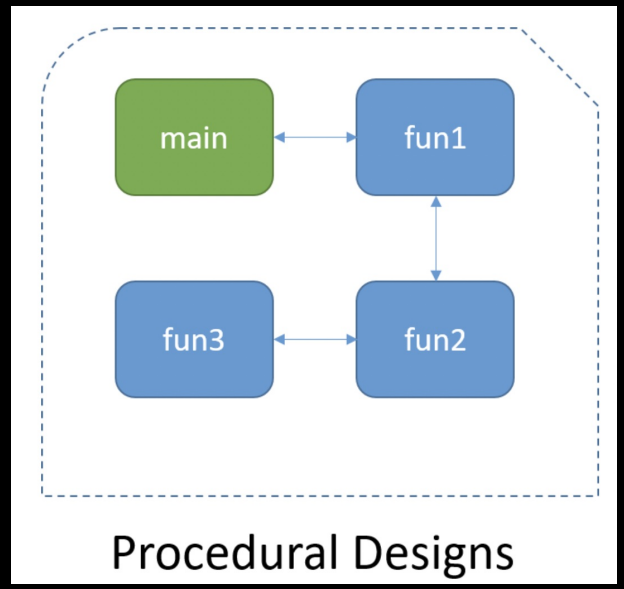
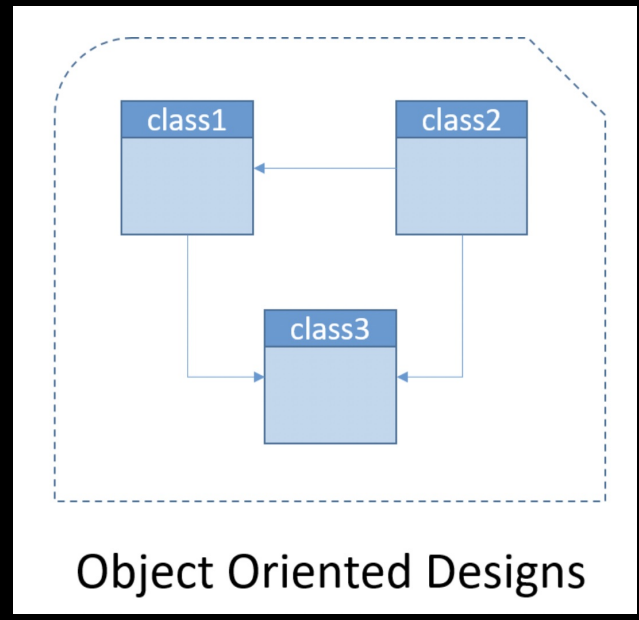
Software Design

→ graphical representation or model to represent interaction between various modules in software.

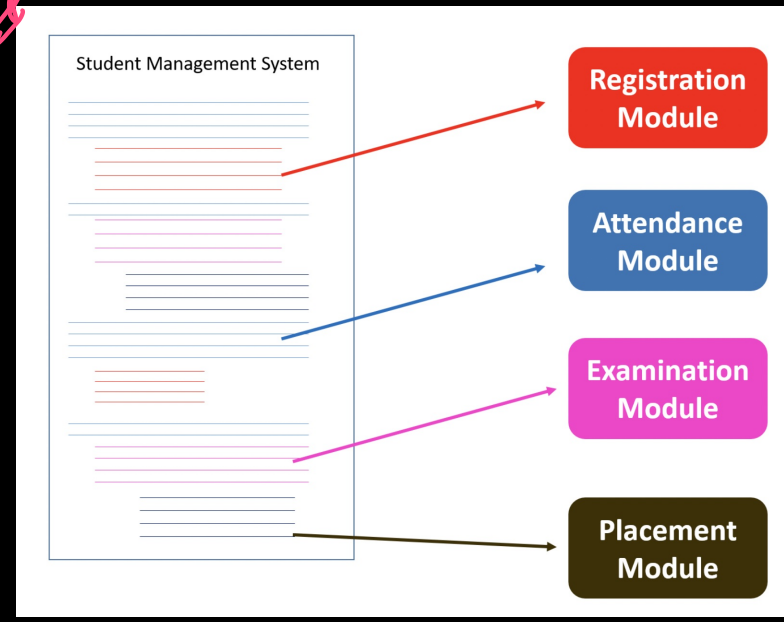
↳ classes / interfaces / methods in Java (OOP language)

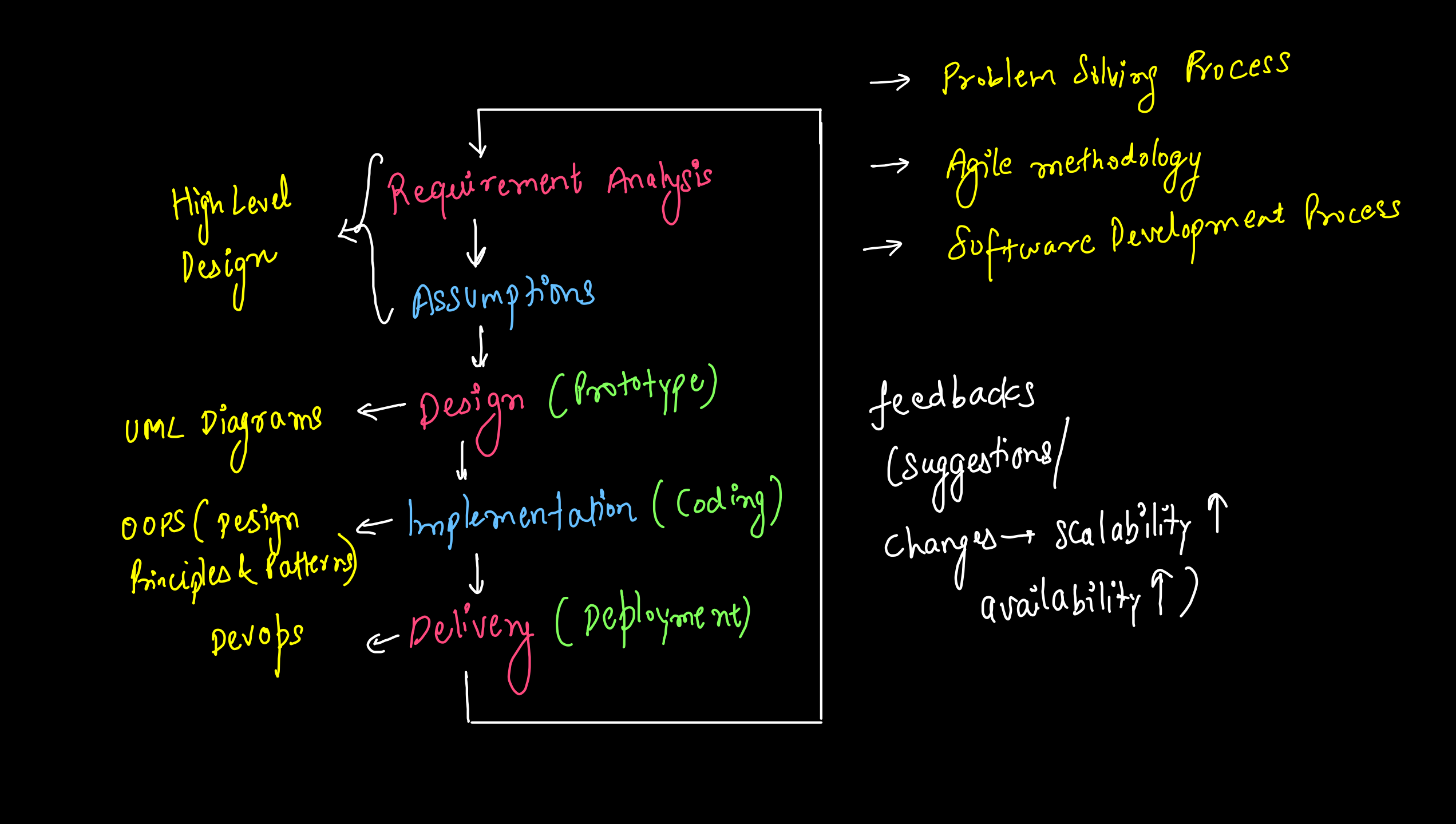


for C procedural language



Eg

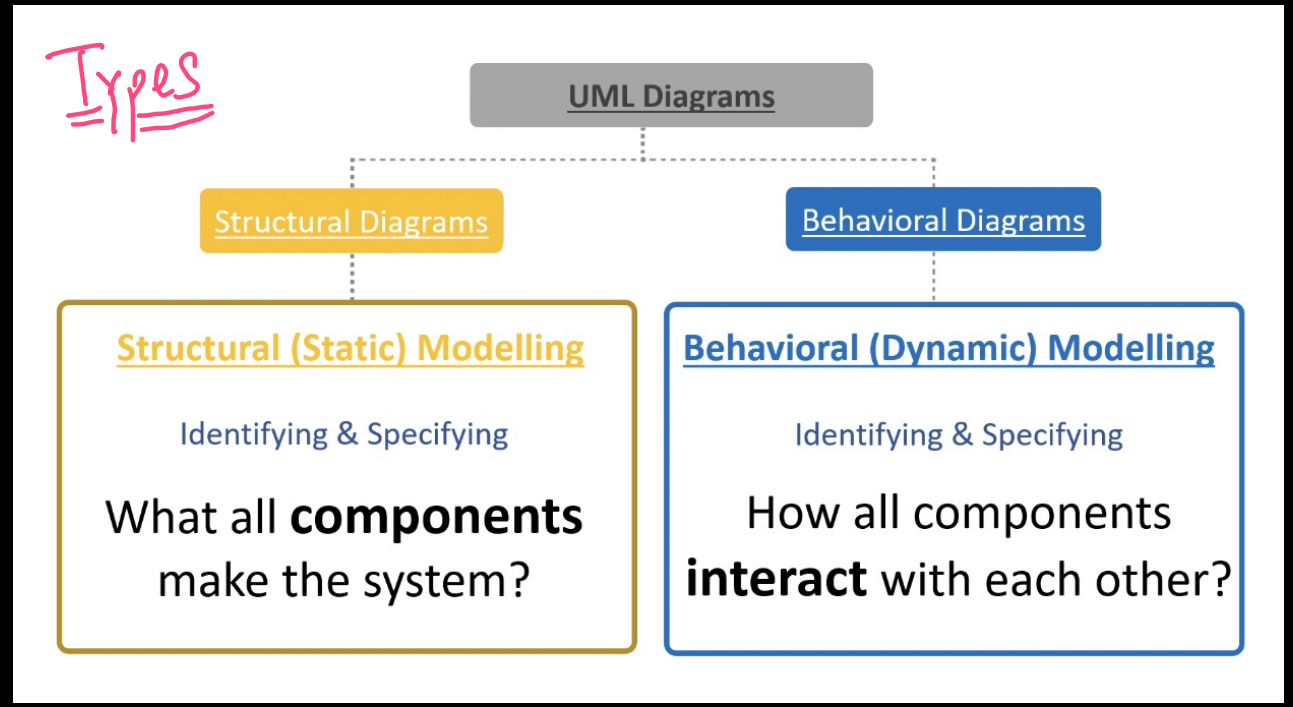


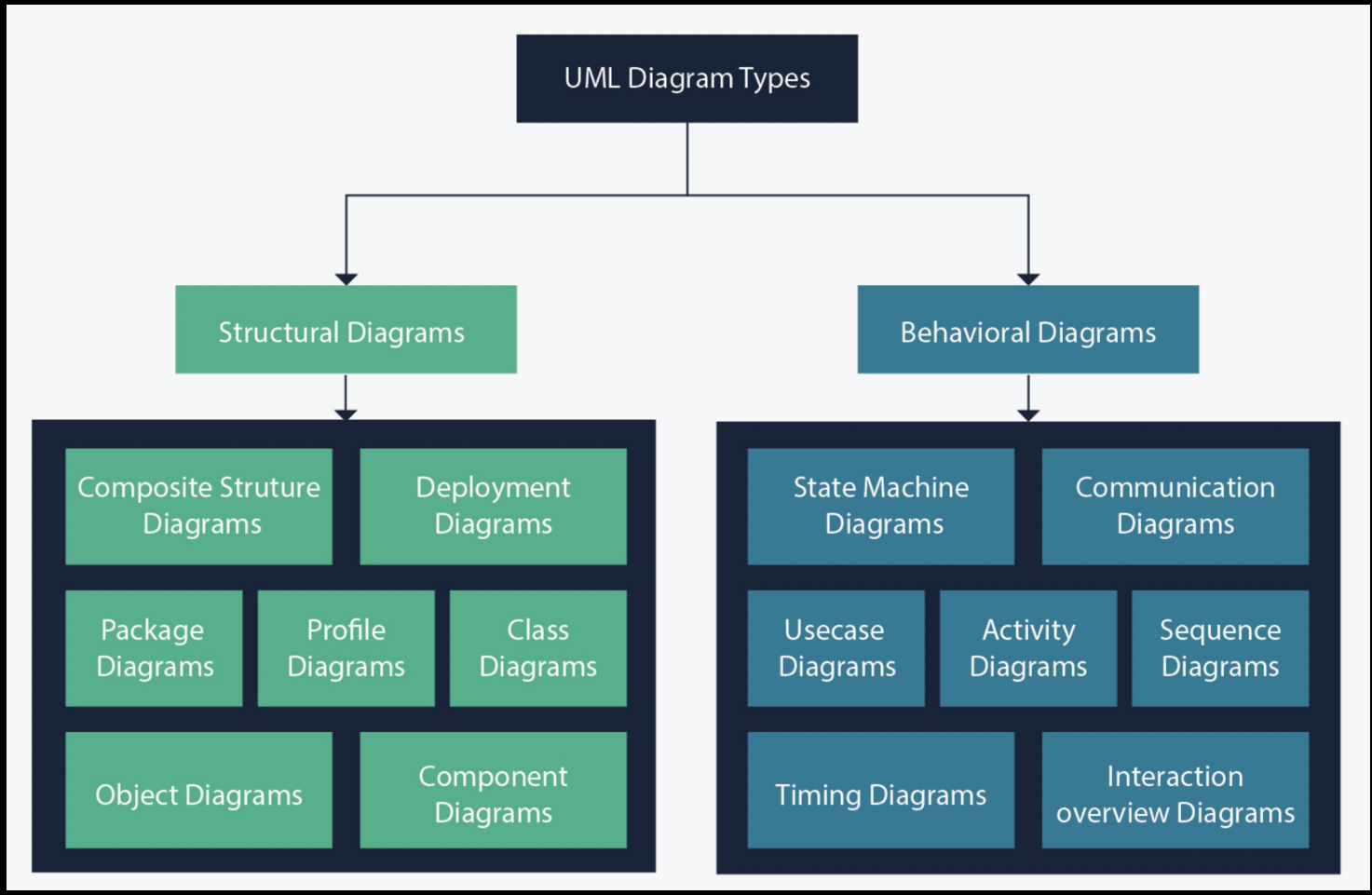

→ Problem Solving Process
→ Agile methodology
→ Software Development Process
High Level Design
Requirement Analysis
Assumptions
Design (Prototype)
UML Diagrams
Implementation (Coding)
OOPS (Design Principles & Patterns)
Delivery (Deployment)
Devops
feedbacks (suggestions/ Changes → scalability ↑ availability ↑)

Unified Modelling Language (UML) Diagrams

- conventional & standard set for designers/developers/Product Managers/Team Lead or Architect etc
- graphical representation of object oriented designs
- visual modelling language (not a programming language)
- describe classes/interfaces, objects, relationships between them (IsA/HasA)

Types





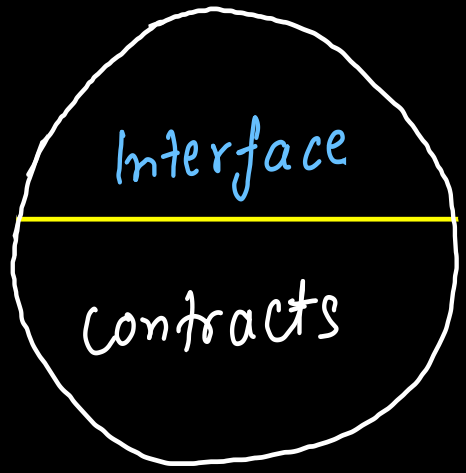
Important UML Diagrams

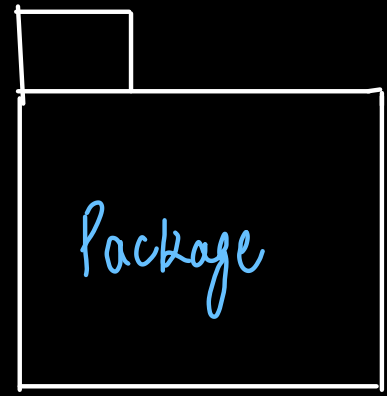
1. Class Diagram
2. Object Diagram
3. Use-case Diagram
4. Activity Diagram
5. Sequence Diagram

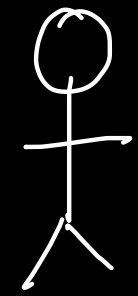
Common Notations in UML Diagrams

Set of Actions for specific goal

Annotation
(description/remarks/comments)

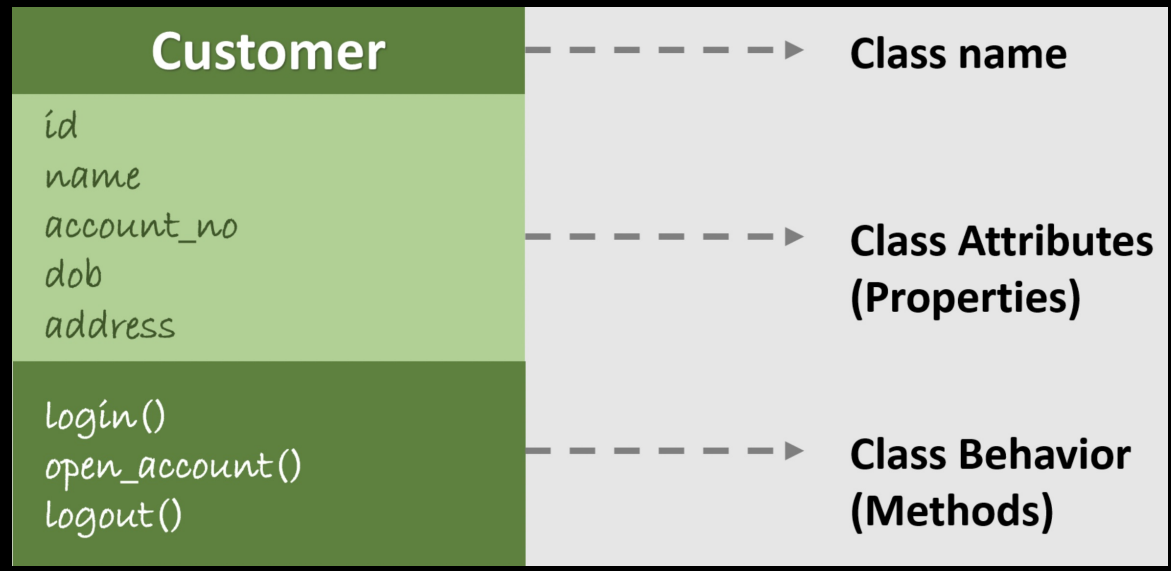


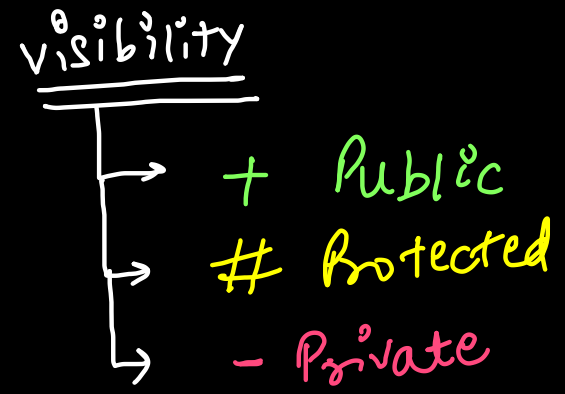




Actor
internal/external entity to interact

Class Diagrams

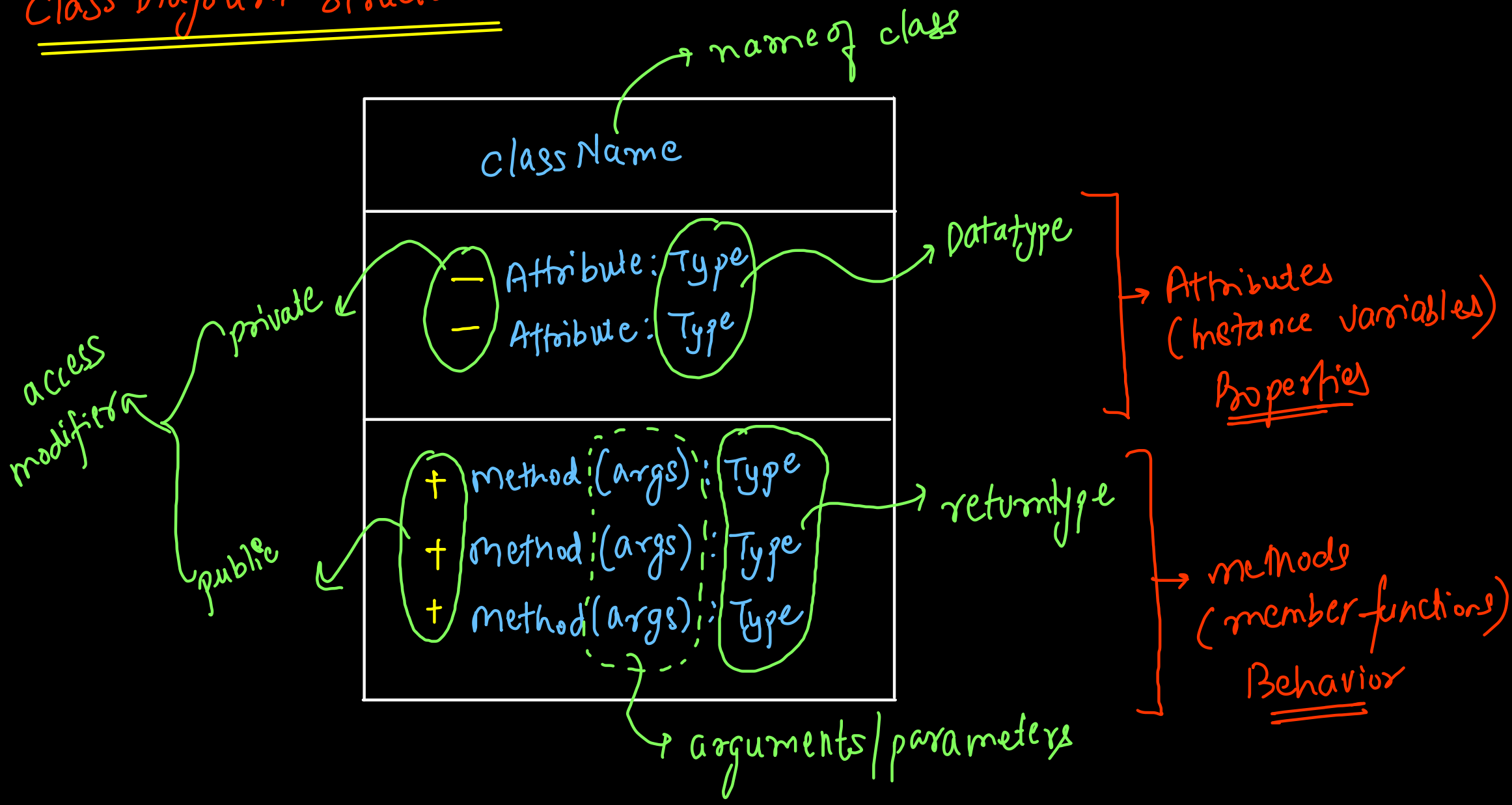




A class diagram in the Unified Modeling Language (UML) is a type of structural diagram that describes the structure of a system by showing the System's Classes, their Attributes, Operations (or methods), and the relationships among objects.

- provides static view of application
- represents responsibilities & relationships of various entities

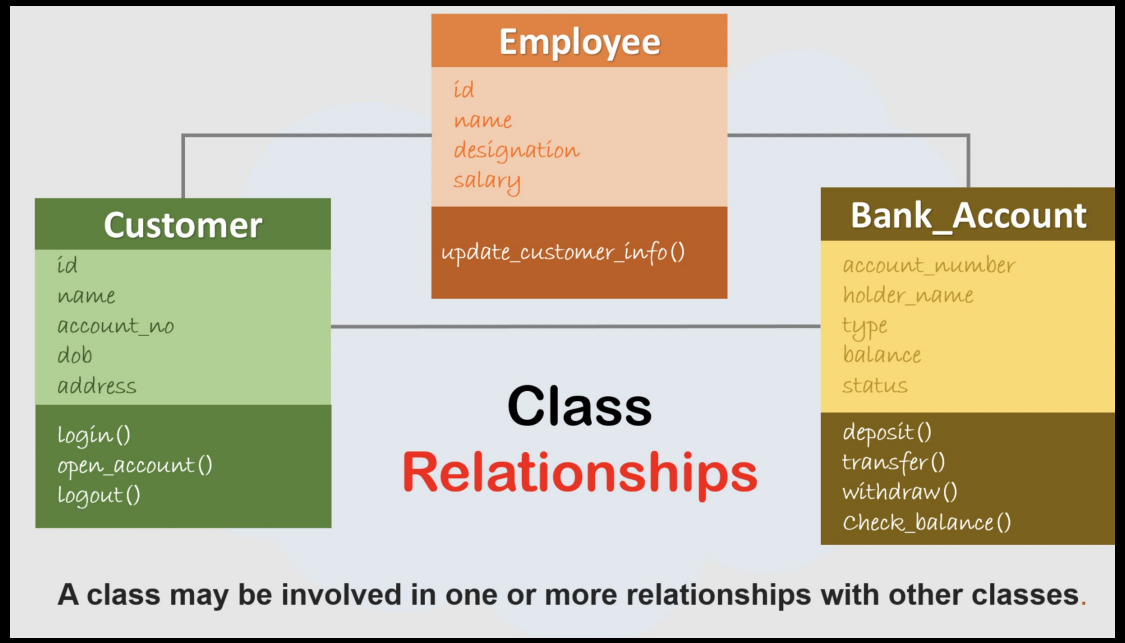
Class Diagram Structure

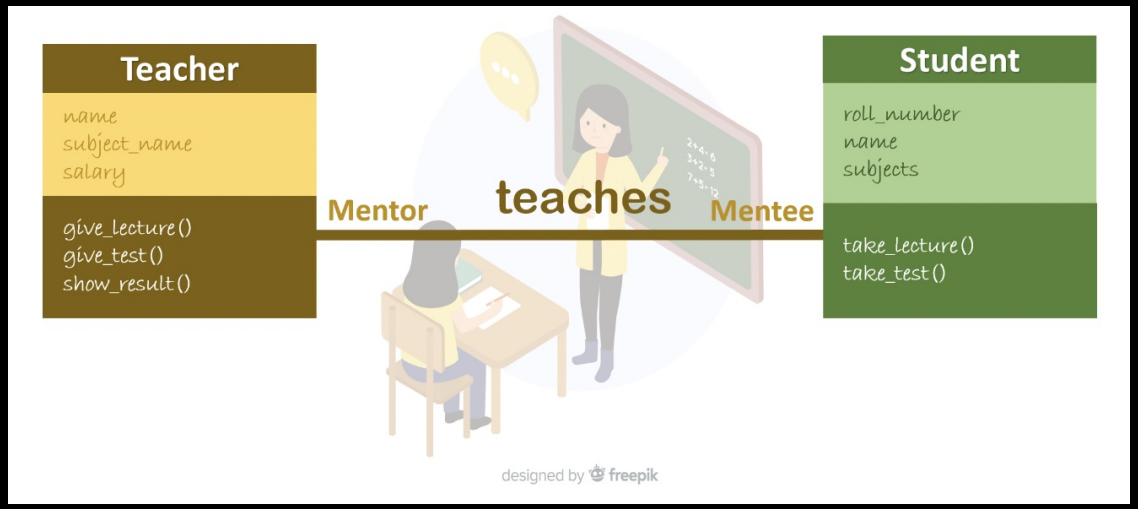


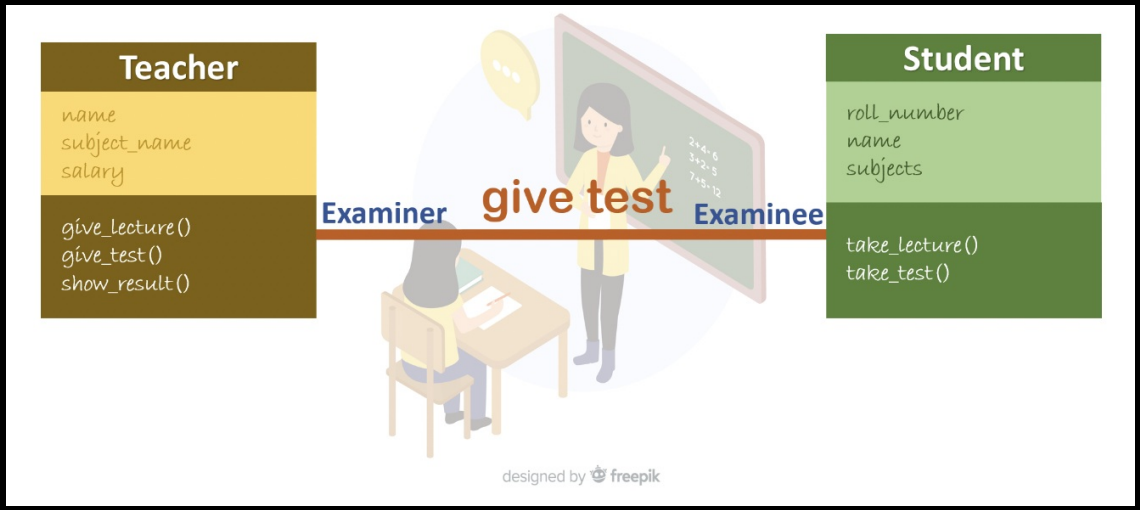
Relationships in Class Diagram

Connection b/w 2 classes / interfaces is known as relationship

Example :→ Virtual Bank System







Types of Relationships

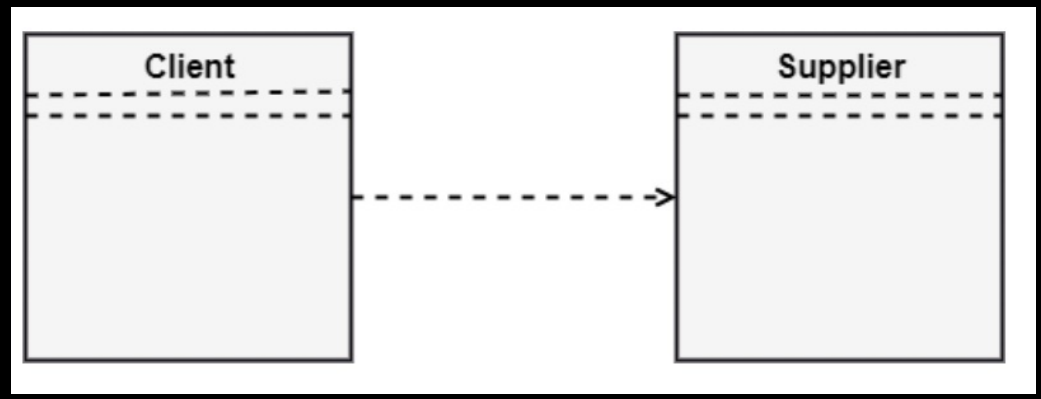
1. Dependency

unidirectional relationship

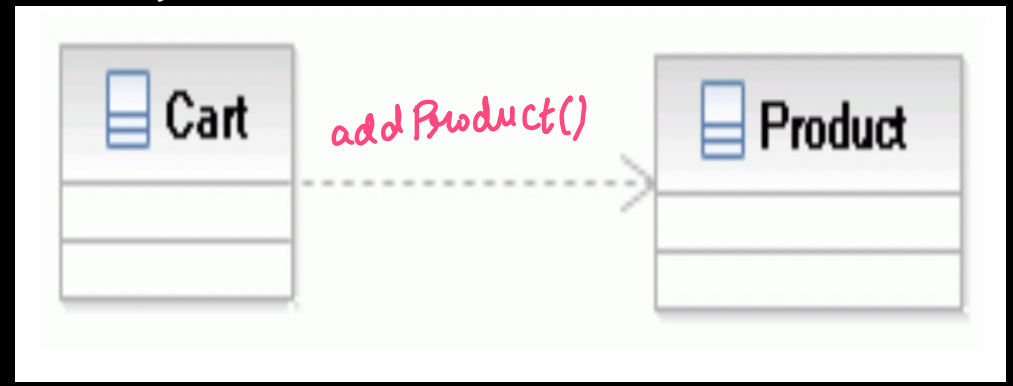
Class contained in another class
(as Nested class / Parameter of function / Implementing Interface)

change in structure/behavior of one class will affect the other class.

General Application



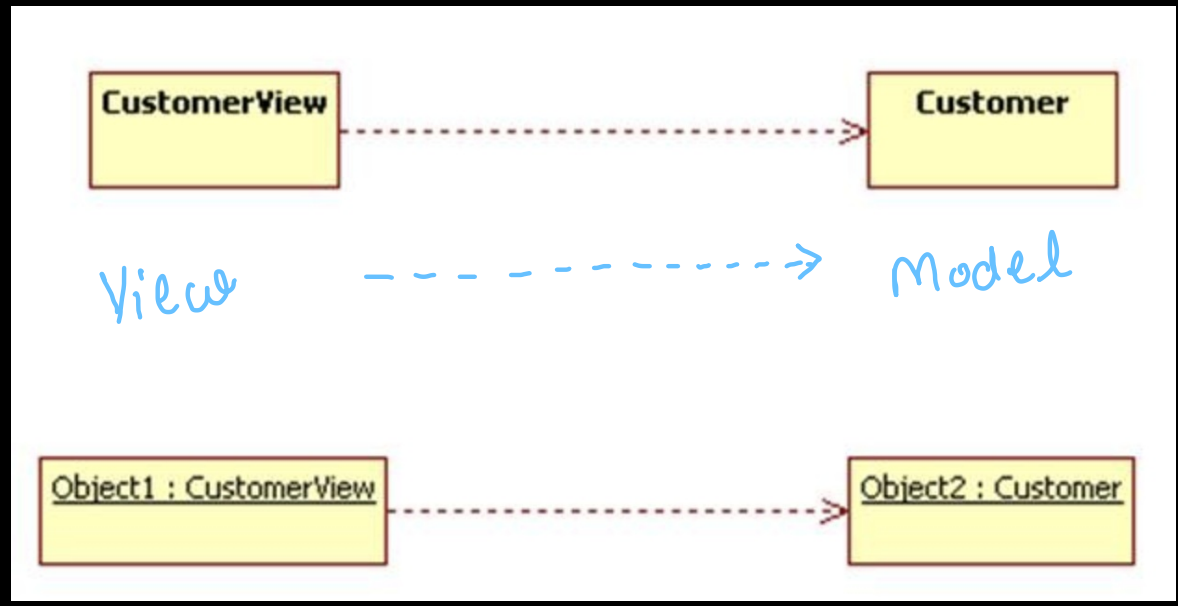
Eg1 : Amazon Cart/wishlist



eg2) Shape is an Interface, Circle is a class implementing Shape, hence Circle is dependent on Shape (needs to over-ride all abstract methods)

↳ If we change Shape, it affects Circle also.

eg3)



② Association ⟷ {has A relationship}

⇒ Instance (object) of one class is a property (instance variable) in another class' object.

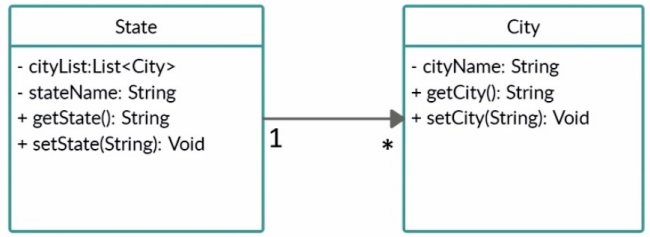
⇒ 1:1 or 1:many or many:1 or many:many relationship

⇒ Types → Aggregation (Weak association)
→ Composition (strong association)

⇒ directed relationship (one class has a object of another class)

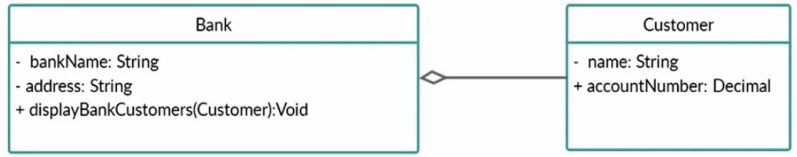
Multiplicity

Multiplicity indicates how many instances of a class participate in the relationship. It is a constraint that specifies the range of permitted cardinalities between two classes.



Aggregation

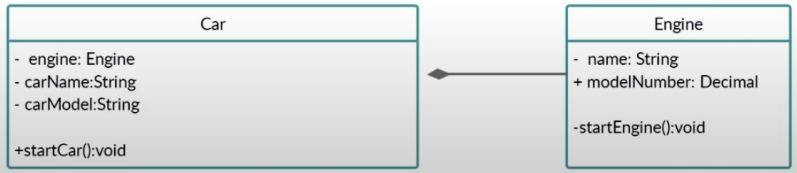
Aggregation represents the Has-A relationship.
It is a one-way relationship and called unidirectional association. For example, Bank can have customers but vice versa is not possible and that's why it unidirectional in nature.
Both the class's object will not affect each other.



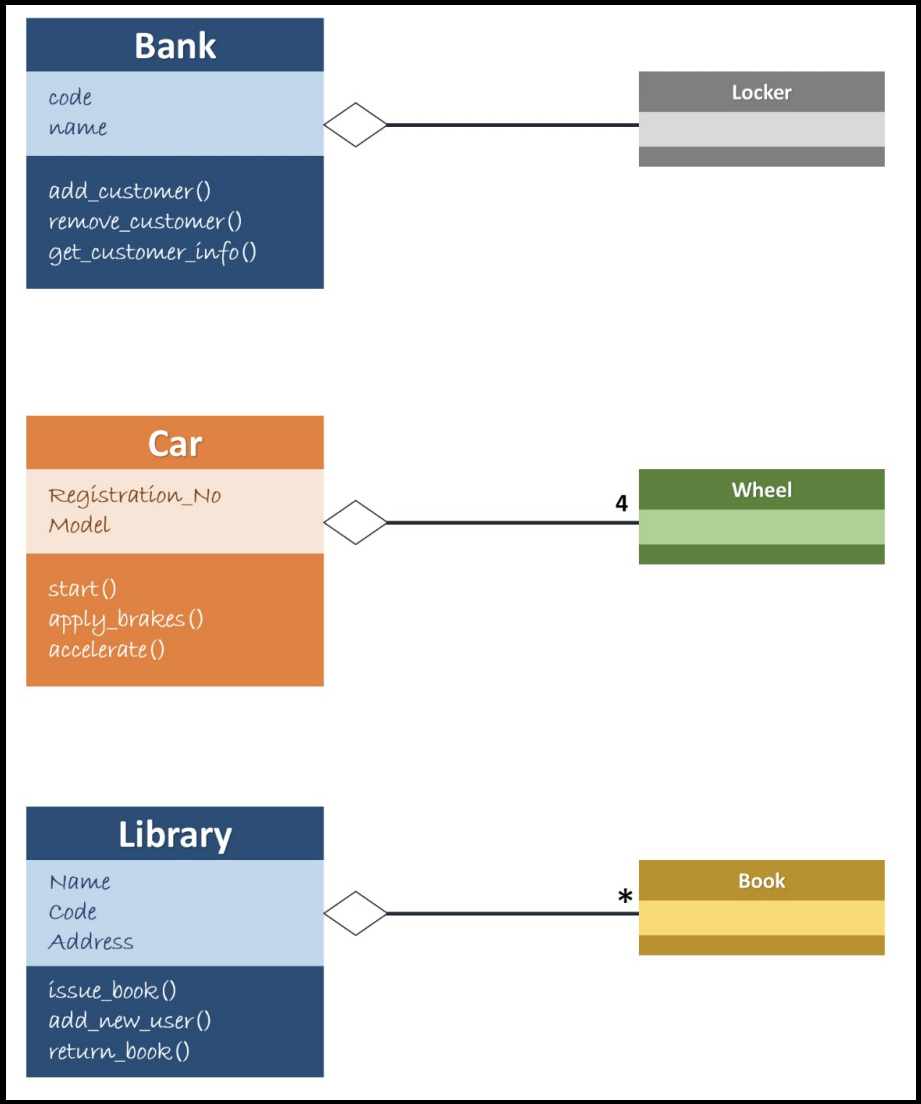
A has an instance of B.
B can exist without A

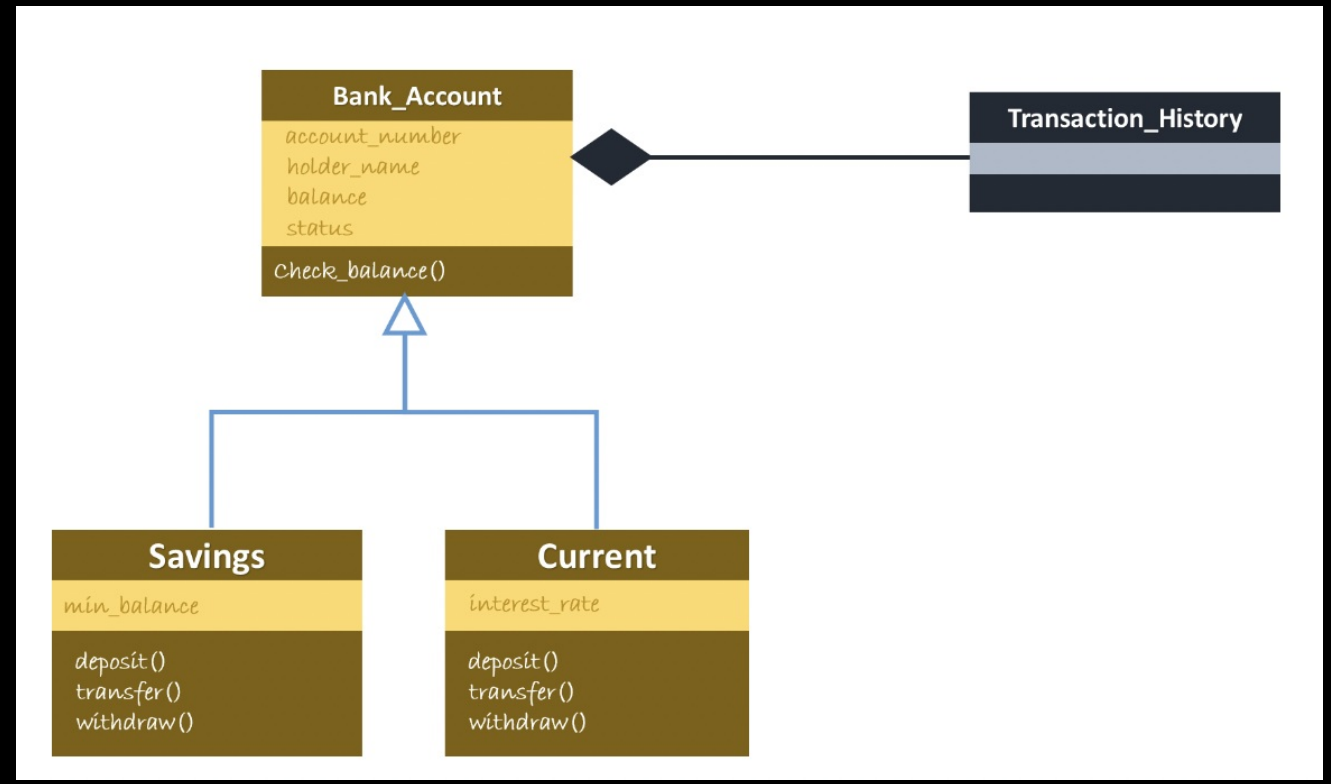
Composition

It is a restricted form of Aggregation. In composition two entities are highly dependent on each other. One entity cannot exist without the other.



A has an instance of B.
B can't exist without A

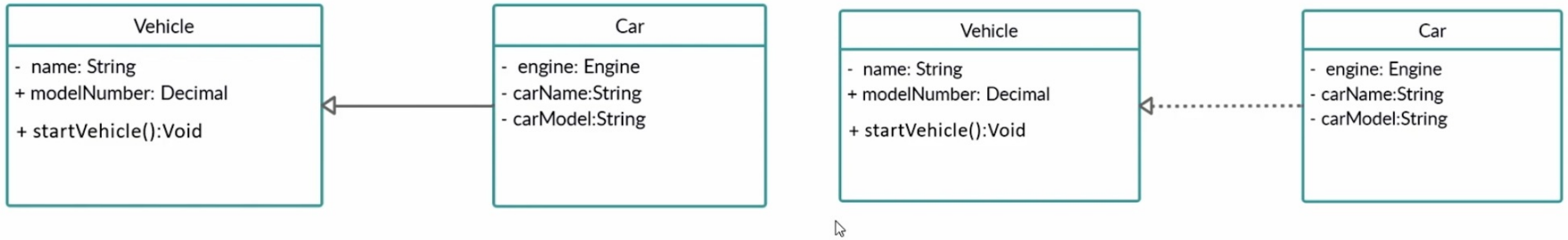

Bank
code
name
add_customer()
remove_customer()
get_customer_info()
Locker
Car
Registration_No
Model
start()
apply_brakes()
accelerate()
4
Wheel
Library
Name
Code
Address
issue_book()
add_new_user()
return_book()
*
Book


Bank_Account
account_number
holder_name
balance
status
Check_balance()
Transaction_History
Savings
min_balance
deposit()
transfer()
withdraw()
Current
interest_rate
deposit()
transfer()
withdraw()

3 Generalizat

Generalization

Generalization is the mechanism for combining similar classes of objects into a single, more general class. Generalization identifies commonalities among a set of entities.

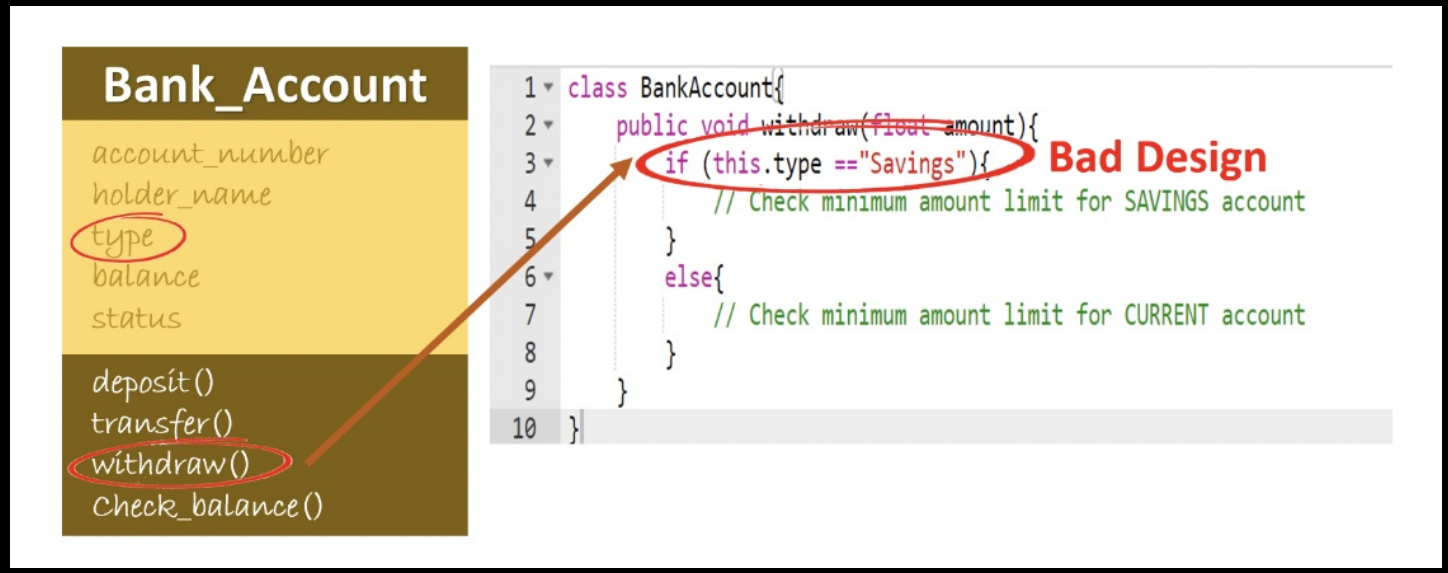


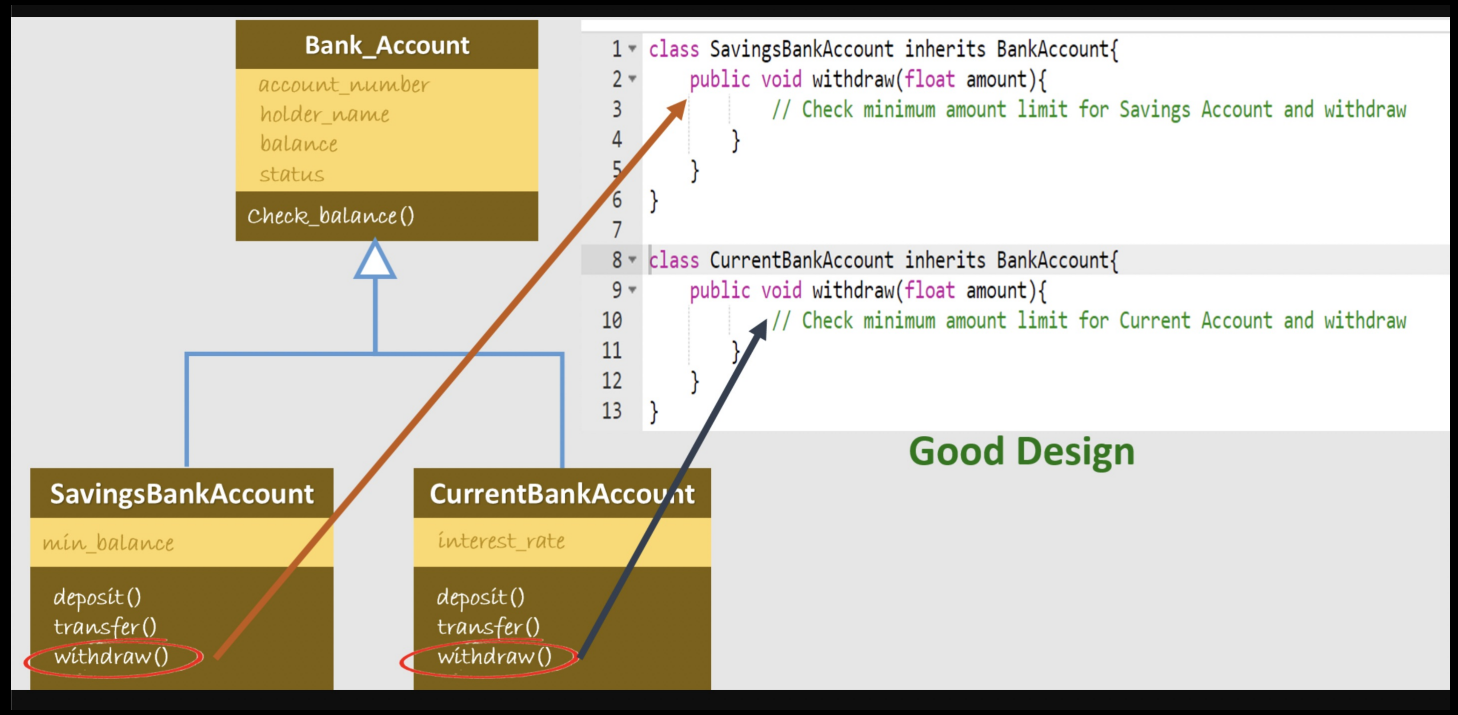
A ◁——— B

B inherits from A,
B “is a” A.

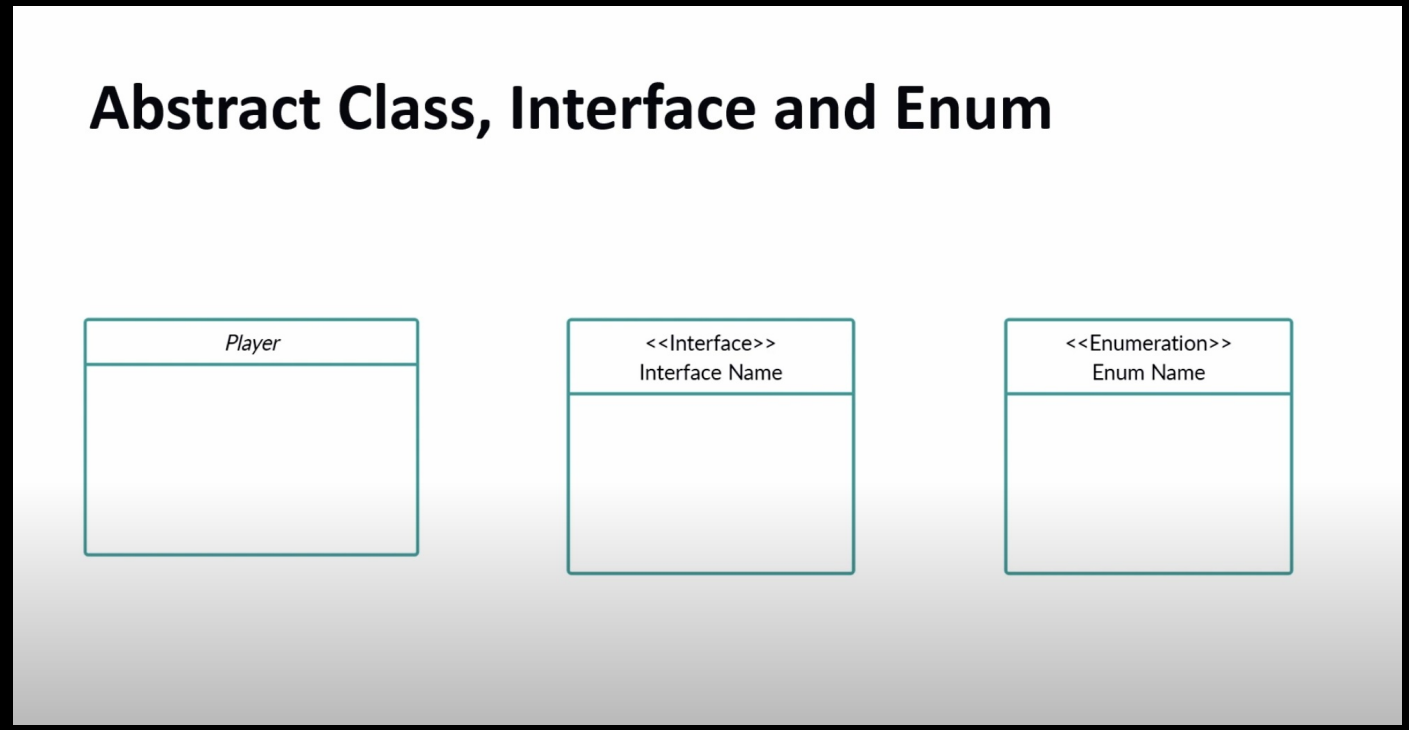
A ◁····· B

B Implements A

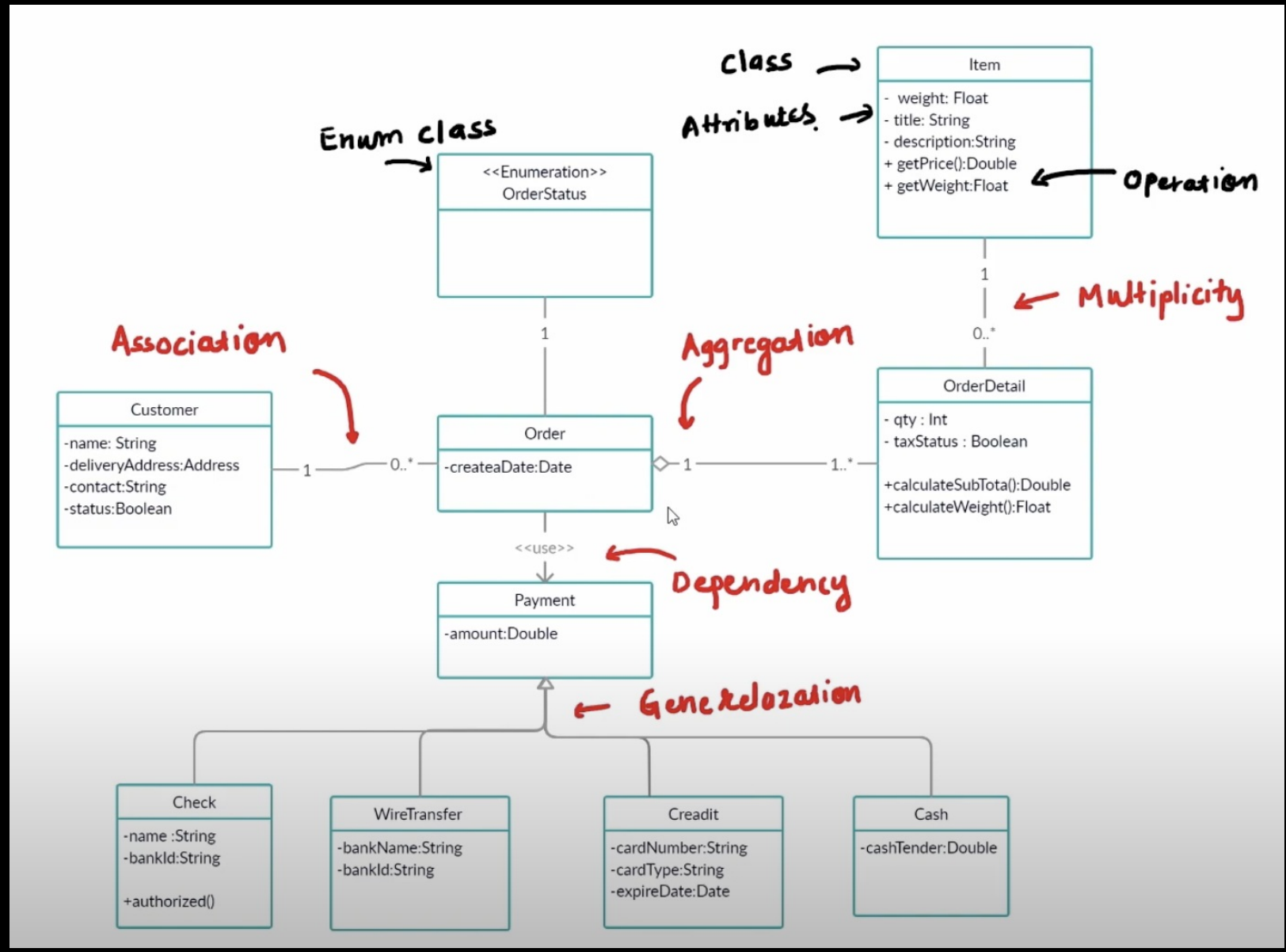

Bank_Account
account_number
holder_name
type
balance
status
deposit()
transfer()
withdraw()
check_balance()
class BankAccount{
public void withdraw(float amount){
if (this.type =="Savings"){
// Check minimum amount limit for SAVINGS account
}
else{
// Check minimum amount limit for CURRENT account
}
}
}
Bad Design

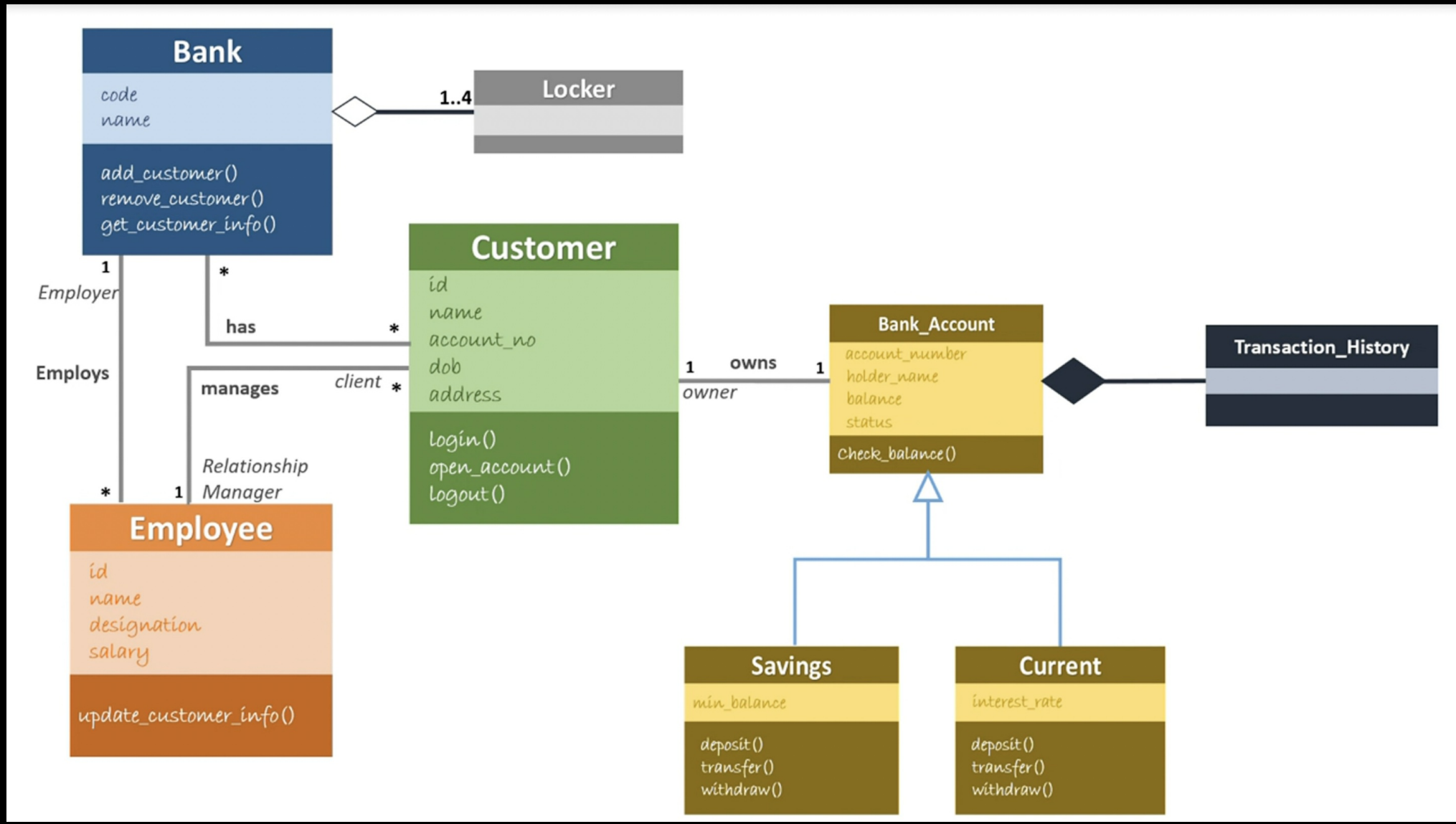

Bank_Account
account_number
holder_name
balance
status
Check_balance()
SavingsBankAccount
min_balance
deposit()
transfer()
withdraw()
CurrentBankAccount
interest_rate
deposit()
transfer()
withdraw()
class SavingsBankAccount inherits BankAccount{
public void withdraw(float amount){
// Check minimum amount limit for Savings Account and withdraw
}
}
}
class CurrentBankAccount inherits BankAccount{
public void withdraw(float amount){
// Check minimum amount limit for Current Account and withdraw
}
}
}
Good Design

4 Realization



Place An Order Class Diagram Example




Bank
code
name
add_customer()
remove_customer()
get_customer_info()
1..4
Locker
1
Employer
*
has
*
Employs
manages
client
*
Relationship
Manager
*
1
Customer
id
name
account_no
dob
address
login()
open_account()
logout()
1
owns
1
owner
Bank_Account
account_number
holder_name
balance
status
Check_balance()
Transaction_History
Employee
id
name
designation
salary
update_customer_info()
Savings
min_balance
deposit()
transfer()
withdraw()
Current
interest_rate
deposit()
transfer()
withdraw()

Object Diagram
(Abstract View)

Mr. Singh

Mr. singh :Customer

Mr. Agrawal

Mr. Agrawal :Customer

Class Diagram

Customer

id
name
account_no
dob
address

login()
open_account()
logout()

Object Diagram

Mr. Singh

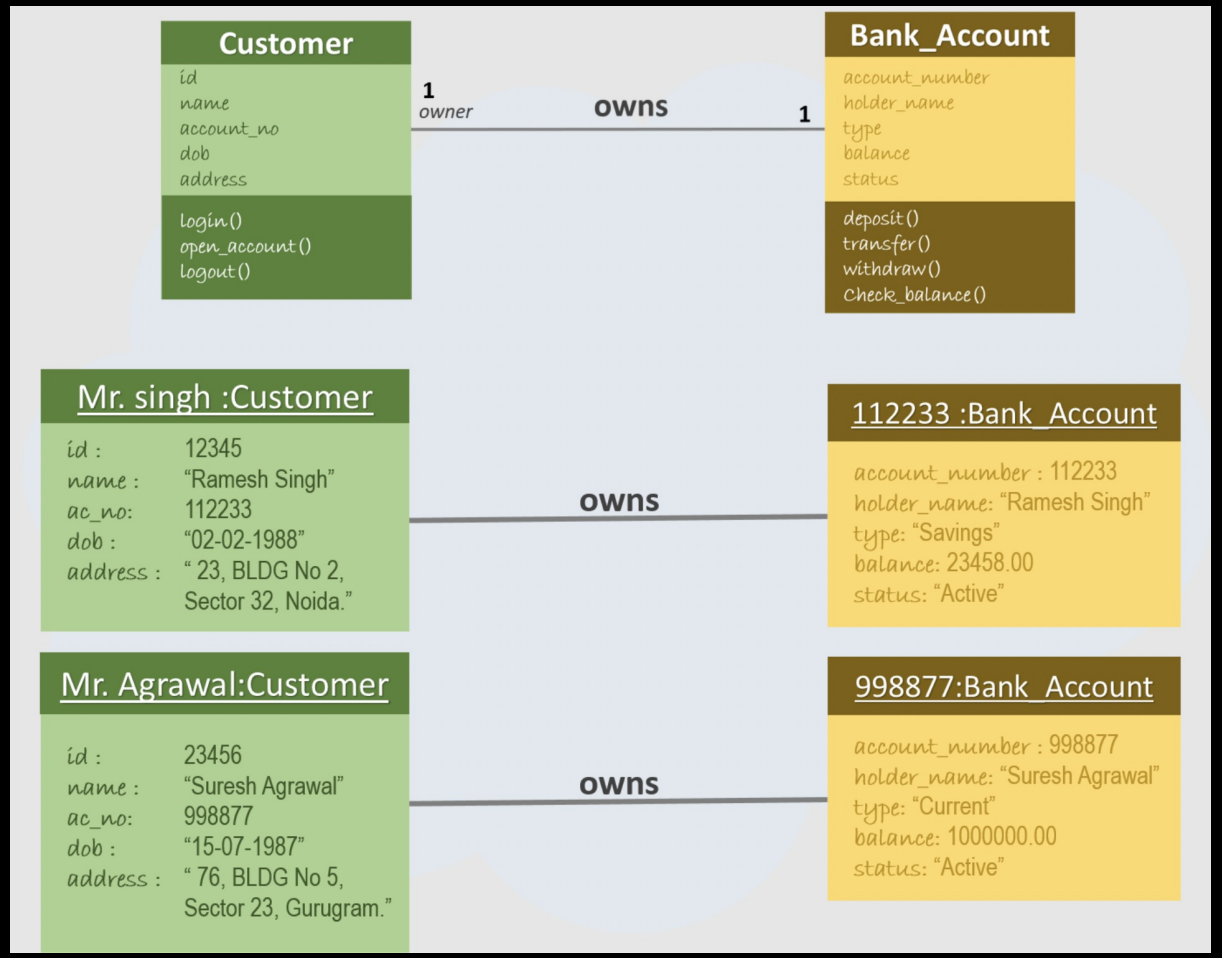
Mr. singh :Customer

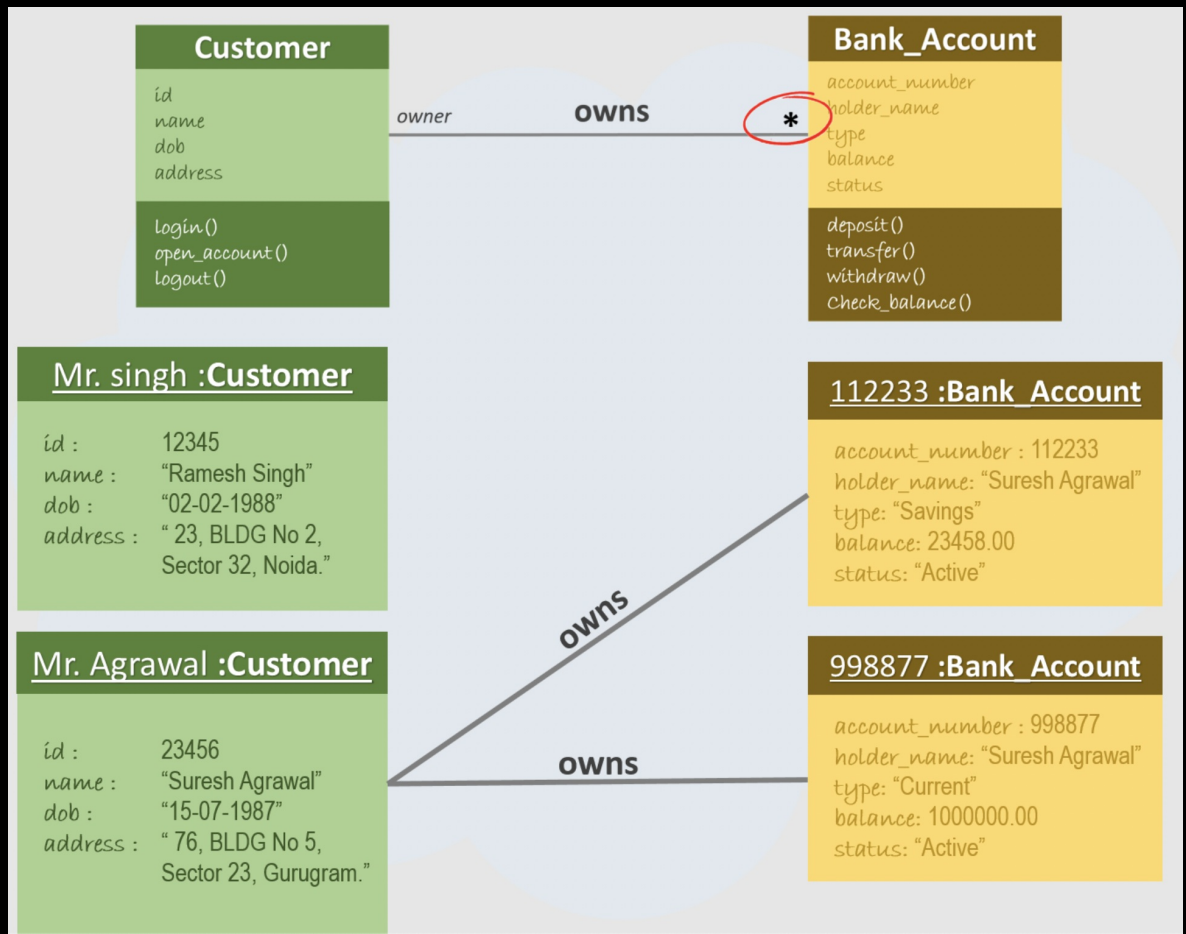
id =	12345
name =	"Ramesh Singh"
ac_no =	112233
dob =	"02-02-1988"
address =	" 23, BLDG No 2, Sector 32, Noida."

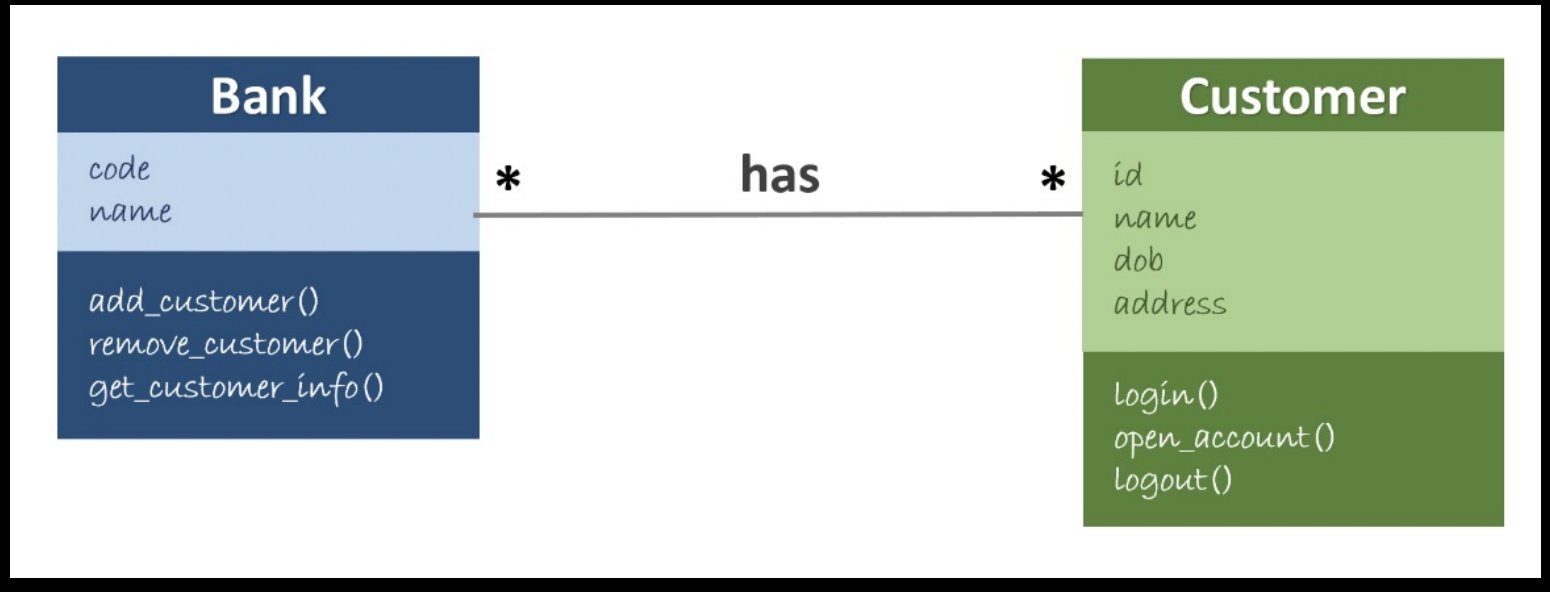
Mr. Agrawal

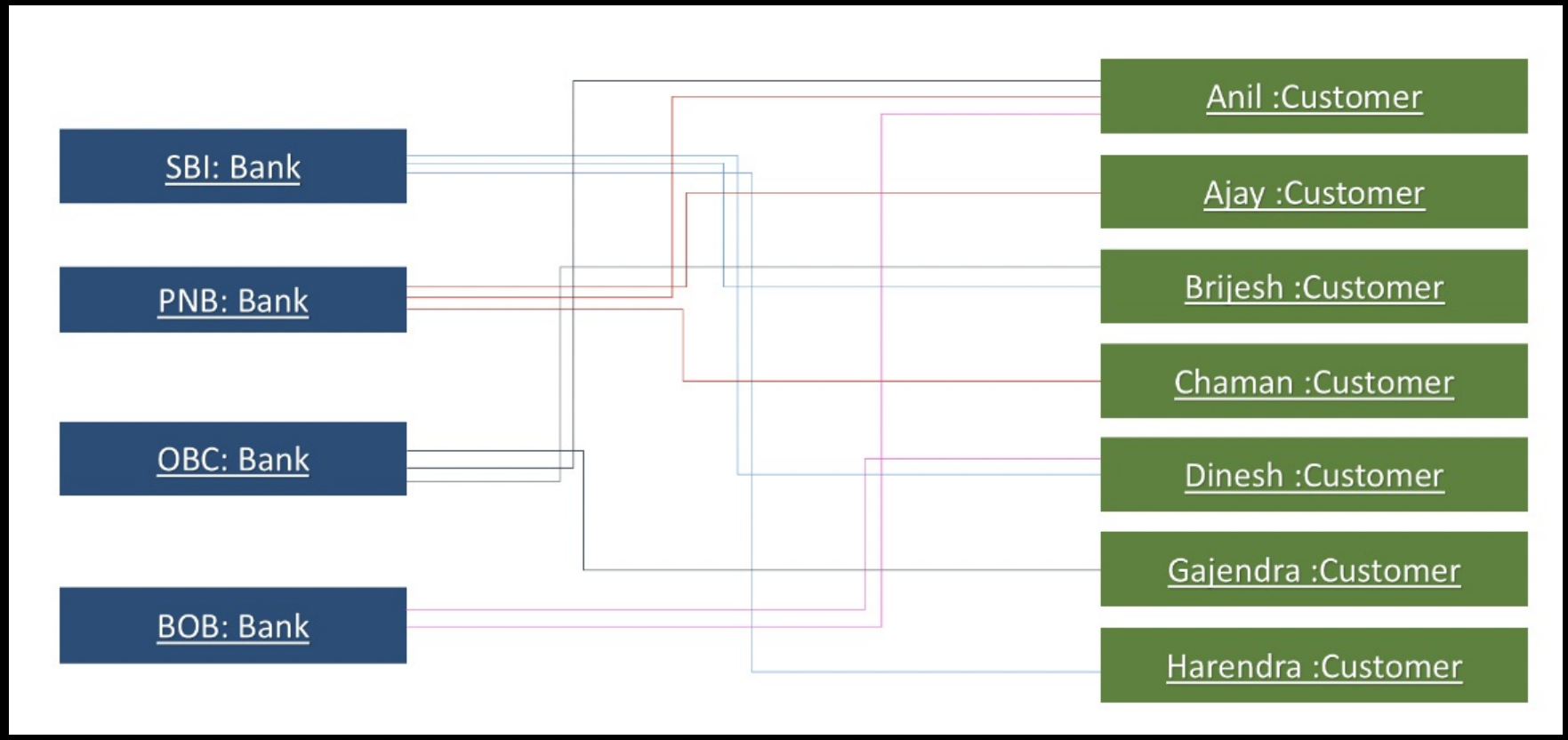
Mr. Agrawal:Customer

id =	23456
name =	"Suresh Agrawal"
ac_no =	998877
dob =	"15-07-1987"
address =	" 76, BLDG No 5, Sector 23, Gurugram."


Customer
id
name
account_no
dob
address
login()
open_account()
logout()
1
owner
owns
1
Bank_Account
account_number
holder_name
type
balance
status
deposit()
transfer()
withdraw()
Check_balance()
Mr. singh :Customer
id : 12345
name : "Ramesh Singh"
ac_no: 112233
dob : "02-02-1988"
address : " 23, BLDG No 2, Sector 32, Noida."
owns
112233 :Bank_Account
account_number : 112233
holder_name: "Ramesh Singh"
type: "Savings"
balance: 23458.00
status: "Active"
Mr. Agrawal:Customer
id : 23456
name : "Suresh Agrawal"
ac_no: 998877
dob : "15-07-1987"
address : " 76, BLDG No 5, Sector 23, Gurugram."
owns
998877:Bank_Account
account_number : 998877
holder_name: "Suresh Agrawal"
type: "Current"
balance: 1000000.00
status: "Active"


Customer
id
name
dob
address
login()
open_account()
logout()
owner
owns
*
Bank_Account
account_number
holder_name
type
balance
status
deposit()
transfer()
withdraw()
Check_balance()
Mr. singh :Customer
id : 12345
name : "Ramesh Singh"
dob : "02-02-1988"
address : " 23, BLDG No 2, Sector 32, Noida."
112233 :Bank_Account
account_number : 112233
holder_name: "Suresh Agrawal"
type: "Savings"
balance: 23458.00
status: "Active"
owns
Mr. Agrawal :Customer
id : 23456
name : "Suresh Agrawal"
dob : "15-07-1987"
address : " 76, BLDG No 5, Sector 23, Gurugram."
owns
998877 :Bank_Account
account_number : 998877
holder_name: "Suresh Agrawal"
type: "Current"
balance: 1000000.00
status: "Active"


Bank
code
name
add_customer()
remove_customer()
get_customer_info()
*
has
*
Customer
id
name
dob
address
login()
open_account()
logout()


SBI: Bank
PNB: Bank
OBC: Bank
BOB: Bank
Anil :Customer
Ajay :Customer
Brijesh :Customer
Chaman :Customer
Dinesh :Customer
Gajendra :Customer
Harendra :Customer

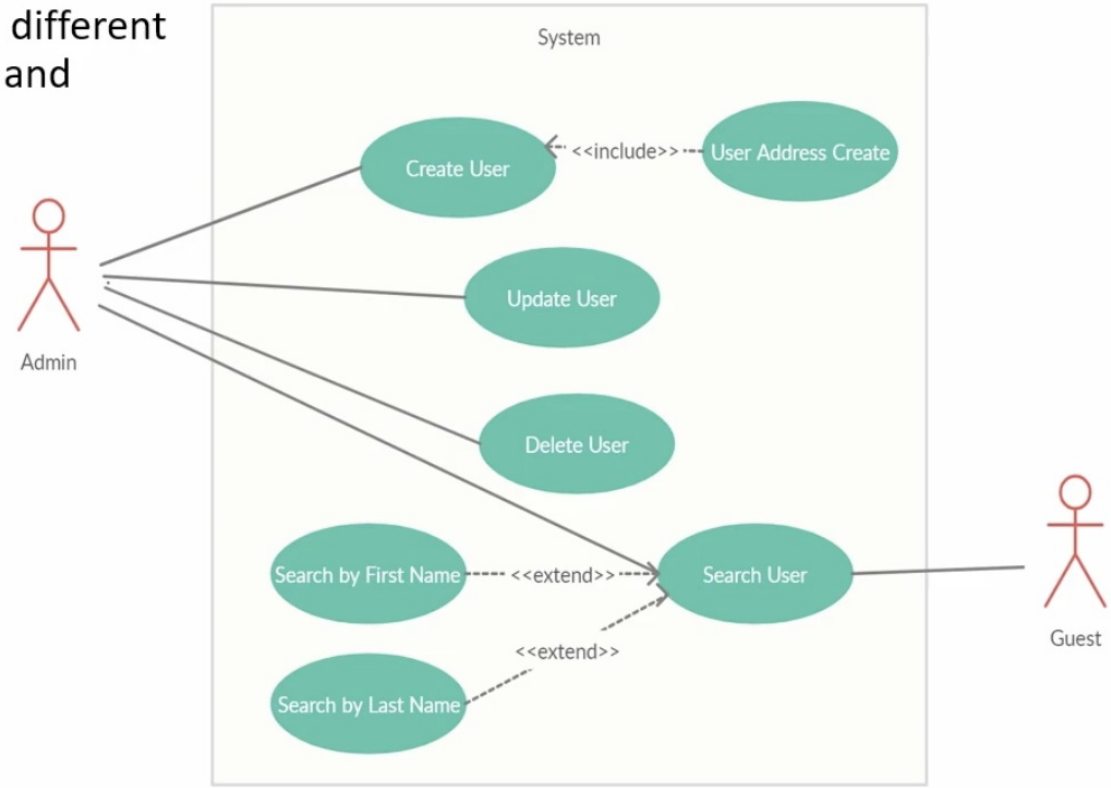
Use Case Diagram

Use Case diagrams are used to analyze the system's high-level requirements. These requirements are expressed through different use cases. Each use case should provide some observable and valuable result to the actors.

- High-level functional behaviour
- What system does from the user point of view
- What system will do and will not do"

Different components of the use case diagram

1. **System boundary**
2. **Actors**
3. **Use Case**
4. **Include**
5. **Extend**

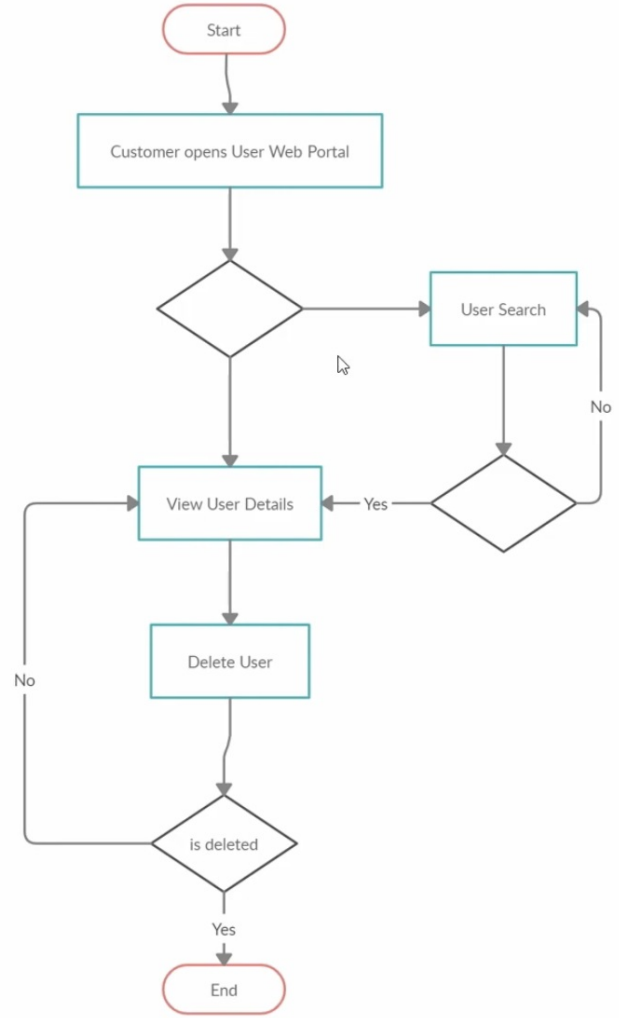


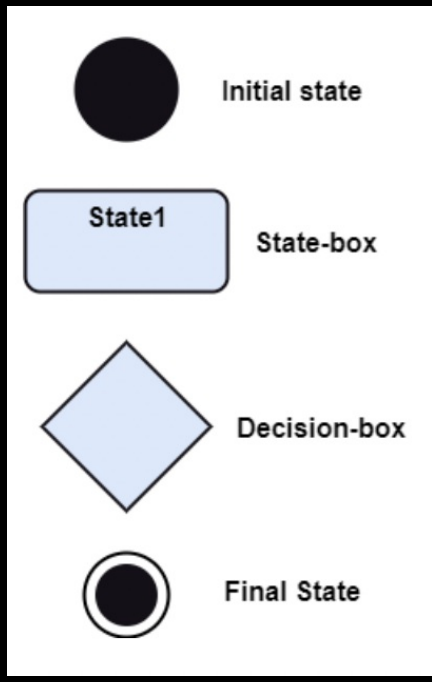
Activity Diagram

Shows the flow of control for a system functionality; it emphasizes the condition of flow and the sequence in which it happens. We can also use an activity diagram to refer to the steps involved in the execution of a use case.

An activity represents an operation on some class in the system that results in a change in the state of the system. Typically, activity diagrams are used to model workflow or business processes and internal operations.

deleteUser



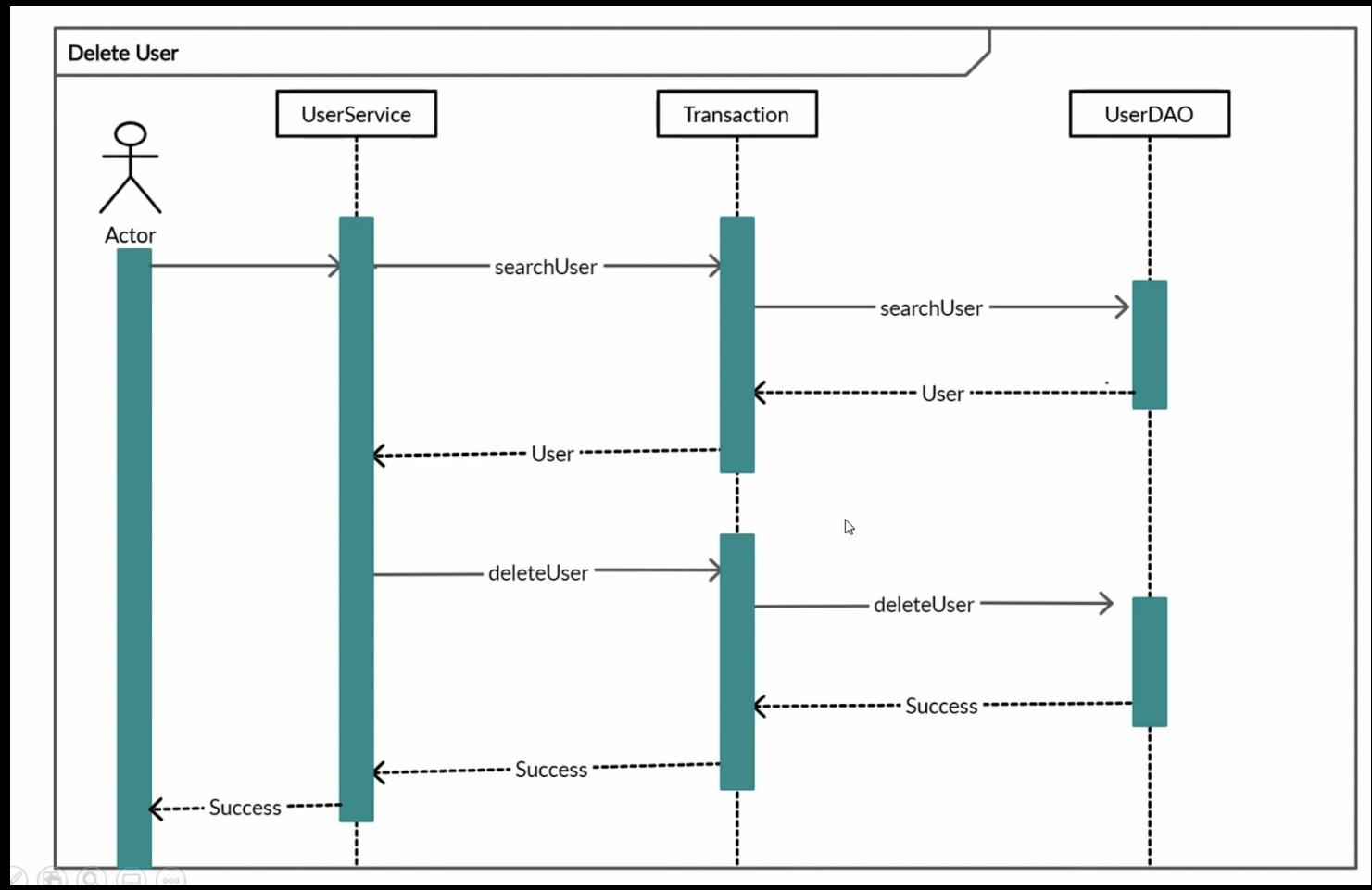


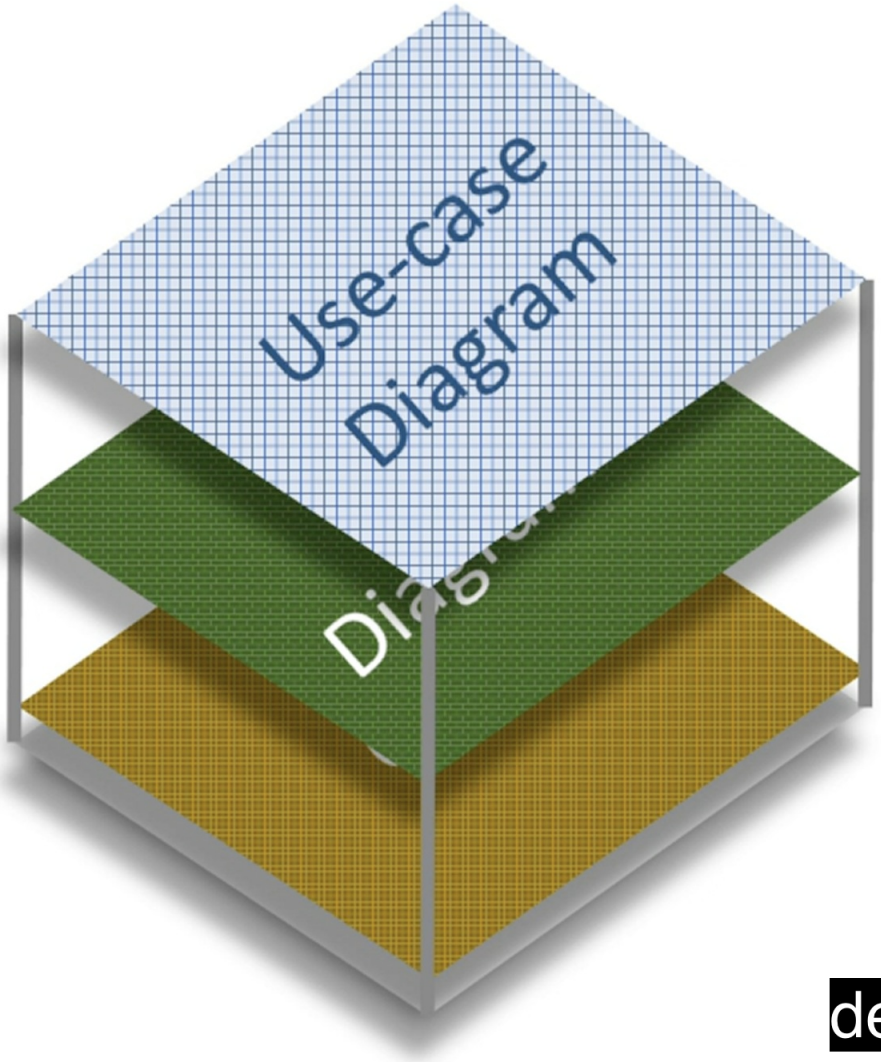
Sequence diagram

- Interactions among classes in terms of an exchange of messages over time.
- It's a calls between different Objects in their sequence.

Sequence diagram has two dimensions :

1. Vertical – Sequence of messages in chronological order.
2. Horizontal – Object instances to which messages are sent.





High-Level Design

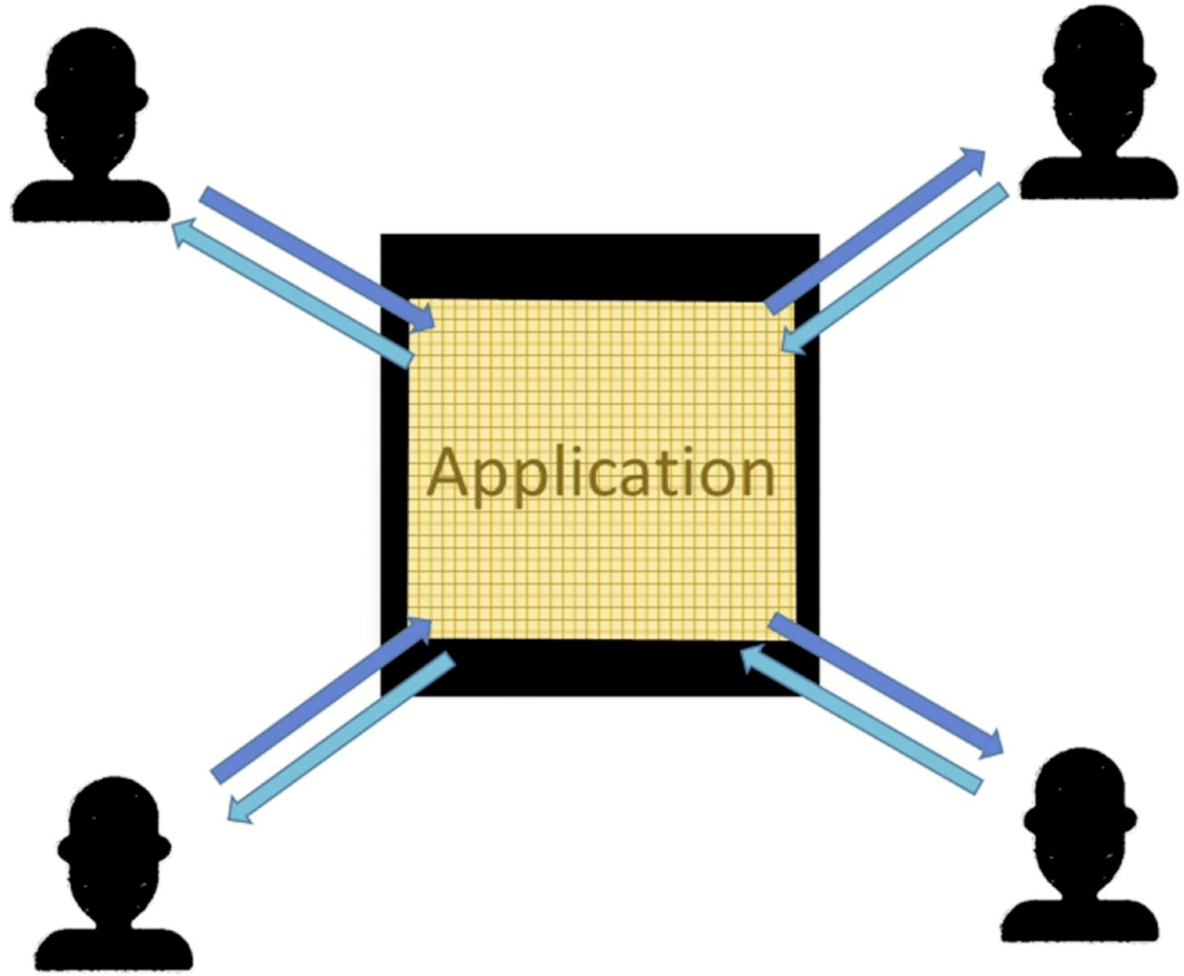
ABSTRACTION

DETAILS

Low-Level Design

details. Hence, we

Use-case Diagram



Use-case Diagram Components

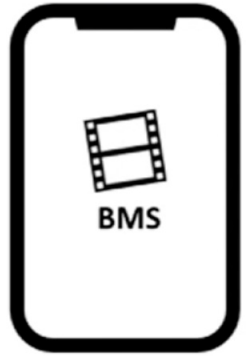
System

Actor

Use-case

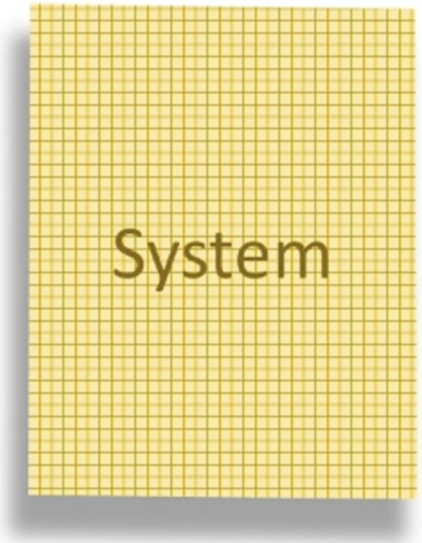
Relationship

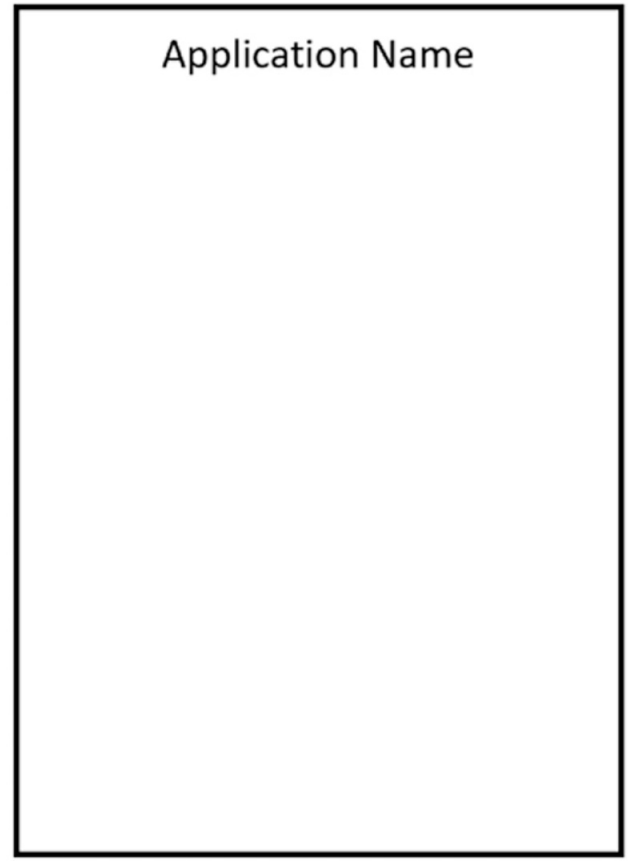
Website / Web-App



Mobile Application

Software Component



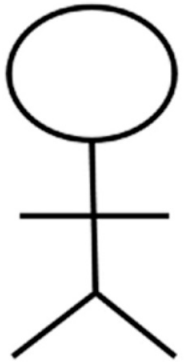


External Agent

System

Person

Application Name

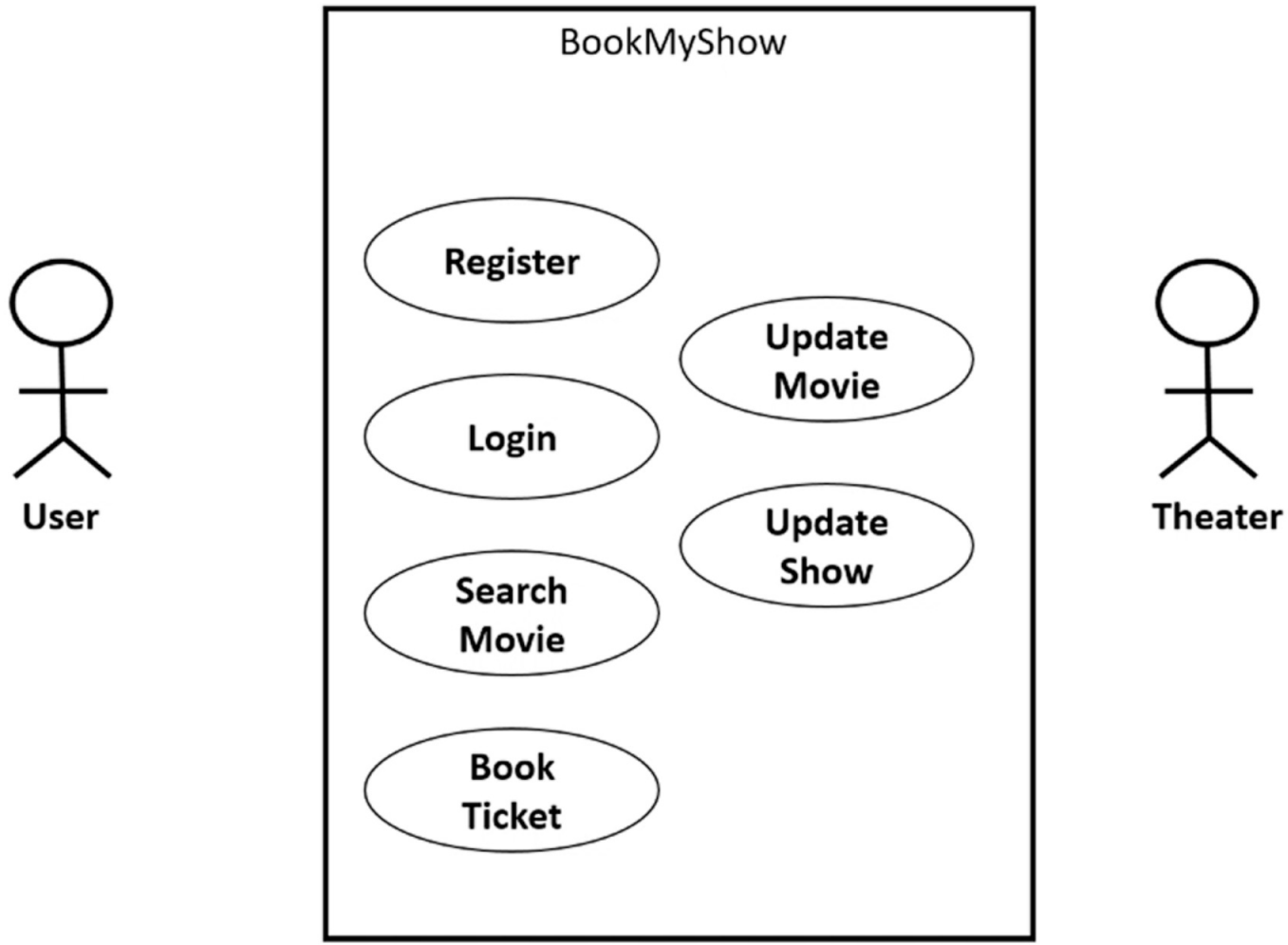


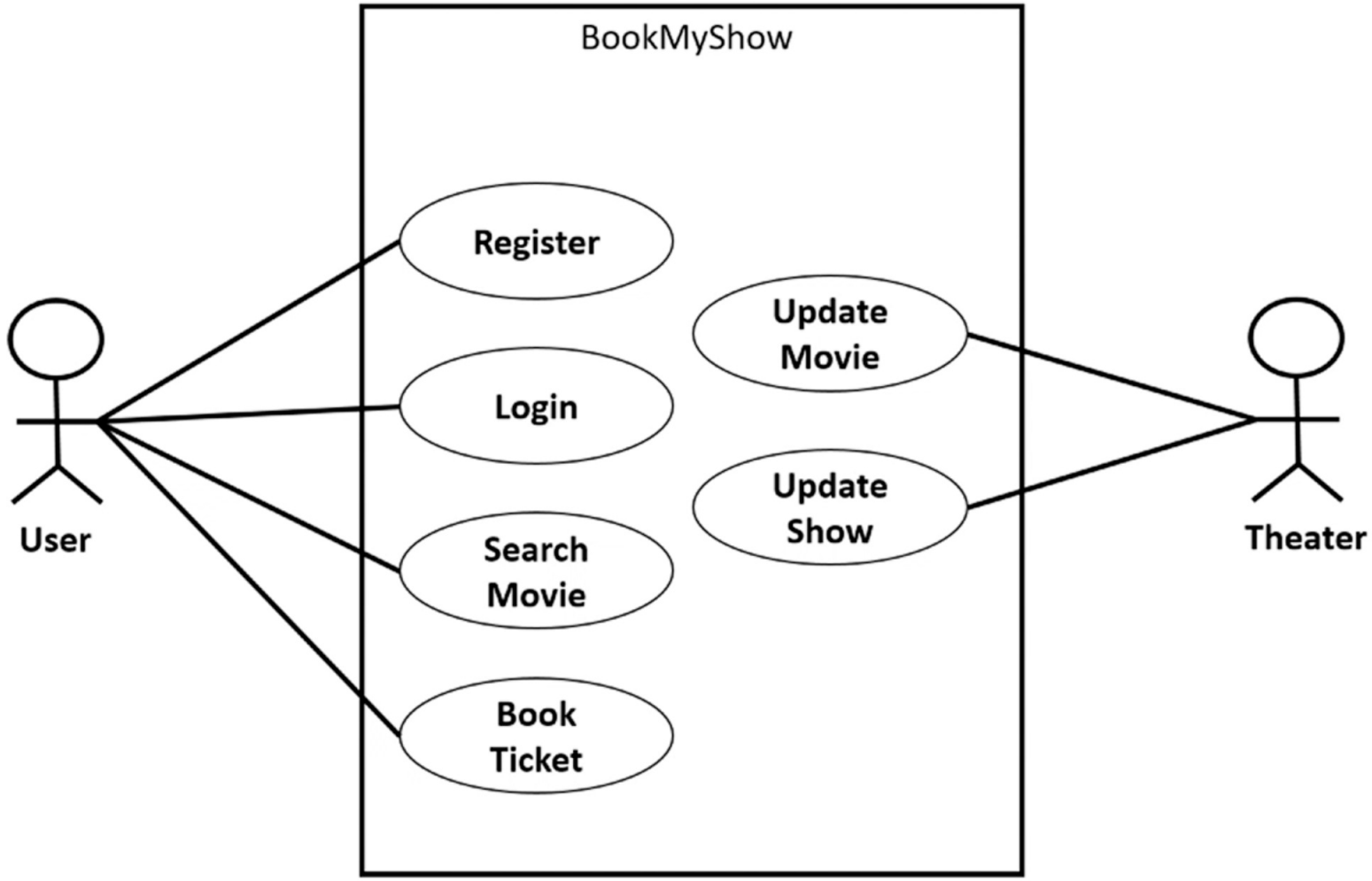
Actor

Organization

Application Name

Other System





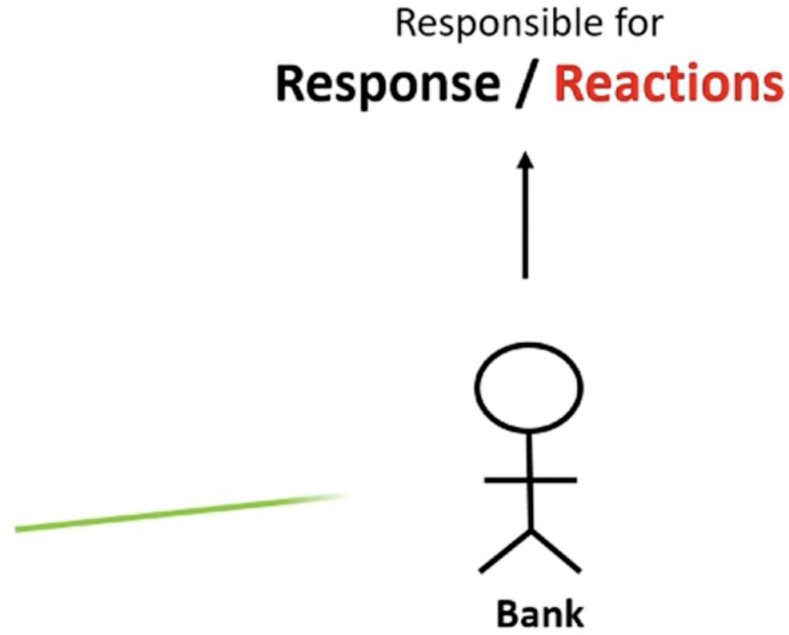
Responsible for

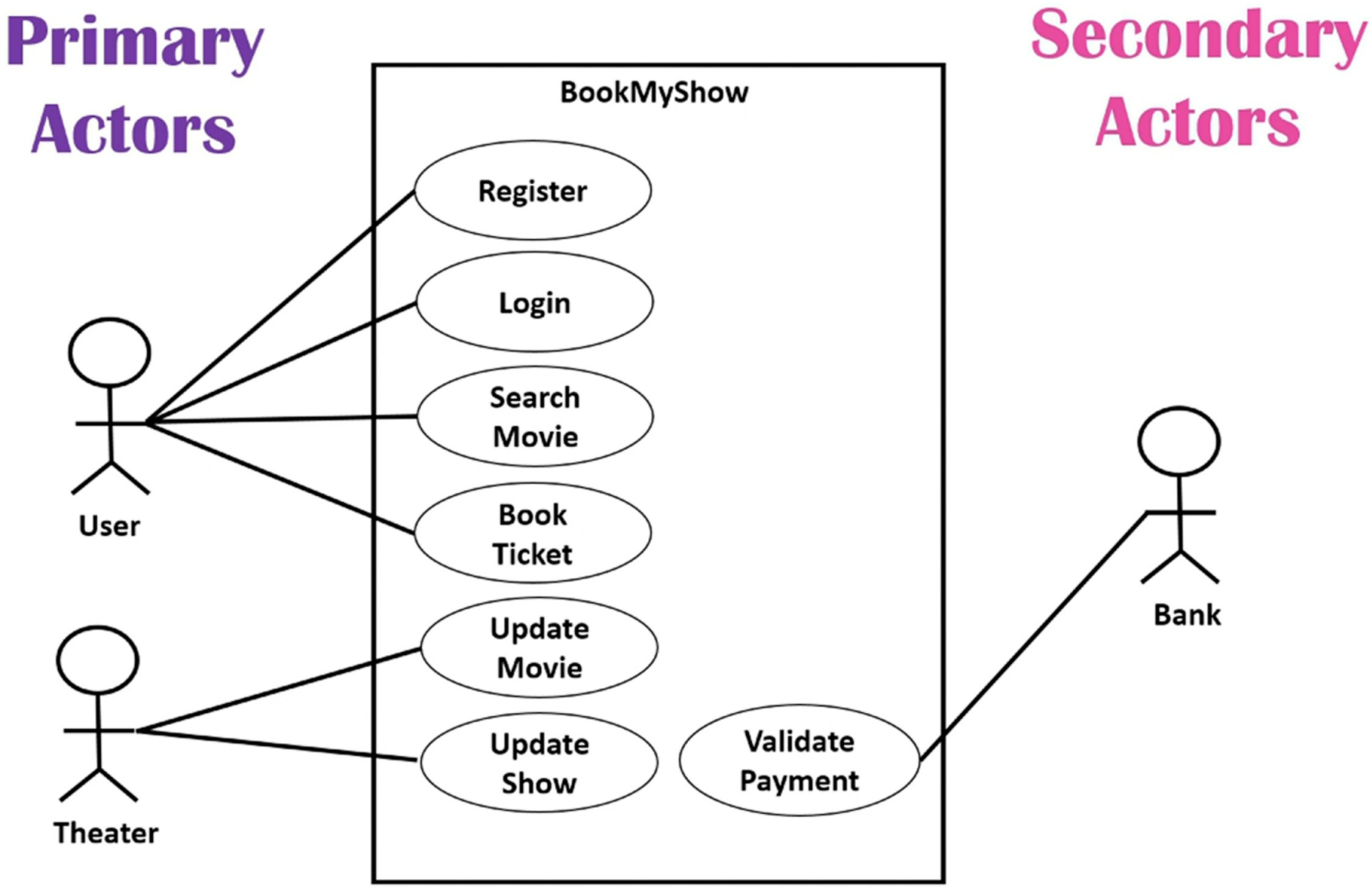
Initiating Action

User

Primary Actors

Secondary Actors





Association