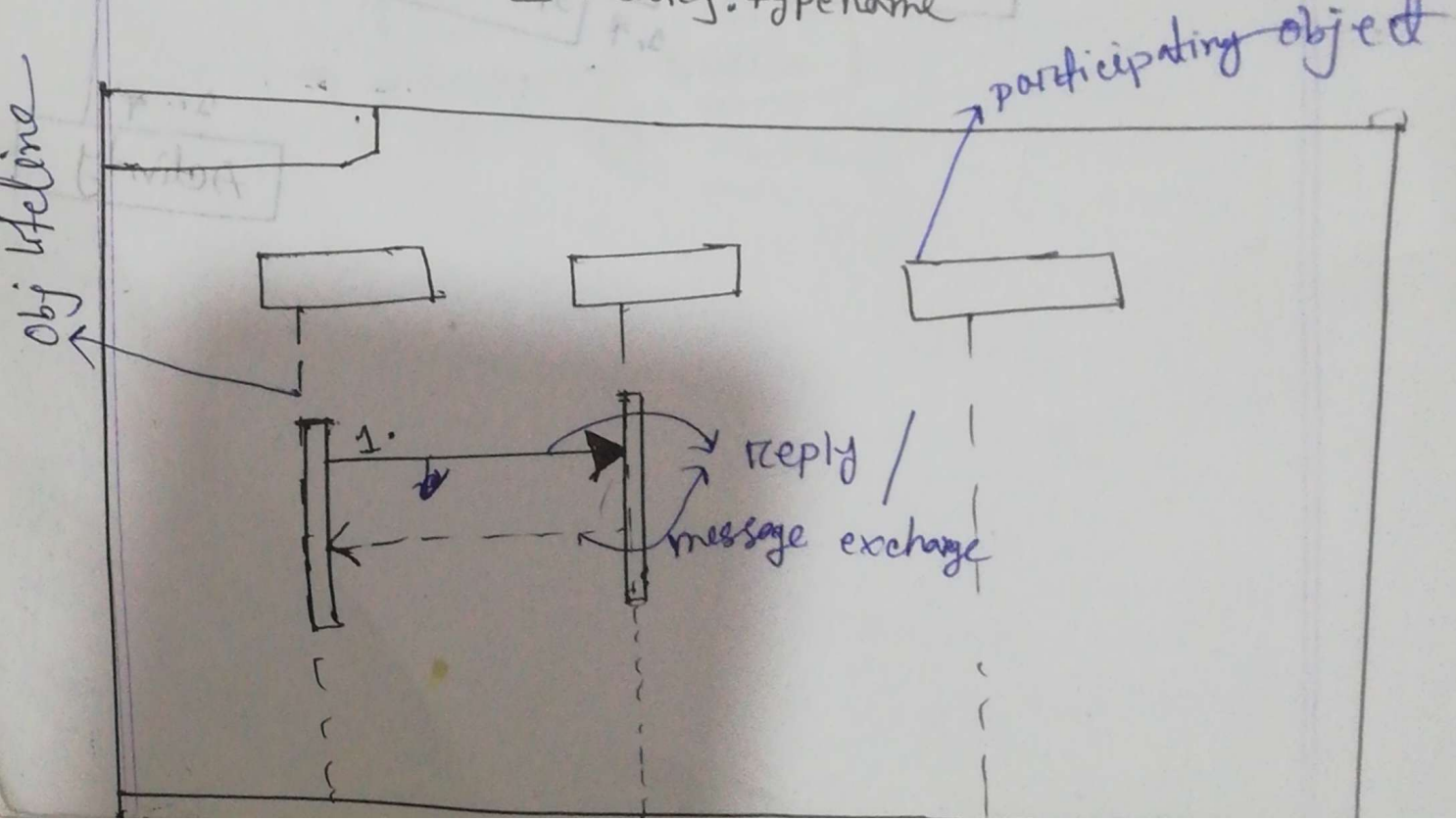
Sequence diagram under Interaction diagram

Sequence diagram focuses on

① single scenario executing in the system

Frame



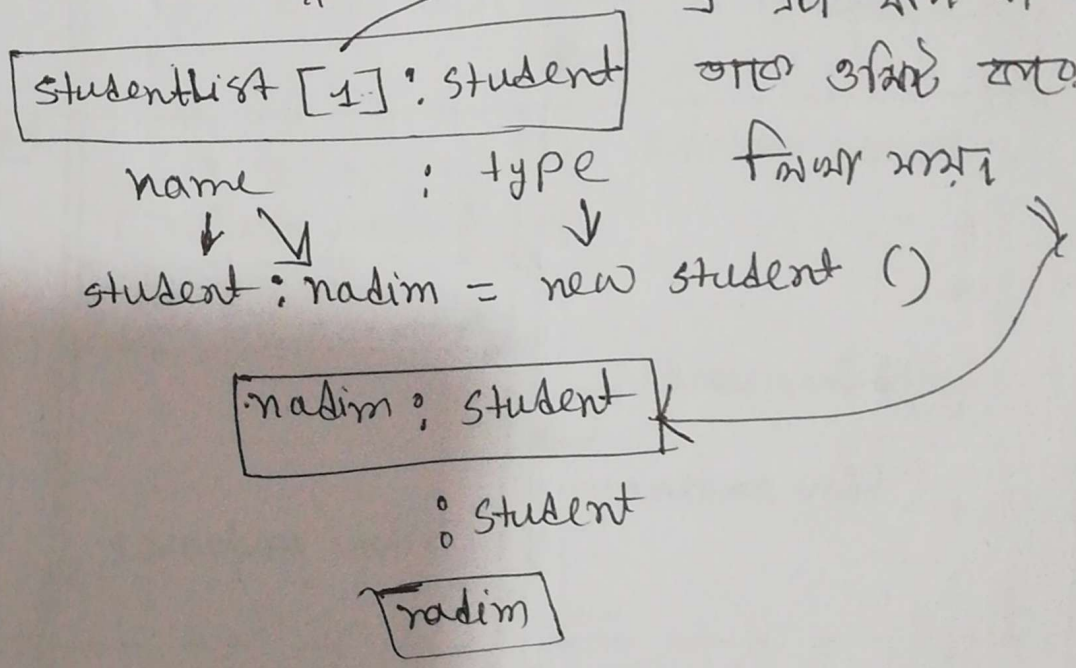


* There are 3 types of message in sequence diagram.

1. Synchronous message → (wait)
2. Asynchronous " → (ex: text message) (no wait)
3. Return of synchronous / object or instance creation - - - →

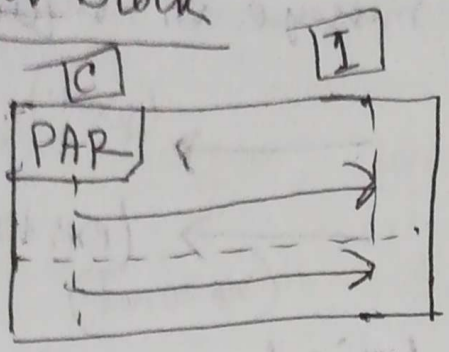
~~student~~

* destroyed object has a truncated lifeline ~~end line~~ "X"

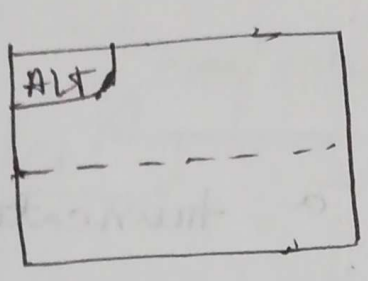


* participating obj identify করার best way হচ্ছে তার সাথে interaction করছে, actively or pasively.

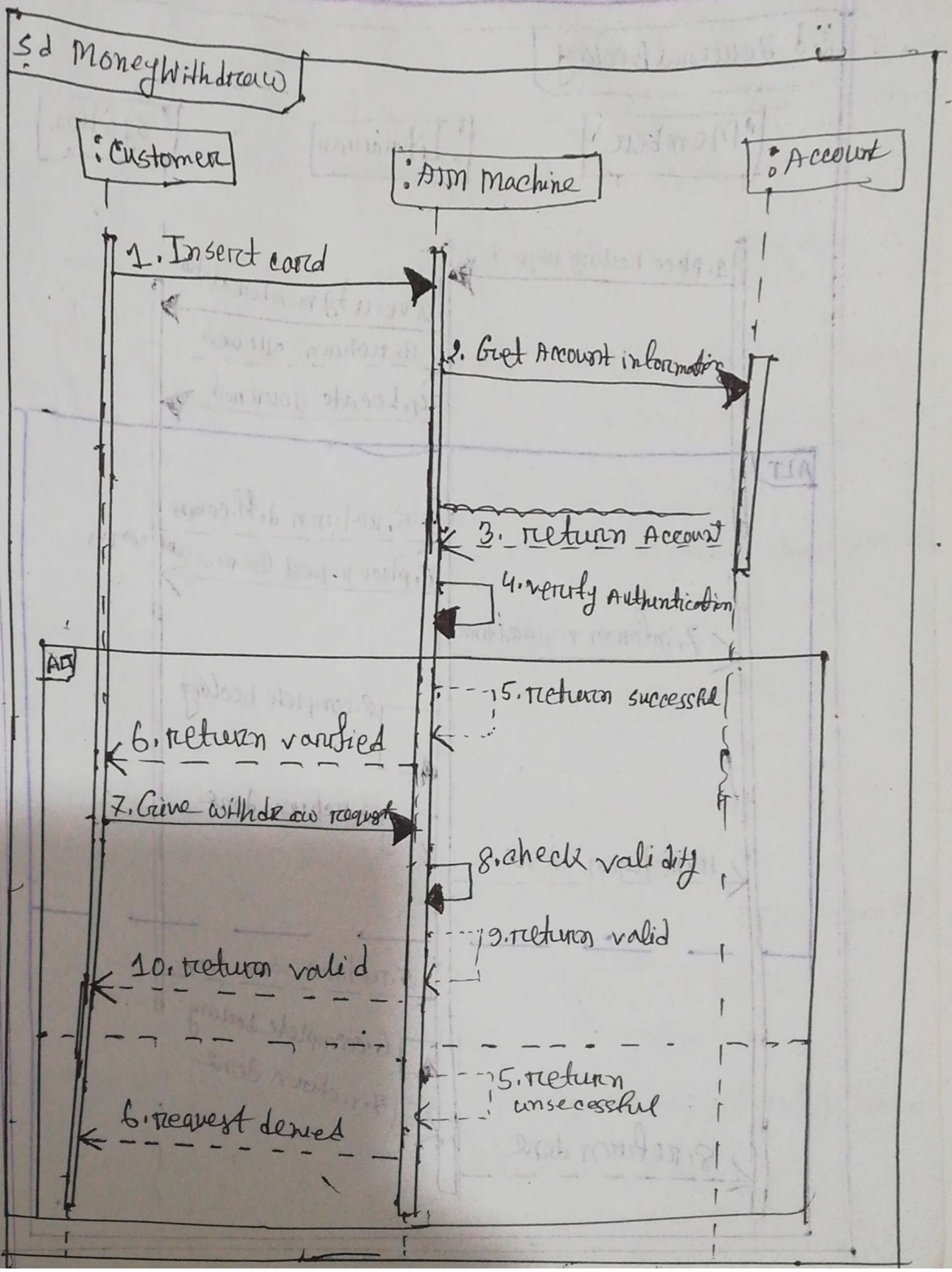
Parallel Block



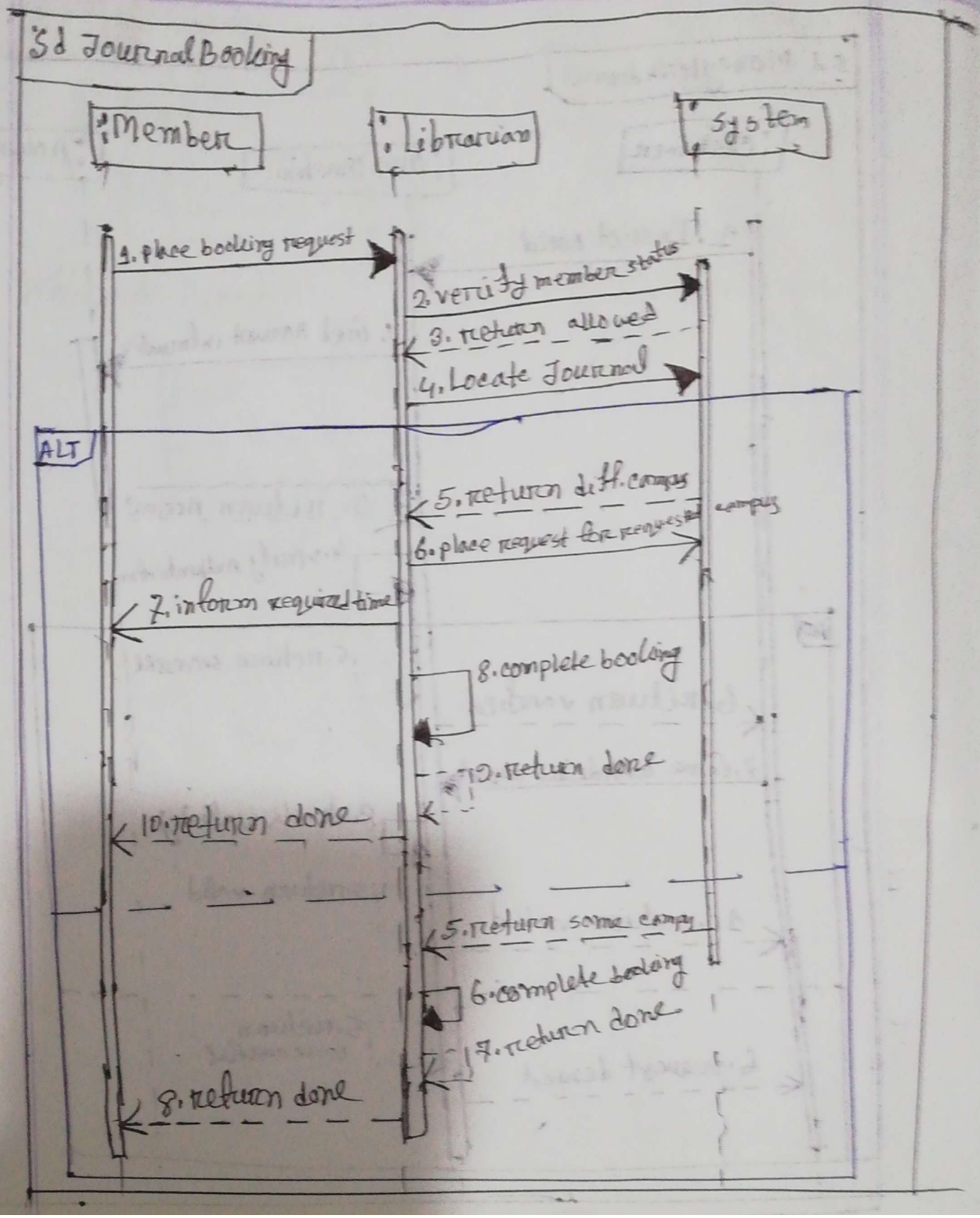
Alternate Block



case-01


sd MoneyWithdraw
:Customer
:ATM Machine
:Account
1. Insert card
2. Get Account information
3. return Account
4. verify Authentication
Alt
5. return successful
6. return varified
7. Give withdraw request
8. check validity
9. return valid
10. return valid
5. return unsecessful
6. request denied

Case-02


Sd Journal Booking
:Member
:Librarian
:System
1. place booking request
2. verify member status
3. return allowed
4. Locate Journal
ALT
5. return diff. campus
6. place request for requested campus
7. inform required time
8. complete booking
9. return done
10. return done
5. return same campus
6. complete booking
7. return done
8. return done

Book page-217

Activity Diagram

An activity diagram is essentially a flow chart, showing flow of control from activity to activity.

* activity diagram is use to model the dynamic aspects of systems.

* Activity diagram: flow of control from activity to activity
* interaction " : " " " " obj to object.

* Activity is non atomic (collection of action)
* Action is atomic [result] → cant be broken into parts.

● → initial node

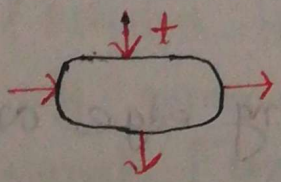
◉ → final node

▭ → Action node

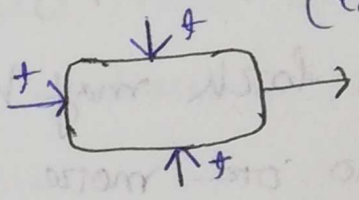
↓ → Activity edge

* Execution is model by token [token → conceptual]

* token, action node produce করে।



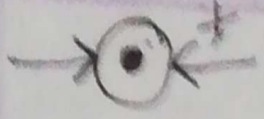
(Execution হচ্ছে না)



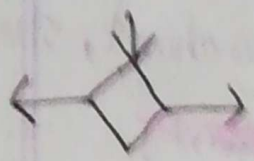
(execution হচ্ছে)

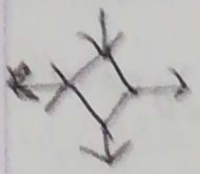
for action node all of the incoming edge should have token in order to begin the execution

* Graphically, an activity diagram is a collection of vertices and arcs.


(activity close)


(activity close)

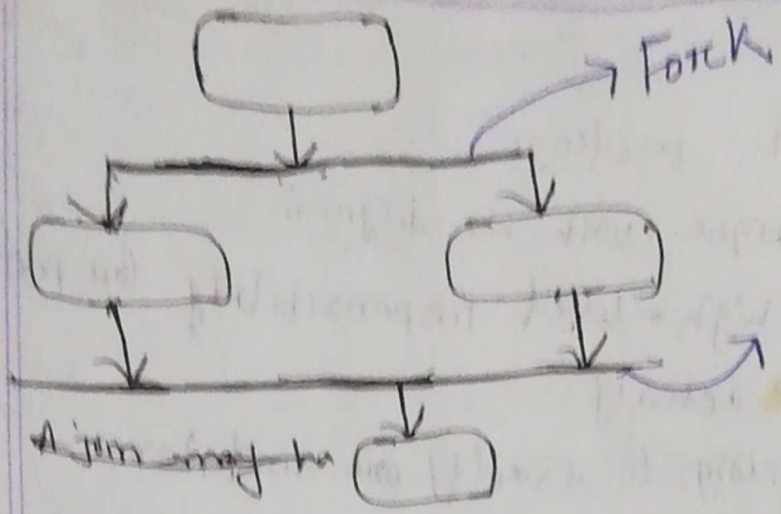

(Decision node)


(incoming, outgoing ka koi combination ho sakta hai)

* If a token come any of the incoming edge, then it will pass the token at all its outgoing edge.
* action node ka dead lock ka solution merge node.

Forking and Joining

* Fork ka kaam hota divide karna
* Parallel action ko alag alag system ko, uske fork ke join node aate.
* A fork may have one incoming edge and two or more outgoing edge, each of which represents an independent flow of control.



A join may have two or more incoming edge and one outgoing edge.

* ~~any~~ incoming edge ও token থাকলে, outgoing edge ও pass হবে।

* join and fork should have balance, meaning that the number of edge that leave a fork should match the number of edge that enter its corresponding joint.

* Activities that are in parallel flows of control may communicate with one another by sending signals. [coroutine]

Object nodes

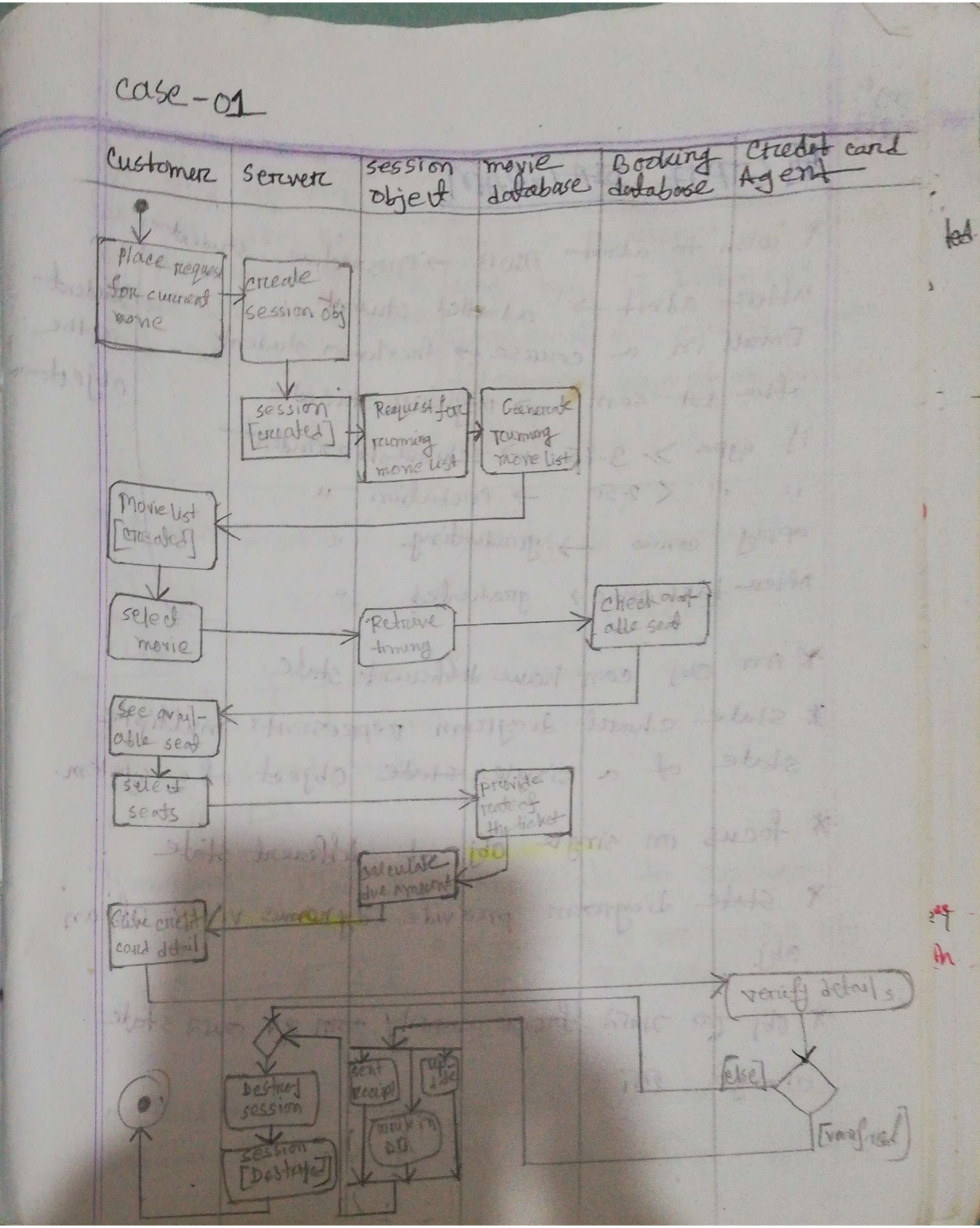
[ ]

Swimlane

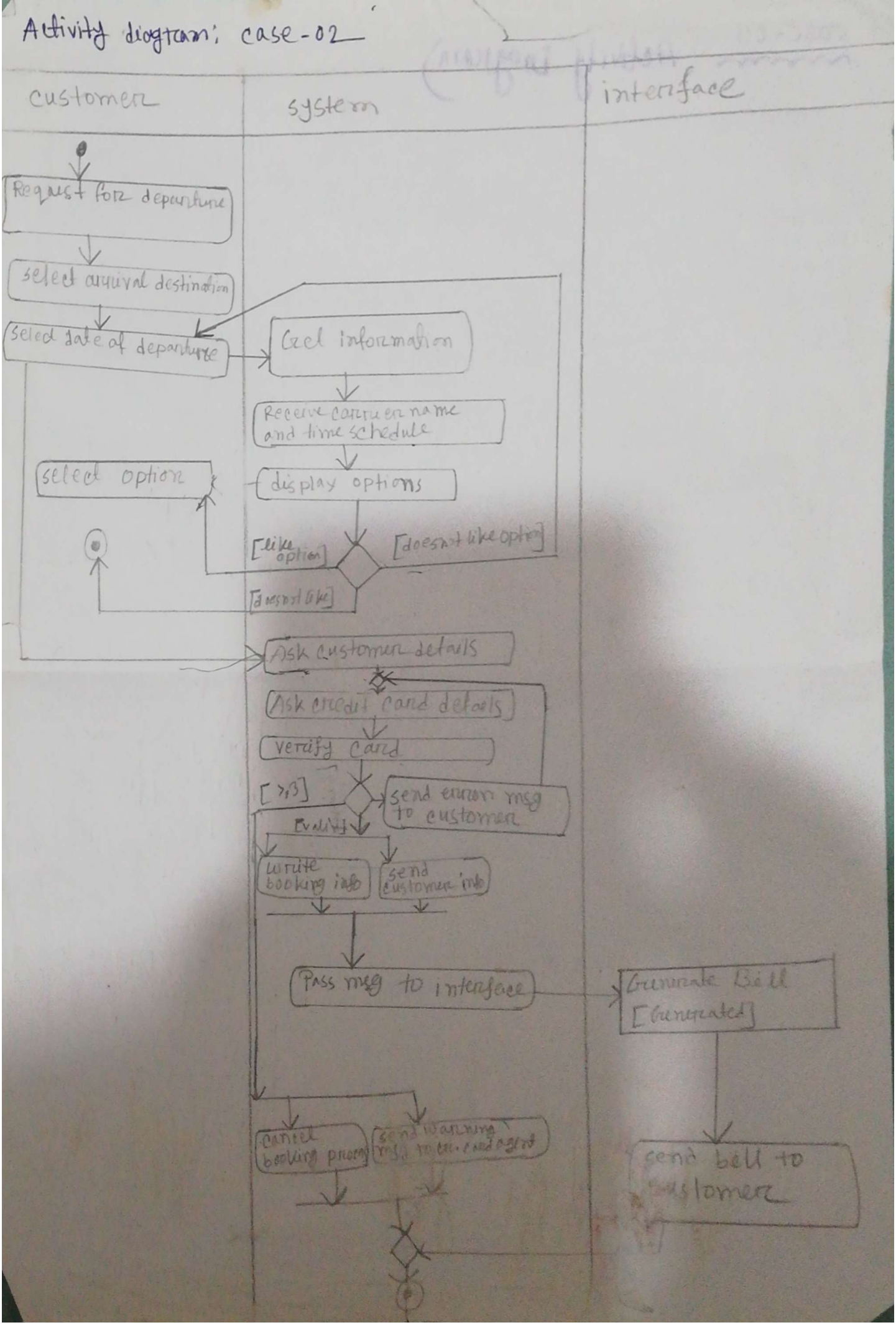
* is a kind of package
* has a name unique with its diagram
* represent a high-level responsibility or part of the overall activity
* every activity belongs to exactly one swimlane

* activities and action → verb phrases
* object node ⇒ noun phrases

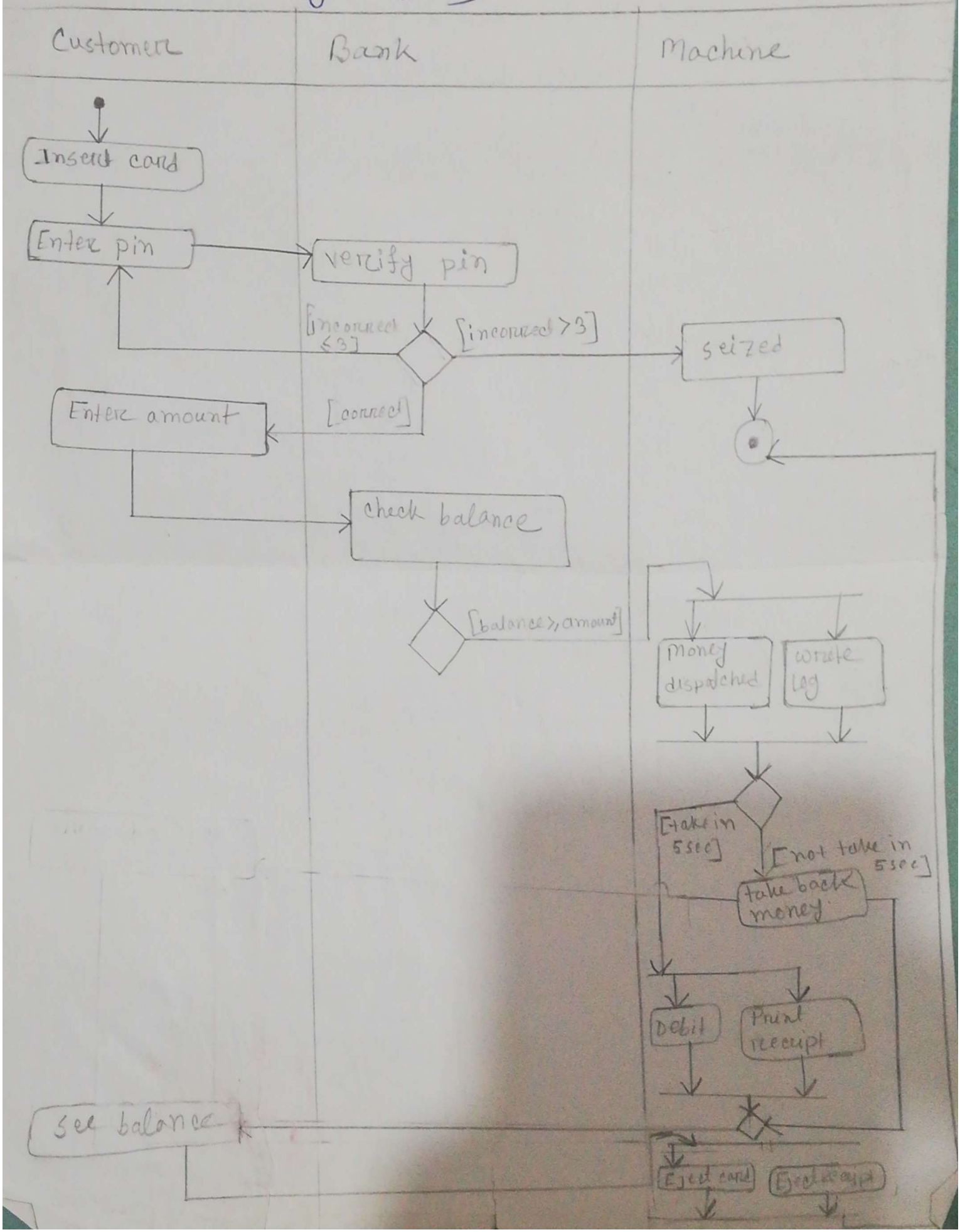
case-01


Customer
Server
session object
movie database
Booking database
Credit card Agent
Place request for current movie
create session obj
session [created]
Request for running movie list
Generate running movie list
Movie list [created]
select movie
Retrive timing
check avail able seat
See avail-able seat
select seats
provide price of the ticket
calculate due amount
Give credit card detail
verify details
[else]
[verified]
destroy session
session [Destroyed]
Rent receipt
update DB

Activity diagram: case-02



case-03 (Activity Diagram)



STATE DIAGRAM

* wish to admit ATUB → Prospective student
after admit → admitted student
Enroll in a course → freshman student
after 1st sem → regular student
if cgpa ≥ 3.75 → scholarship student
" " < 2.50 → Probation "
apply convo → graduating "
After take convo → graduated "

Student is the object

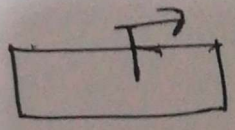
* An obj can have different state.
* state chart diagram represents multiple state of a single ~~state~~ object of a system.
* focus on single obj of different state
* state diagram provide dynamic view of an obj.
* obj কে অন্য force apply করা হলে অন্য state change হয়।

Activity diagram	State diagram
used to model how different areas of work behave with each other and interact.	used to represent a single obj and how its behaviour causes it to change state. [dynamic view of an object]

* A state diagram is composed by 3 components

(1) State (2) transitions (3) events

State

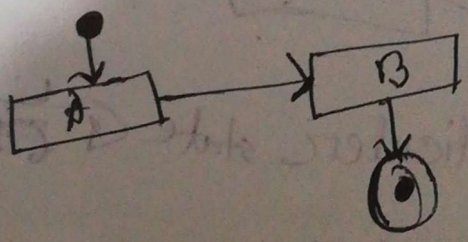


state name [noun or verb form]

(state)

- start state
- ◉ End "

Transitions: Flow from one state to another.



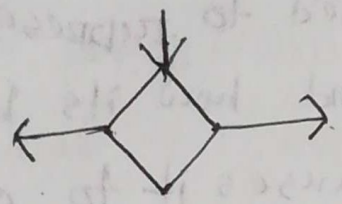
A থেকে B তে যাওয়ার জন্য obj trigger করতে হয়।

→ transition

archived → data store but not in active state.

software ও delete করা যায় কিন্তু যায়, archive হয়ে যায়।

Decission Point



Decission points make the diagram more visually appealing by grouping transitions.

Activity diagram

fork, join

state chart

synchronization

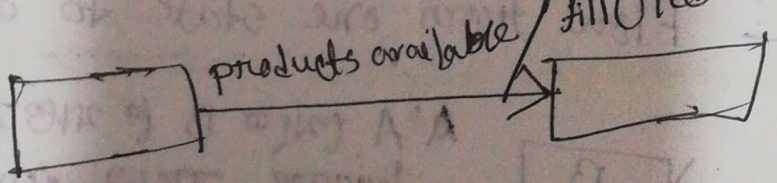
Event

Event create হবে, কিছু ~~to~~ action ঘটবে পরবর্তী।

Event / Action

Event : Reason of transition.

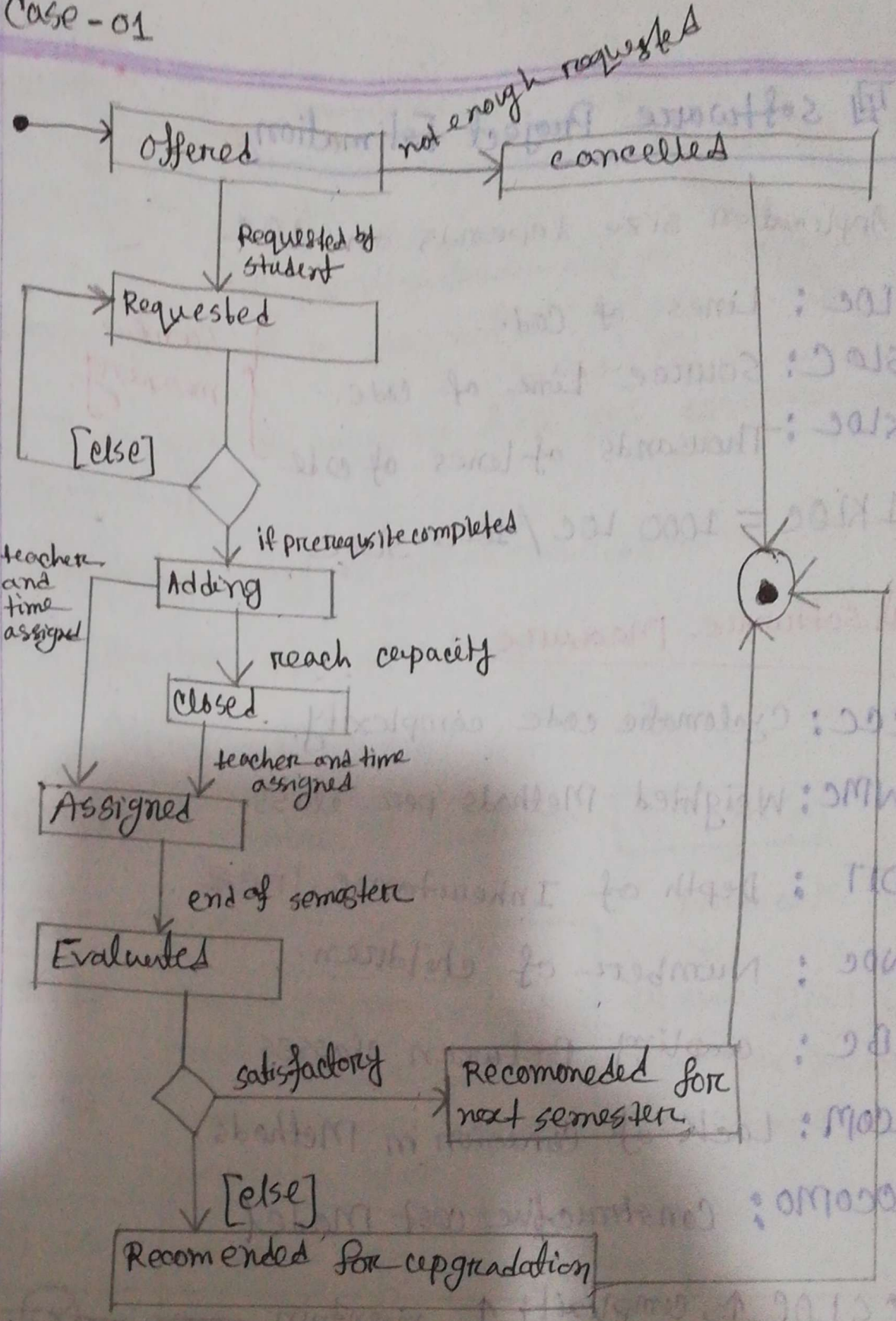
Action : Result of an event.



~~X~~ কোনো obj যদি কোন particuler state এ থাকে, সেই state এর যে action করবে।

- entry
- do
- exit

Case - 01


Offered
not enough requested
cancelled
Requested by student
Requested
[else]
if prerequisite completed
teacher and time assigned
Adding
reach capacity
Closed
teacher and time assigned
Assigned
end of semester
Evaluated
satisfactory
Recomended for next semester
[else]
Recomended for cepgradation

Software Project Estimation

Application size depends on LOC

LOC : Lines of Code.
SLOC : Source Line of code.
KLOC : Thousands of lines of code

} same meaning

1 KLOC = 1000 LOC / 1000 SLOC

Software Measure

CCC : Cyclomatic code complexity.

WMC : Weighted Methods per class.

DIT : Depth of Inheritance Tree

NOC : Number of children.

CBC : Coupling Between classes.

LCOM : Lack of Cohesion in Methods

COCOMO : Constructive cost Model

* SLOC ↑, complexity ↑, maintain করাও কঠিন।
* Cyclomatic number above 10 → considered to be very complex.

Cyclomatic code complexity

Node → statement
execution → edge

CCC is a software metrix used to determine the complexity of a program.

* There are 4 ways to find out cyclomatic code complexity

1) Finding linearly independent path (McCabe Formula)
2) Number of area/region
3) $V(G) = E - N + 2$
4) Total number of predicate $C = P + 1$

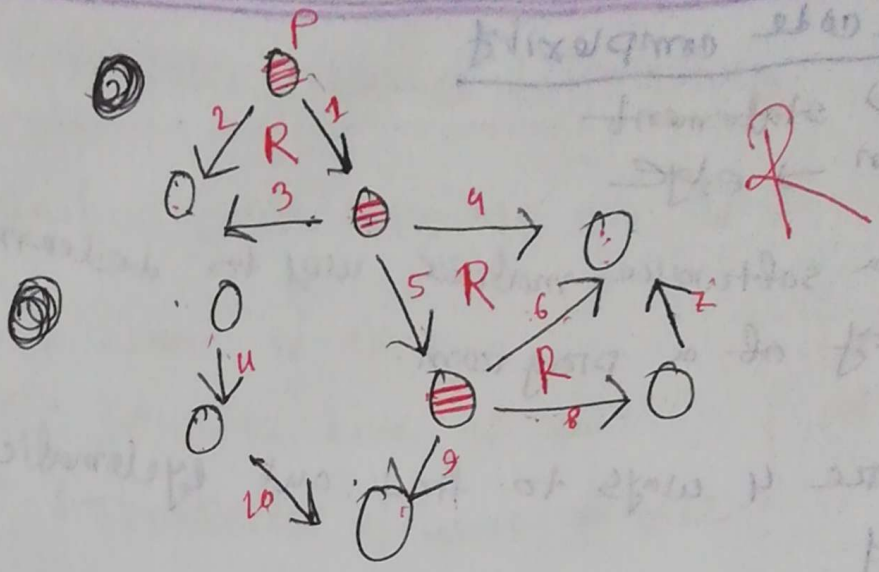
predicate node

যে node থেকে একাধিক branch বের হয়

(conditional node)

Threshold values

CC	Risk evaluation
1-10	Simple program, without much risk.
11-20	More complex program, moderate risk.
21-50	complex program, high risk.
>50	Un-testable program, very high risk



① $V(G) = E - N + 2$

$= 11 - 9 + 2 = 4$

② Number of region $= 4$

③ $C = P + 1$

$= 3 + 1 = 4$

CCC → Cyclomatic method বলে থাকে,

WMC → Total number of class.

Let, A be a class.

$m_1, m_2, m_3, \ldots\ldots m_n$

$WMC(A) = C(m_1) + C(m_2) + C(m_3) + \ldots\ldots C(m_n)$

$= \sum_{i=1}^{i=n} C(M_i)$

$= \sum_{i=1}^{i=1} c_i$

WMC ↓, Reusability ↑
- n ↑, potentially error prone
- impact on the children of the class
- Reusability ↓ [number of complex test method in class]

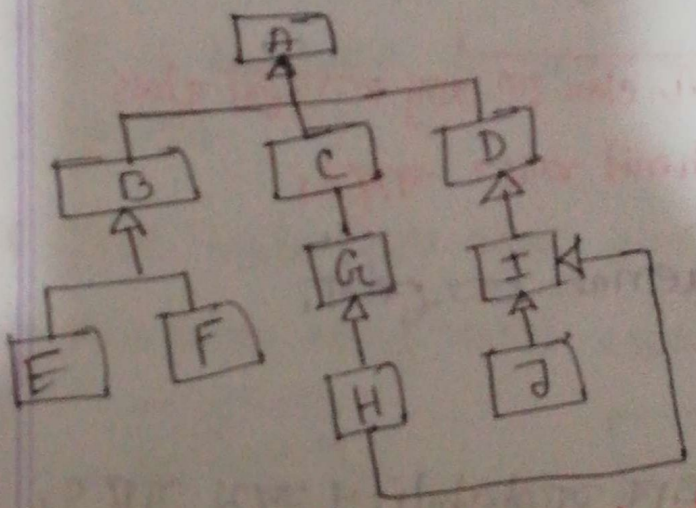
*fault-proneness → error প্রবণতা নির্ণয় করতে ব্যবহৃত।

Depth of Inheritance Tree (DIT):

* class এর different প্রকার নির্ণয় করতে ব্যবহার করা, reuse করার potentiality জানা যায়।

If DIT metric value higher, reuse potentiality is also higher.

* class কত deep তা নির্ণয় করে unique behaviour predict করা যায়।



DIT (D) = 1

DIT (G) = 0

DIT (H) = 3 } Highest length.

DIT (H) = 1 ✗

In case of multiple inheritance, DIT is calculated based on the maximum length.

DIT ↑, Functionality ↑ (Maintainability, portability, Efficiency) ↓

* Threshold of DIT matrix value = 10.

* DIT class এর দৈর্ঘ্য measure করতে সাহায্য করে।

*As DIT grows, lower level classes inherit many methods.

*A deep class hierarchy means greater design complexity

Number of Children (NOC)

NOC (C) = 0

NOC (A) = 3

NOC (B) = 2

একটা class কতগুলো inheritance হচ্ছে সেটা

* DIT represents reuse potentiality

*NOC " degree of reuse

একটা class কে অন্য কতগুলো class inherit করছে সেটা।

influence ↑, class correctly designed

NOC যত বেশি influence তত বেশি।

NOC (A) = 3.

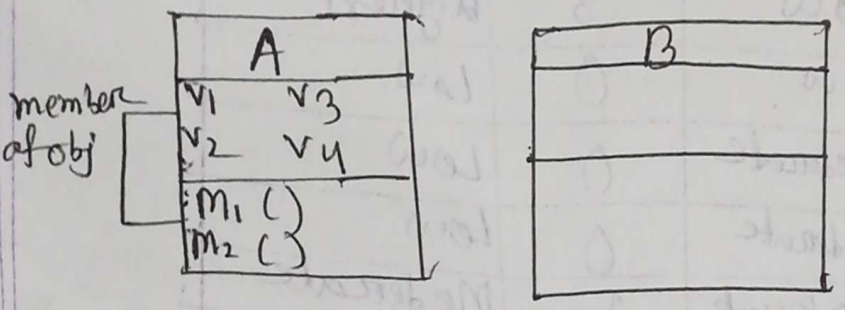
* NOC ↑, detect করার probability কম যায়।

Coupling Between classes (CBC)

Coupling : কোনো class অন্য class এর method/member use করলে।

*our aim, when system design → reduce coupling

inheritance, association কম করা।



1. inheritance
2. Association
3. Interface implementation [indirect form of coupling]

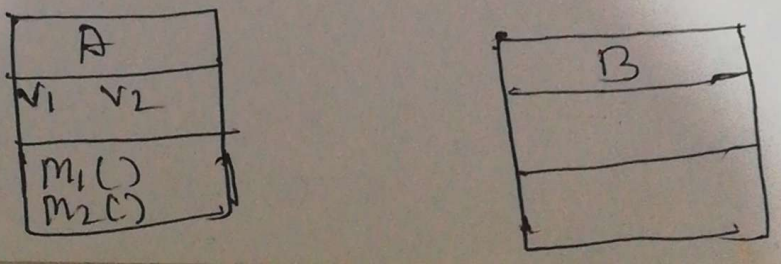
Modularity: একাই যদি সব কাজ করা যায়।

*coupling যত বেড়ে যাবে modularity কমবে।

* It is desirable to reduce the coupling between classes.

* CBC value যত কমবে, class independent তত বাড়বে।

⇒ Association apply করলে CBC value add হবে B class এ।



* user interface এর জন্য যে class গুলো থাকে, তাদের CBC value high থাকে।

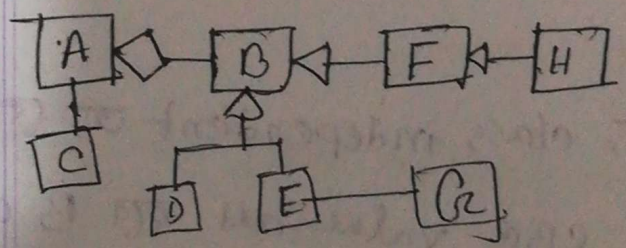
class	DIT	Reusability	NOC	Influence on design
A	0	Low	0	Low
B	0	Low	3	Highest
C	0	Low	0	Low
D	1	Moderate	0	Low
E	1	Moderate	0	Low
F	1	Moderate	1	Moderate
G	0	Low	0	Low
H	2	Highest	0	Low

influence → parent class এর perspective

DIT → child " " "

* যে সবচেয় class inheritance ও participate করে, তাদের CBC বেশি।

Question



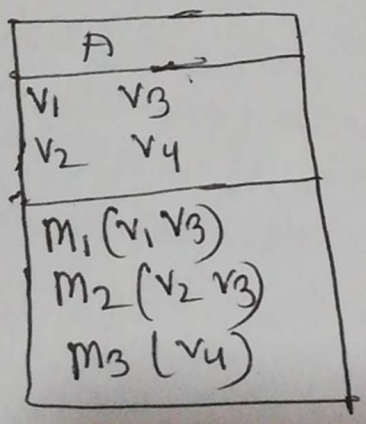
LCOM

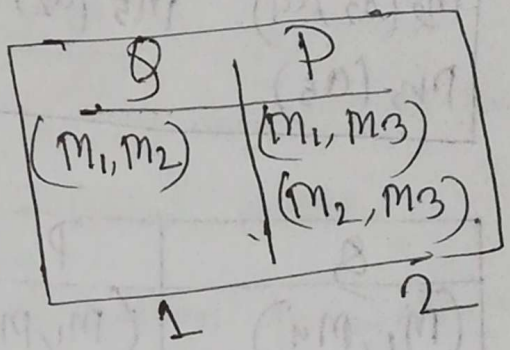
LCOM → Lack of Cohesion in Methods.

Cohesion: কোন class এর ~~et~~ method গুলো কতটুকু connected [closely connected কিনা].

Cohesive (+) → Q
Non-cohesive (-) → P
] pair

#





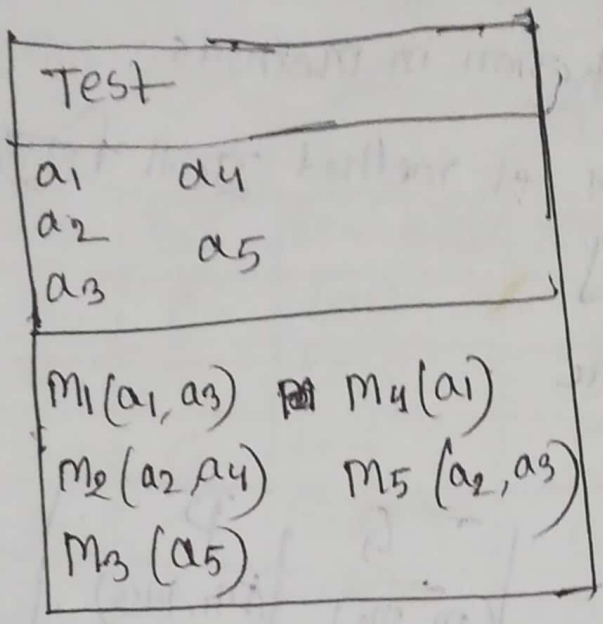
P > Q ; LCOM = 2 - 1 = 1

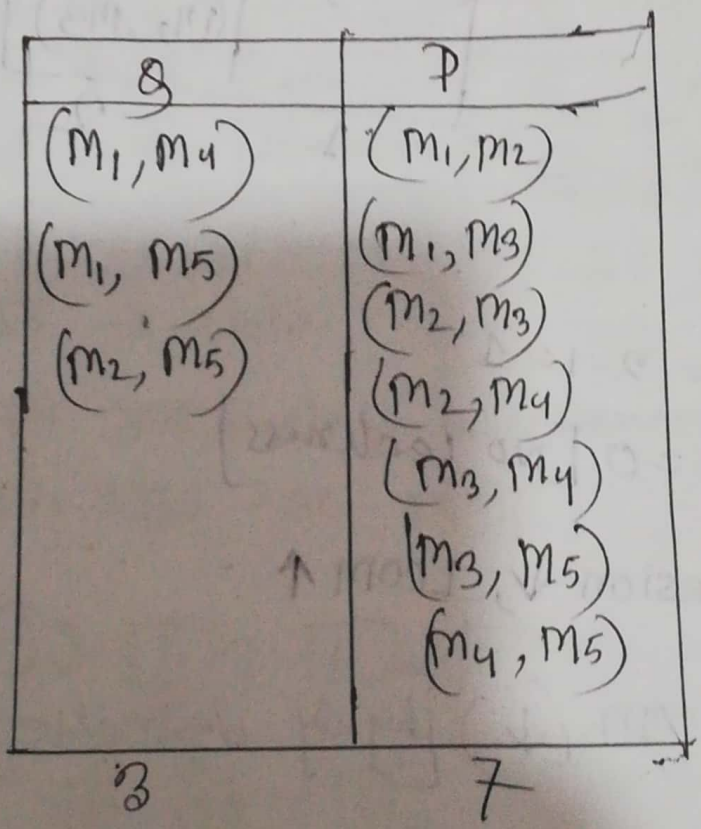
if Q >/ P ; LCOM = 0 [No Lackness]

Coupling ↑, Cohesion ↓, LCOM ↑

* Cohesion (↑), LCOM (↓) [highly desirable property for modularity]

#


Test
a1 a4
a2 a5
a3
m1(a1, a3) m4(a1)
m2(a2 a4) m5 (a2, a3)
m3 (a5)


Q
P
(m1, m4)
(m1, m5)
(m2, m5)
(m1, m2)
(m1, m3)
(m2, m3)
(m2, m4)
(m3, m4)
(m3, m5)
(m4, m5)
3
7

∴ P ⊃ Q

LCOM = 7 - 3 = 4

cocomo

cocomo → Constructive cost model

Project category

(i) Organic (ii) Semidetached (iii) Embedded

Organic

- small size ↗ SLOC (2-50 KLOC)
- less innovative [traditional, no complex scenario]
- small team but good experience people.

example: showing VUES information to webpage, payroll system.

Semidetached

- ~~midi~~ medium size (50-300 KLOC)
- work mixed experience level

Embedded

large size (>300 KLOC)

- innovative

- Effort = PM = coefficient * (SLOC/1000)^P
  <Effort factor>
- Development time = DM = 2.50 * (PM)^T
- Required number of people = ST = PM / DM

Software Project Type	Coefficient <Effort Factor>	P	T
Organic	2.4	1.05	0.38
semi-detached	3.0	1.12	0.35
Embedded	3.6	1.20	0.32

PM = Person-months needed for project (labor working hours)

P: Project Complexity (1.04 - 1.24)

DM: duration time in months per project (week days)

T: SLOC - dependent Coefficient (0.32 - 0.38)

ST: average staffing necessary

Q. Find required number of people for a semi-detached type project having KLOC = 70

SLOC = 70000 / 1000

PM = Coefficient * $(70)^{P}$

$= 3 * (70)^{1.12}$

$= 349.648 \approx 350$

~~DM = PM * $(2.50)^{T}$~~

~~$= 350 * (2.50)^{0.35}$~~

DM = $2.50 * (PM)^{T}$

$= 2.50 * (350)^{0.35}$

$= 19.42$

$St = PM / DM = \frac{350}{19.42} = 18.02 \approx 18$

Chapter 10 : Design Pattern

* Design pattern describes a problem that occurs over and over in software engineering.
* Describes the solution in a sufficiently generic manner as to be applicable in a wide variety of contexts.

SINGLETON Design pattern

* একটির বেশি obj create করা যাবে না।

The singleton pattern ensures you have at most one instance of a class in your application (one kind of a object)

If more than one instantiated → Incorrect program behavior, overuse of resources, inconsistent result.

* Heap is the dynamic memory.
* Stack " " static " .

```
Student S1 = new Student
```

* constructor can be also private.
* " private হলে obj create করা যাবে না। (class এর বাইরে থেকে)

* class এর member ২ ভাবে access করা যায়।

1) object oriented না হলে static.

Example: student.M1 ()

2) object oriented হলে reference.

• Step of applying SINGLETON design pattern

1) First declare the static obj refference of same class.

2) Declare the constructor as private.

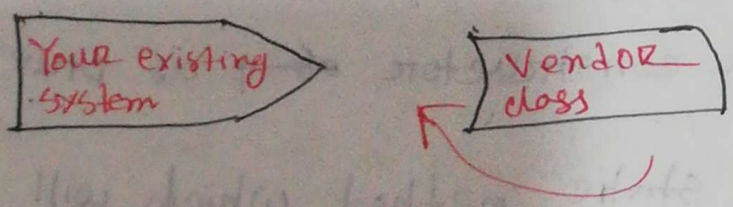
3) Declare a static method which will return the current object refference.

4) Inside static method give the proper logic so that more than one instance cannot be created.

ADAPTER Design pattern

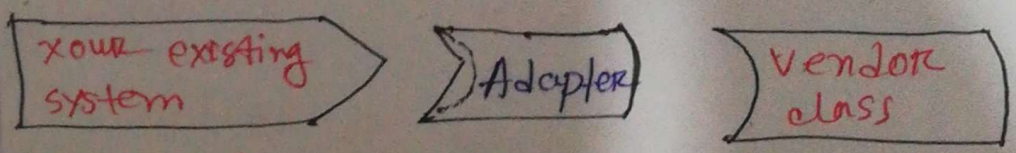
- An adapter is used when you need to use an existing class
- An adapter changes an interface into one that a client expects.

"putting a square plug in a round socket"



Their interface doesn't match the one you've written your code against. Not going to work.

Write a class that adapts the new vendor interface into the one you're expecting



No code changes | New code | No code changes

FAÇADE Design pattern

* complex system কে easier version এ কাজ করা যায় কাজ

- easier to use.

*** - A façade "wraps" a set of object to simplify.

*** - Façade pattern hides all the complexity behind.

OBSERVER Design pattern

- একটি object এর সাথে অনেক multiple obj connect থাকে।

- একটি Object change হলে বাকিগুলো notify হয়ে যায়।

* It is mainly used to implement distributed event handling systems.