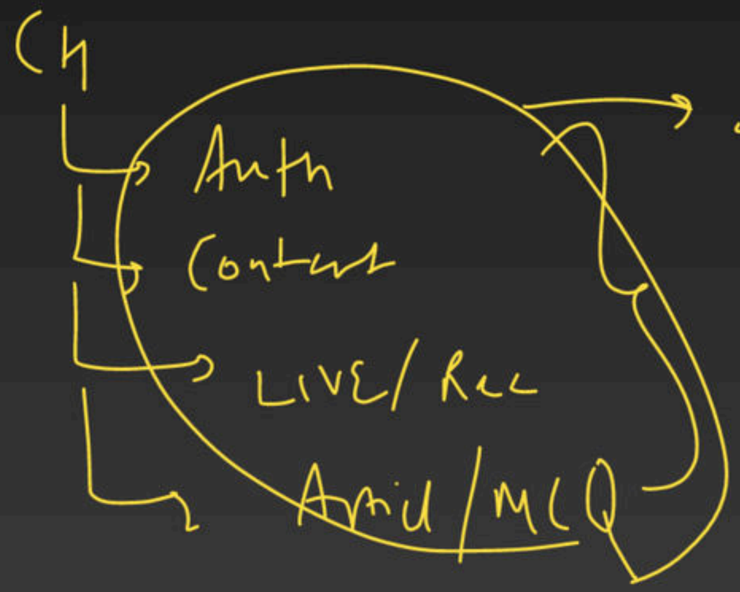
# Intro to UML, Use-case and Class Diagrams

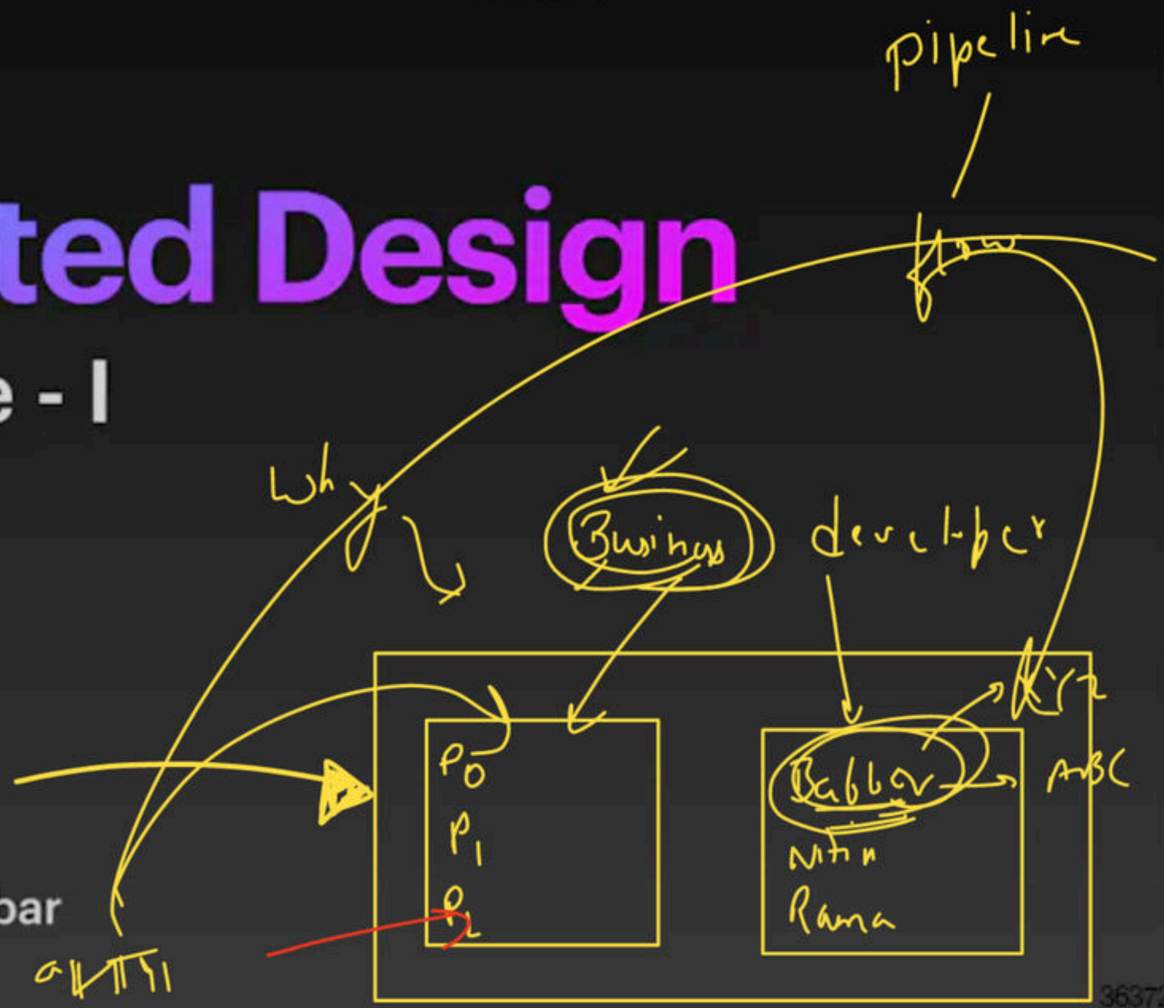
Special class

**Love Babbar** • Mar 9, 2024


# Object Oriented Design

## Module - I

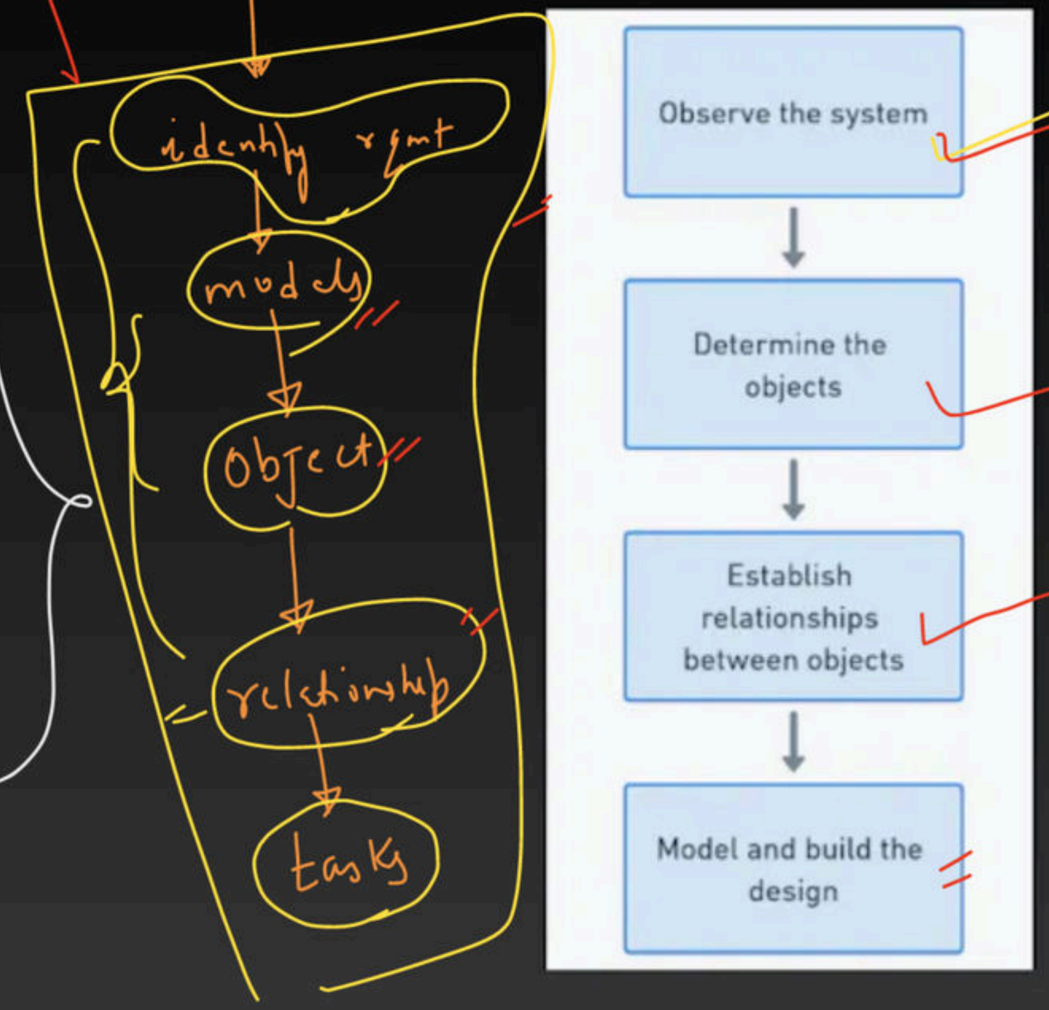



- Love Babbar

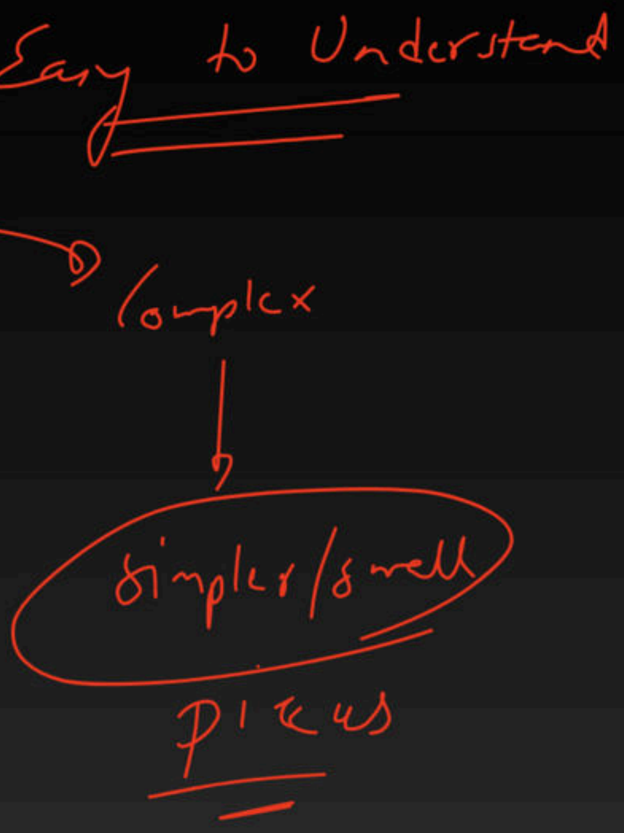
# Object Oriented Analysis & Design

- Object-oriented analysis and design (OOAD) in the process of creating software helps us look at and plan a system using ideas about "objects" - like parts of the software that represent things in the real world. In OOAD, we first check out the whole system we need for a certain issue, then find and name the "objects" in that system. After that, we figure out how these objects are connected and make a plan for the whole system.



# Why OOAD ?

- The object-oriented analysis and design process is highly well-known. The following are a few reasons why it is quite renowned among the community:
  - It's extremely easy to understand, which helps in creating models of complex problems.
  - Concepts such as inheritance help make the data reusable and scalable.
  - It's easily maintainable, which helps identify the issues in the early processes and saves time.



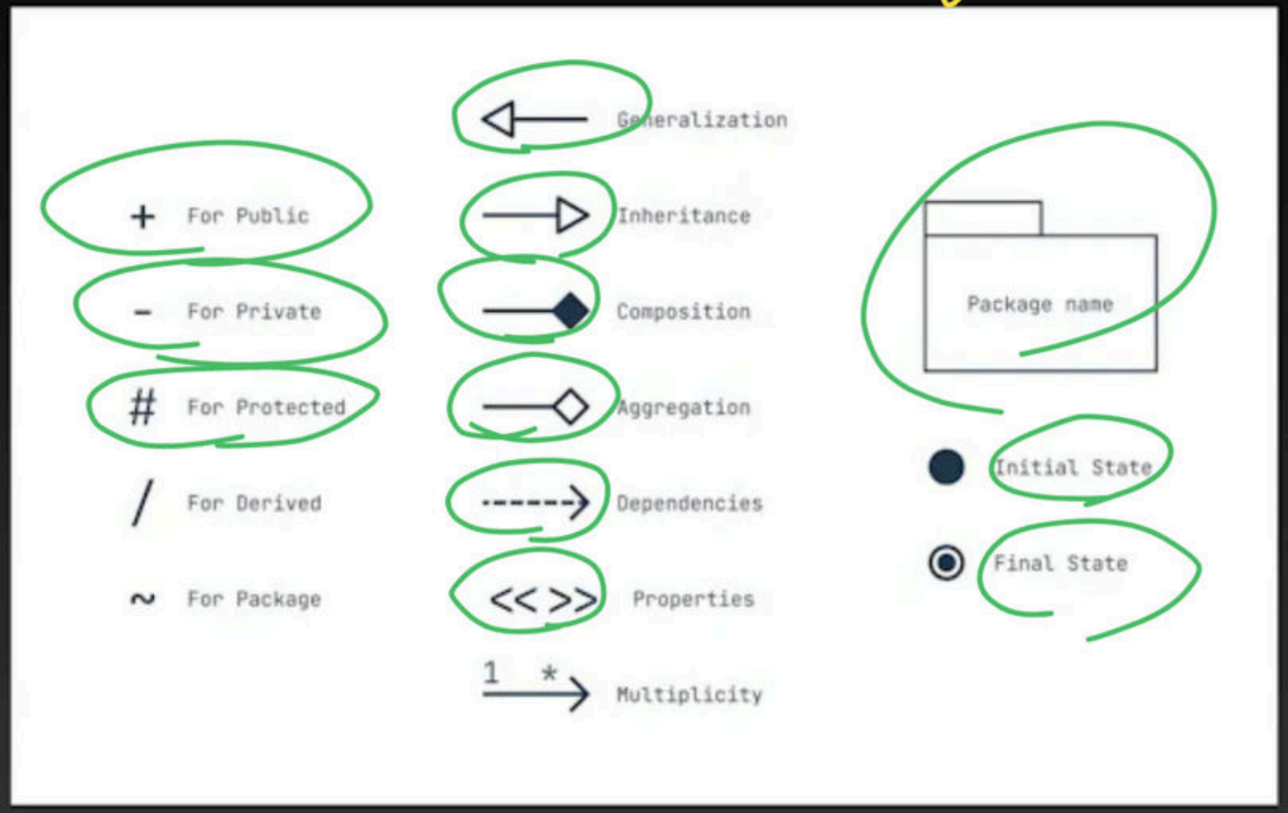
visualisation

# Introduction to UML

Unified Modeling Language

## What is UML ?

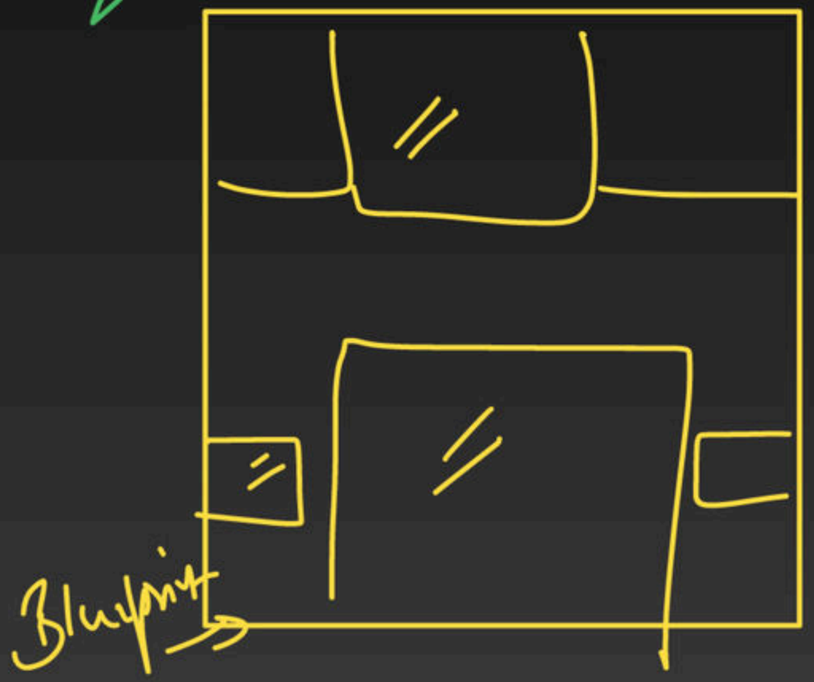
- Unified Modeling Language (UML) as a crucial standard for depicting system designs. UML, though not a programming language, aids in representing the behavioral and structural aspects of a system, thereby assisting software professionals in various stages of software development. It also serves as an effective communication tool among all stakeholders involved in a project, ensuring a clear understanding of the software system's inner mechanics.

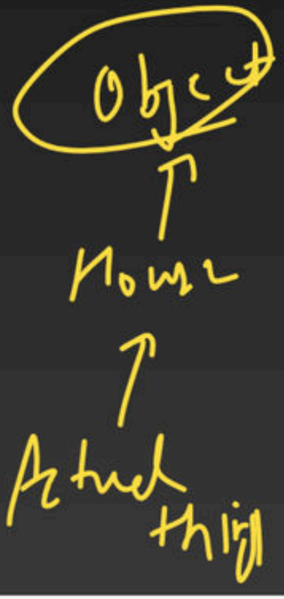


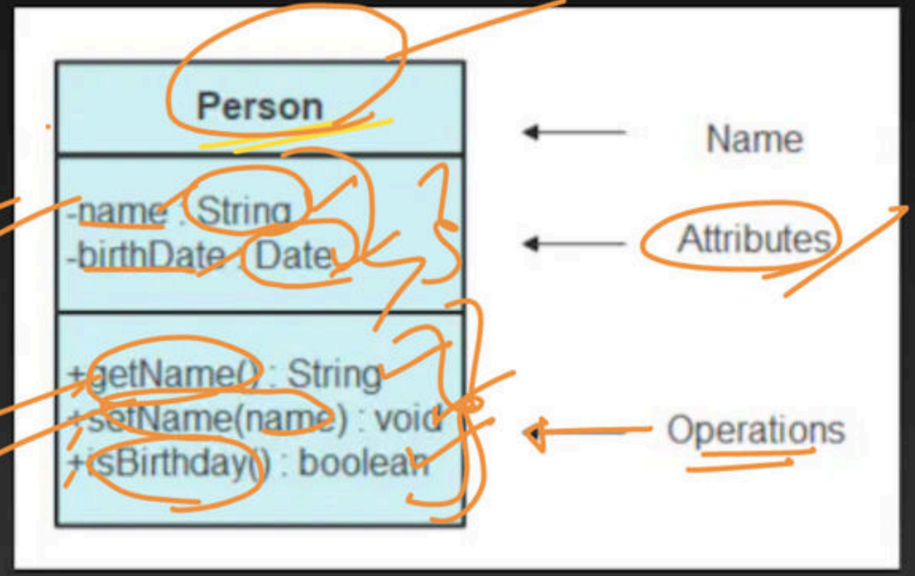
# UML basic notations

UML is composed of three main building blocks: things, relationships, and diagrams. These three exist at the center of UML and play a key role in producing effective and easily understandable models.

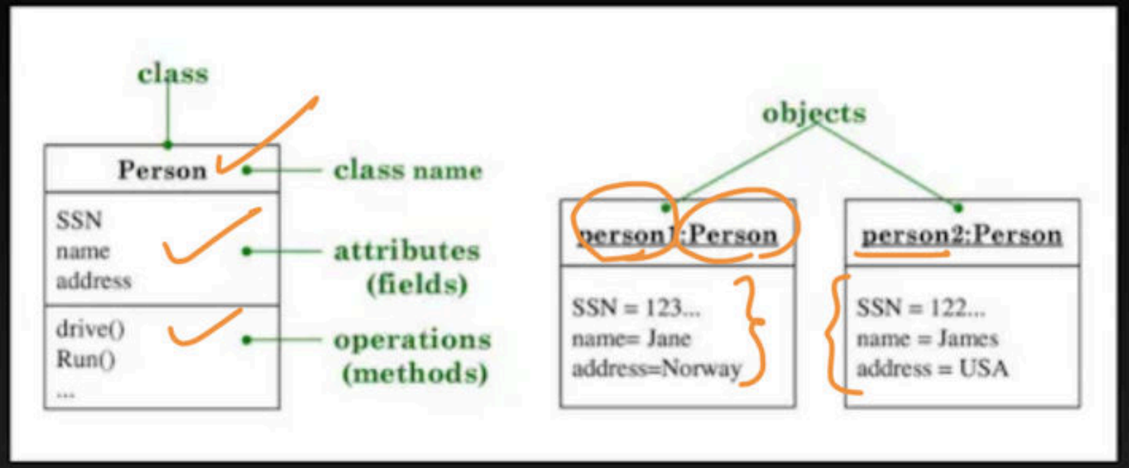
- Class: The notation represents the attributes and methods of an object.



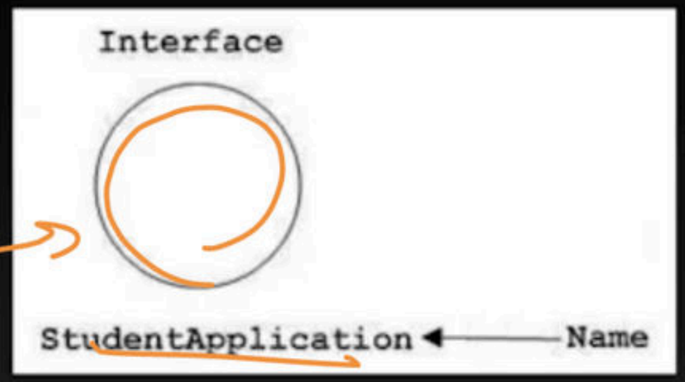



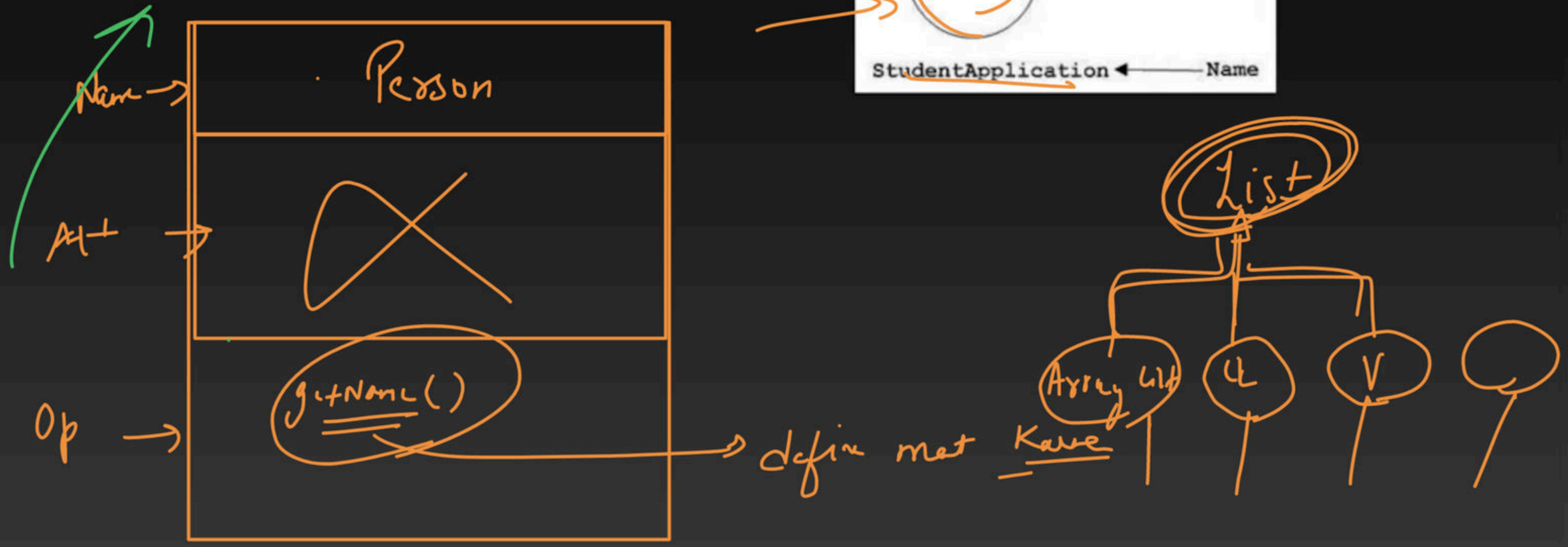



- Object: This notation refers to the instance of a class.

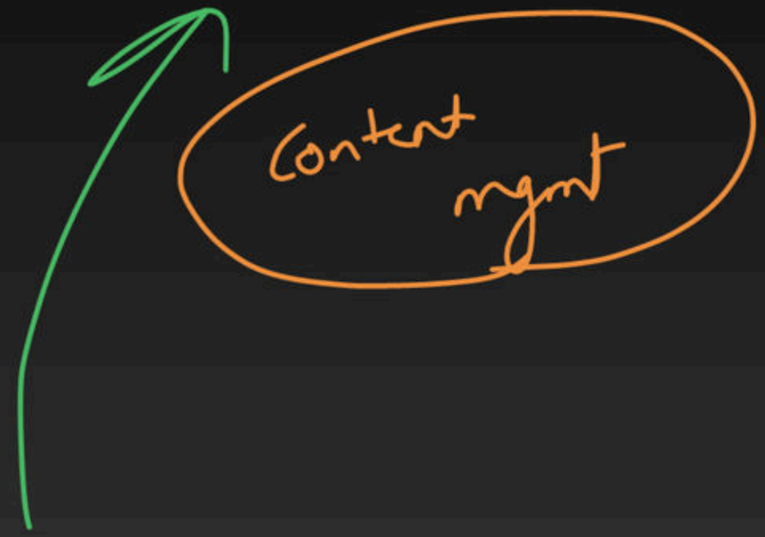


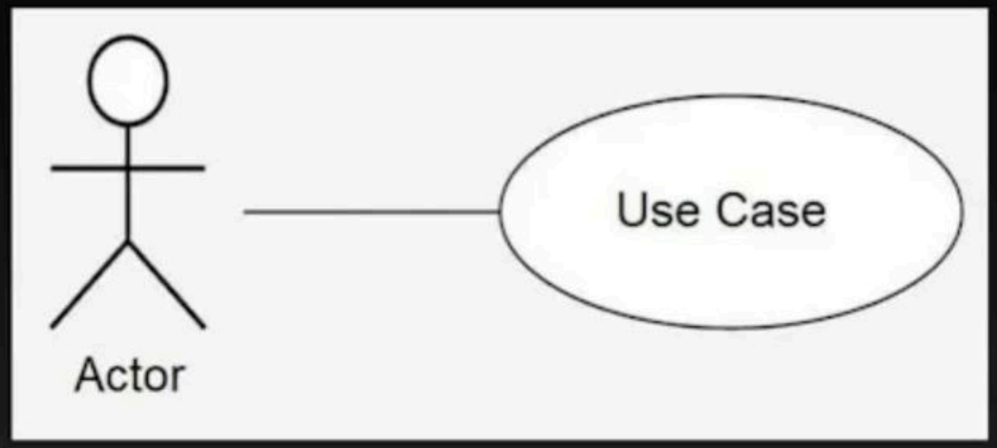
- **Interface**: This notation represents the functionality without its implementation.



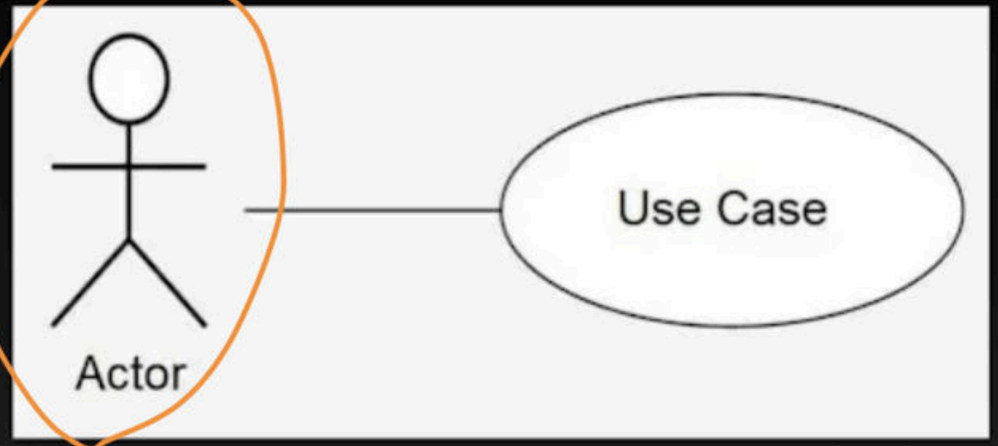



- Use case: This notation describes the users' goals and possible interactions with the system.



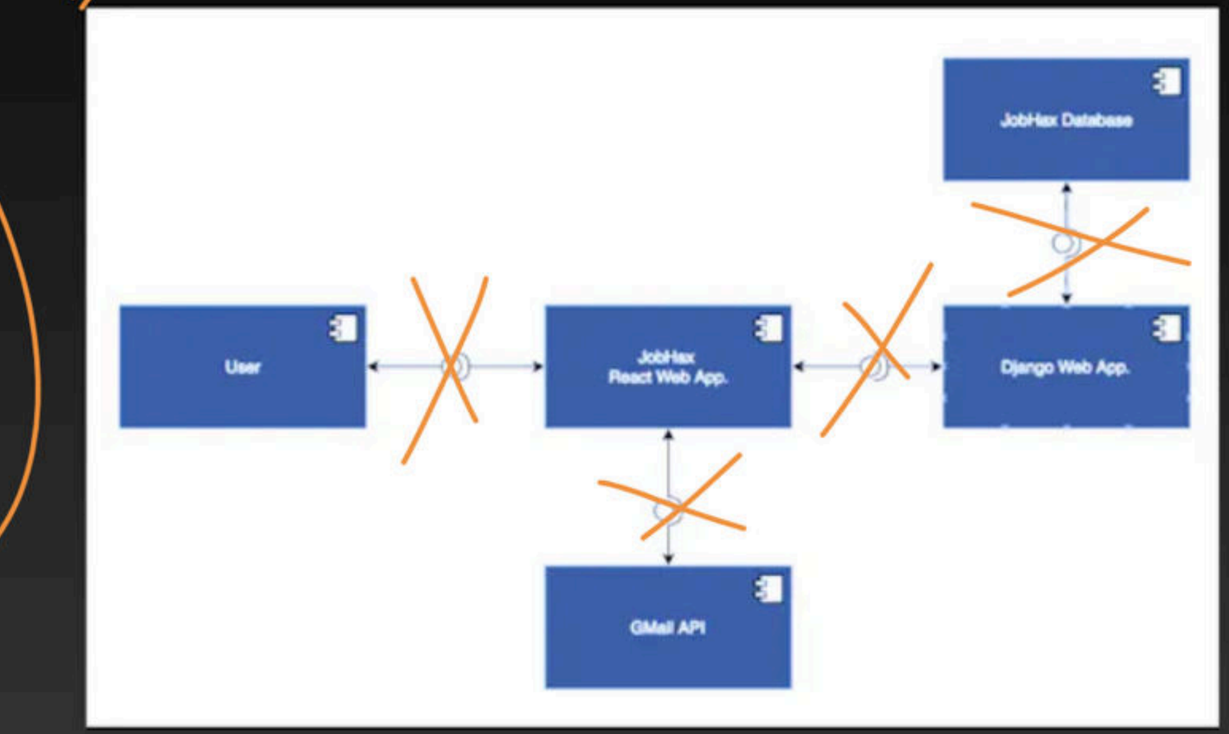
- Actor: This notation represents the entities interacting with the system.



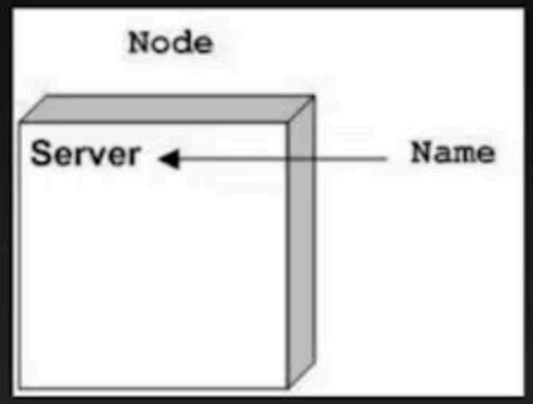

Primary

Secondary → assist Primary actor

- Component: This notation represents a section of the system.

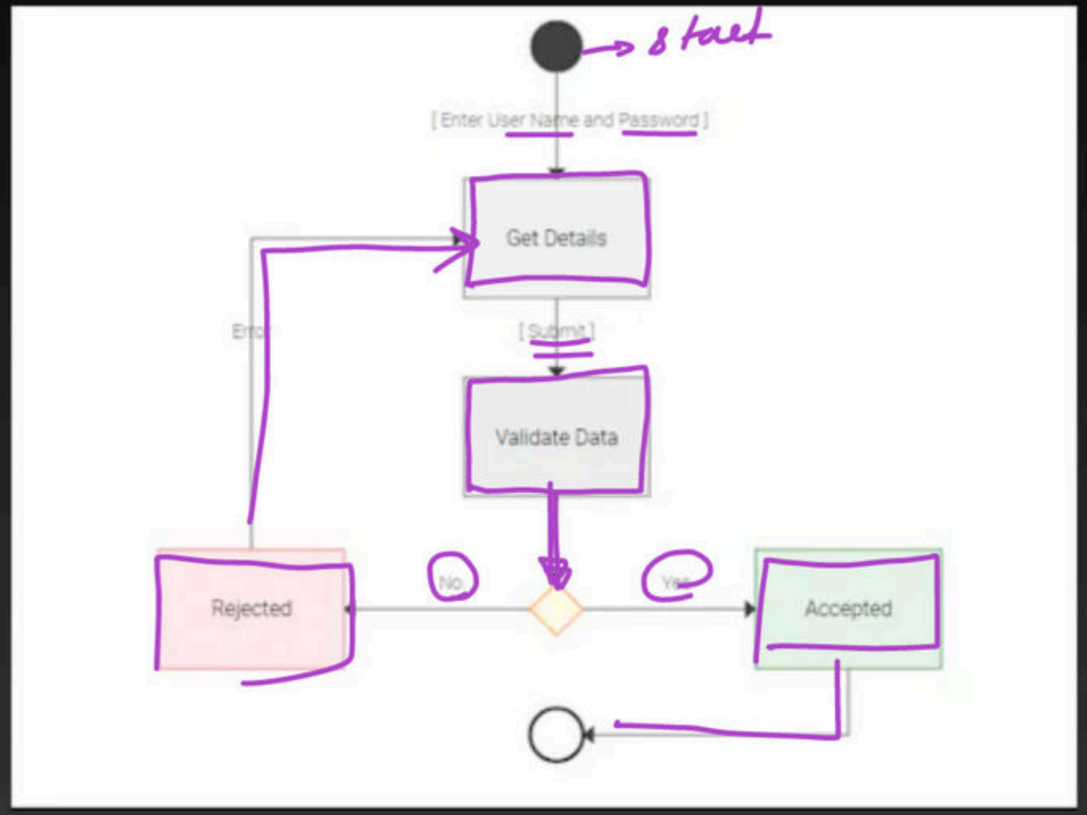


- Node: This notation is similar to the component notation, with the difference being that the node notation refers to the physical aspect of a system, such as a server.



Flowchart

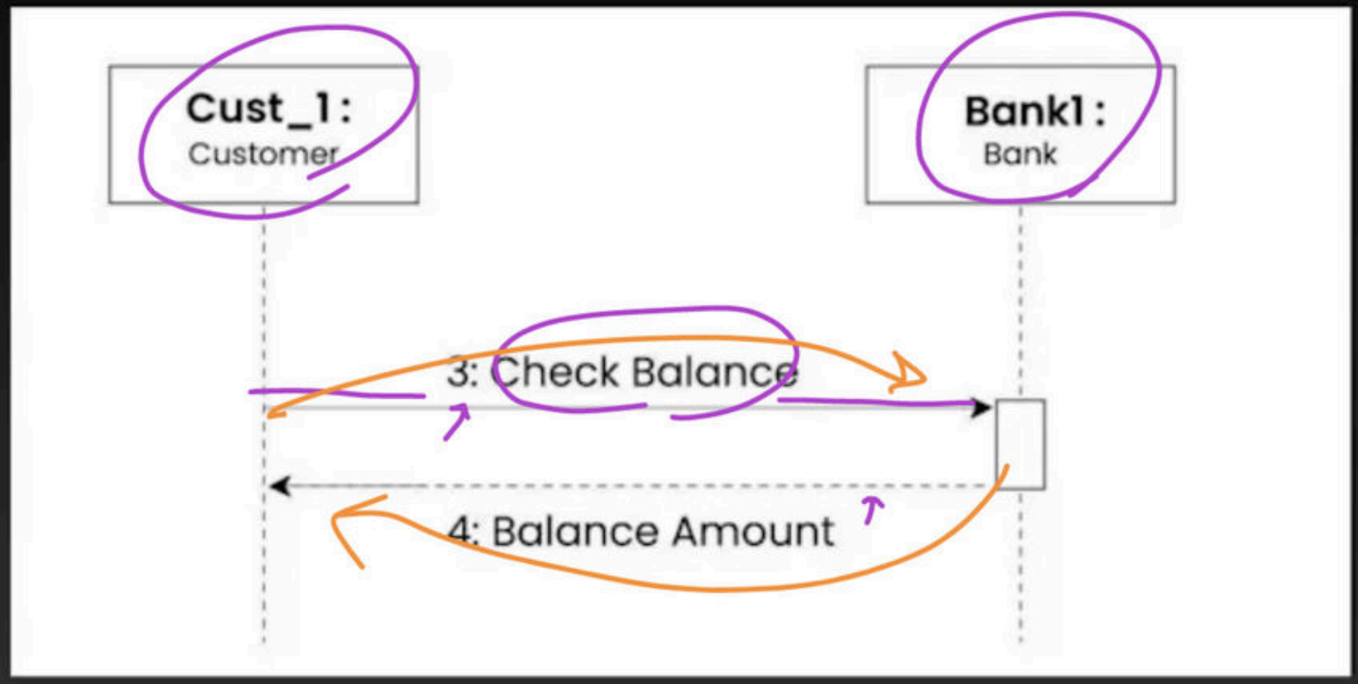
- Activity diagrams: These describe the various interactions performed by different components present in the system.




i/p
print

flowchart

- Interaction diagrams: These diagrams describe the message flow between the different components present in the system.



# Types of UML

## Structural and Behavioural Diagrams

- Most Commonly Used:
  - Class diagram
  - Use-case diagram
  - Sequence diagram
  - Activity diagram

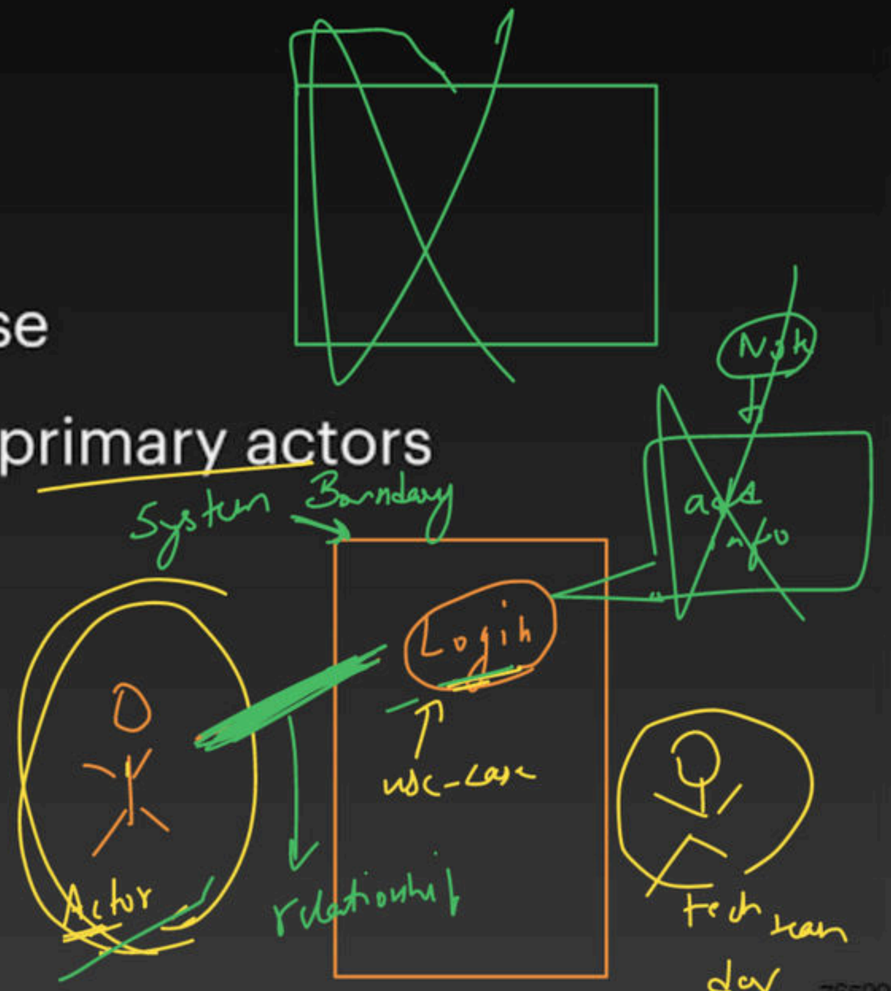
Static
Structure

dynamic
behaviour

# Use-Case Diagram

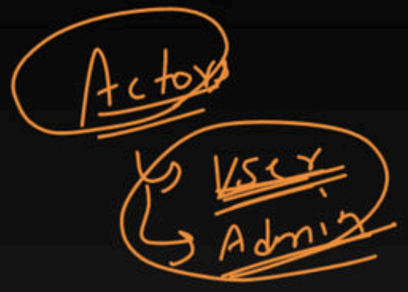
Use case diagram describes the specification of users and their possible interactions with the system. These possible interactions are called use cases.

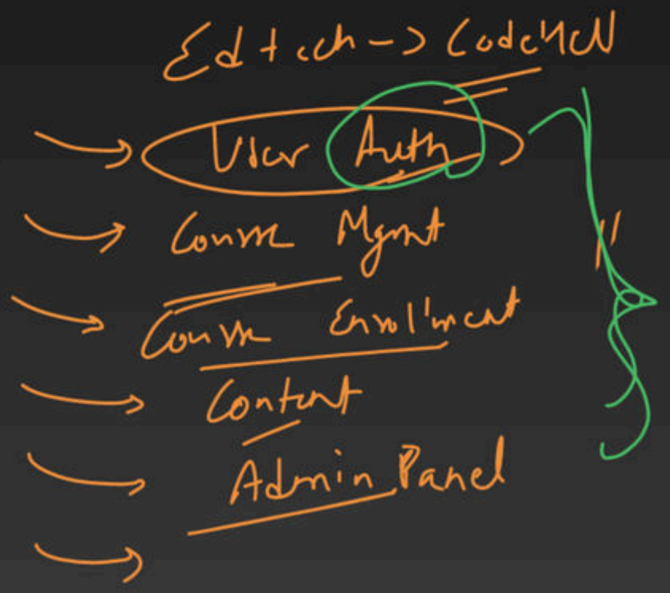
- Components of a use-case diagram:
  - Actor: They interact with systems
    - Primary: responsible for initiating use case
    - Secondary: responsible for assisting the primary actors
  - UseCase: a single function
  - Package: group of different elements
  - Note: used to add additional information

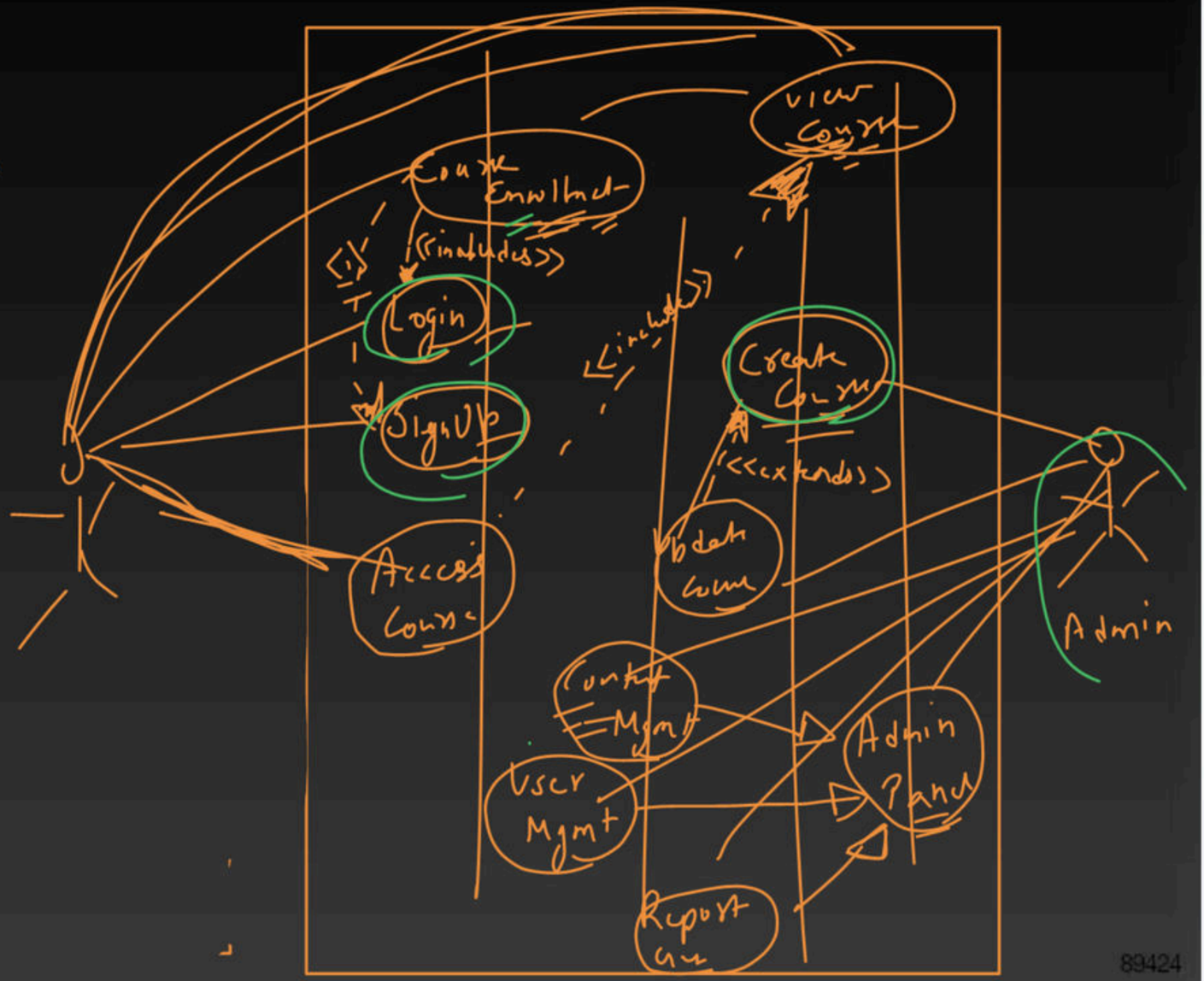


# Relationships in Use-case Diagrams

- Association
- Generalisation
- Include
- Extends



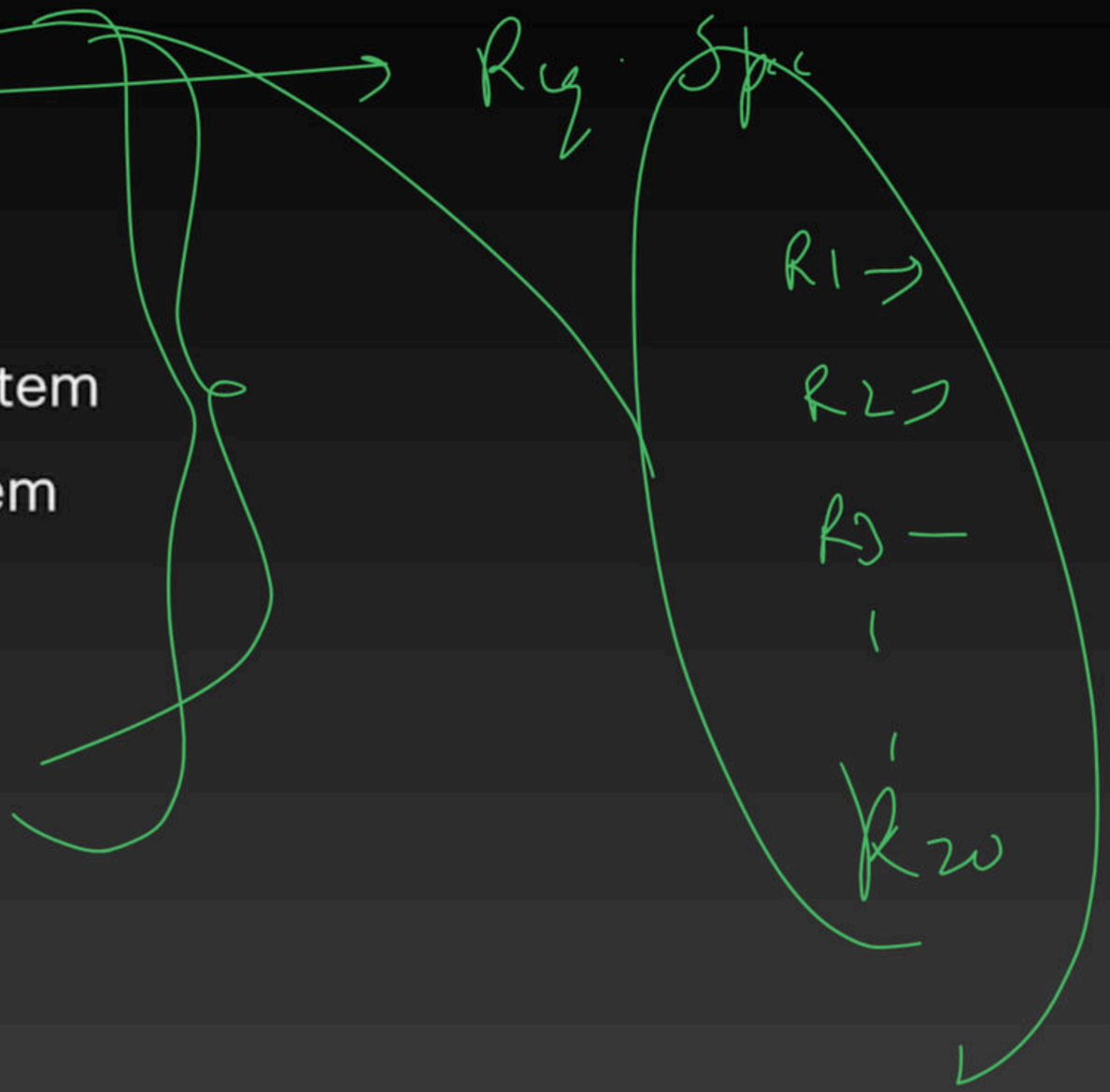



# Benefits of Use-case diagrams:

- It shows what each part of the system does and what it aims to achieve.
- It makes it easy to grasp what the system is supposed to do at a basic level.
- It outlines what the system requires and its setting.
- It describes how the system works from the viewpoint of someone using it.
- It outlines what the system includes and its limits.

# HomeWork:

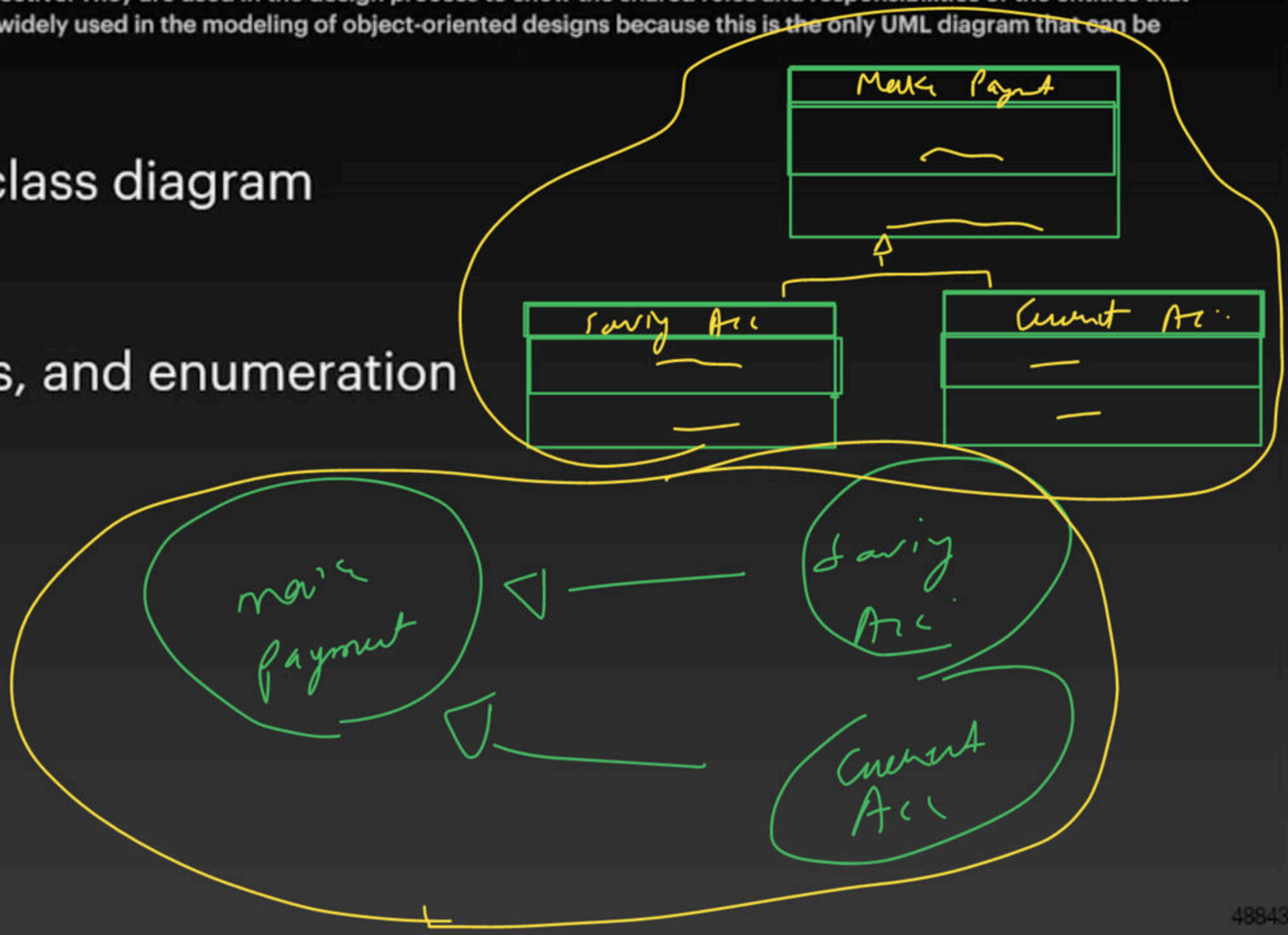
- Design a Parking Lot System
- Design an Elevator System
- Design a Car Rental System
- Design Library Management System
- Design Hotel Management System
- Design ATM
- Design Stack Overflow
- Design LinkedIn



# Class Diagrams:

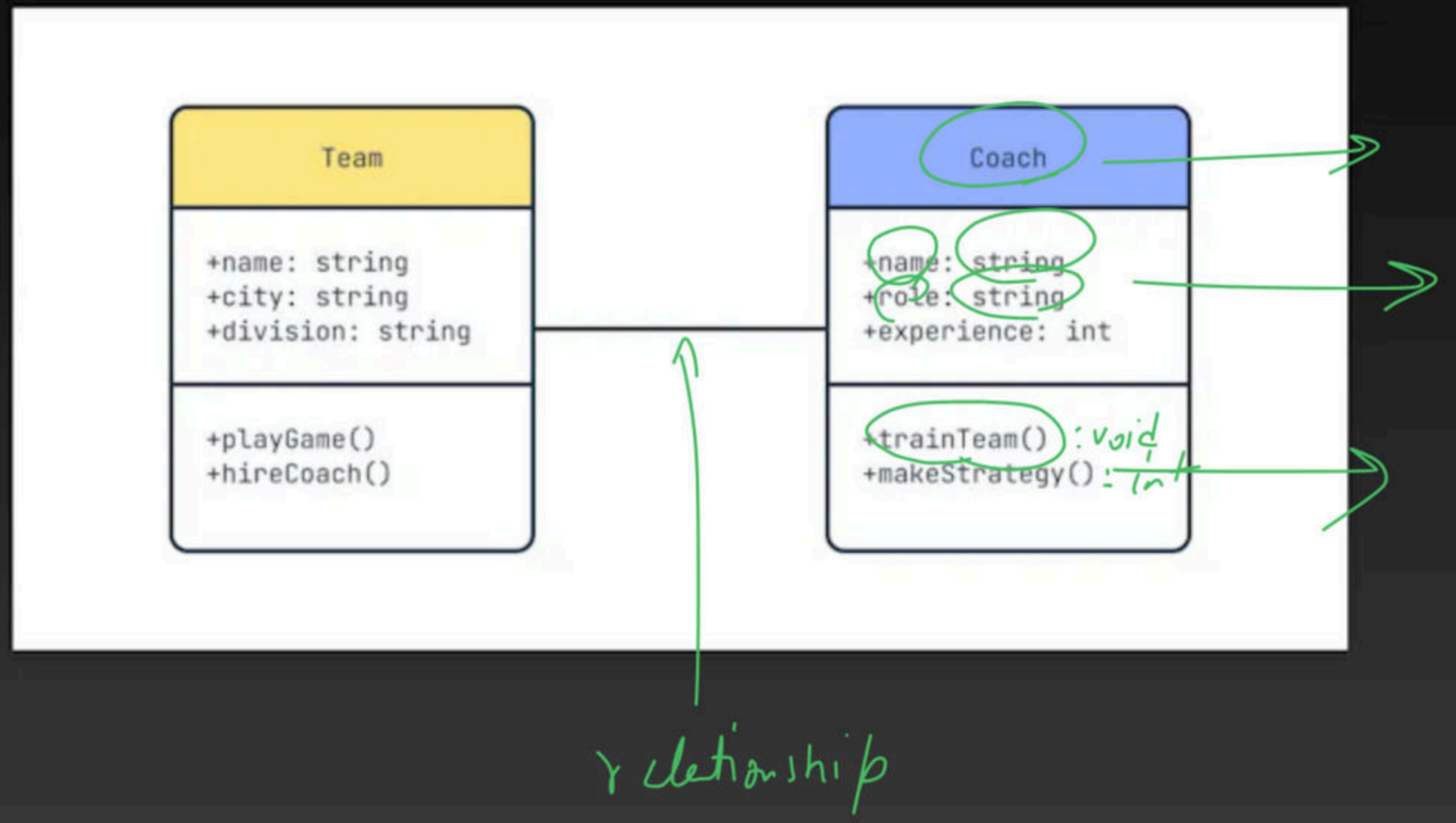
Class diagrams are used to depict the system's static perspective. They are used in the design process to show the shared roles and responsibilities of the entities that produce the behavior of the system. Class diagrams are widely used in the modeling of object-oriented designs because this is the only UML diagram that can be

- Popular notations in the class diagram
  - Class notation
  - Interface, abstract class, and enumeration
  - Access modifiers



# Class Notation

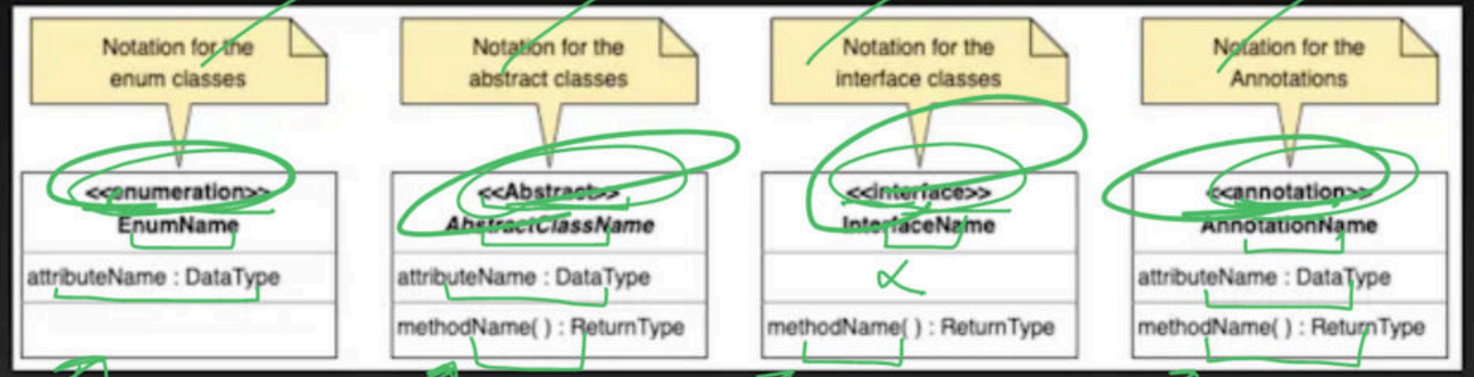
A class looks like a box with three parts. The top part has the class's name. The middle part has a list of features, and the bottom part has a list of actions the class can do. Here's an example using a "Movie" class, showing its features and actions.



# Interface, abstract class, and enumeration

We can declare a class as abstract using abstract keywords. The class name will be printed in italic. We can use the interface, annotation, and enum keywords too. The illustration below shows how to depict these notations in a class diagram.

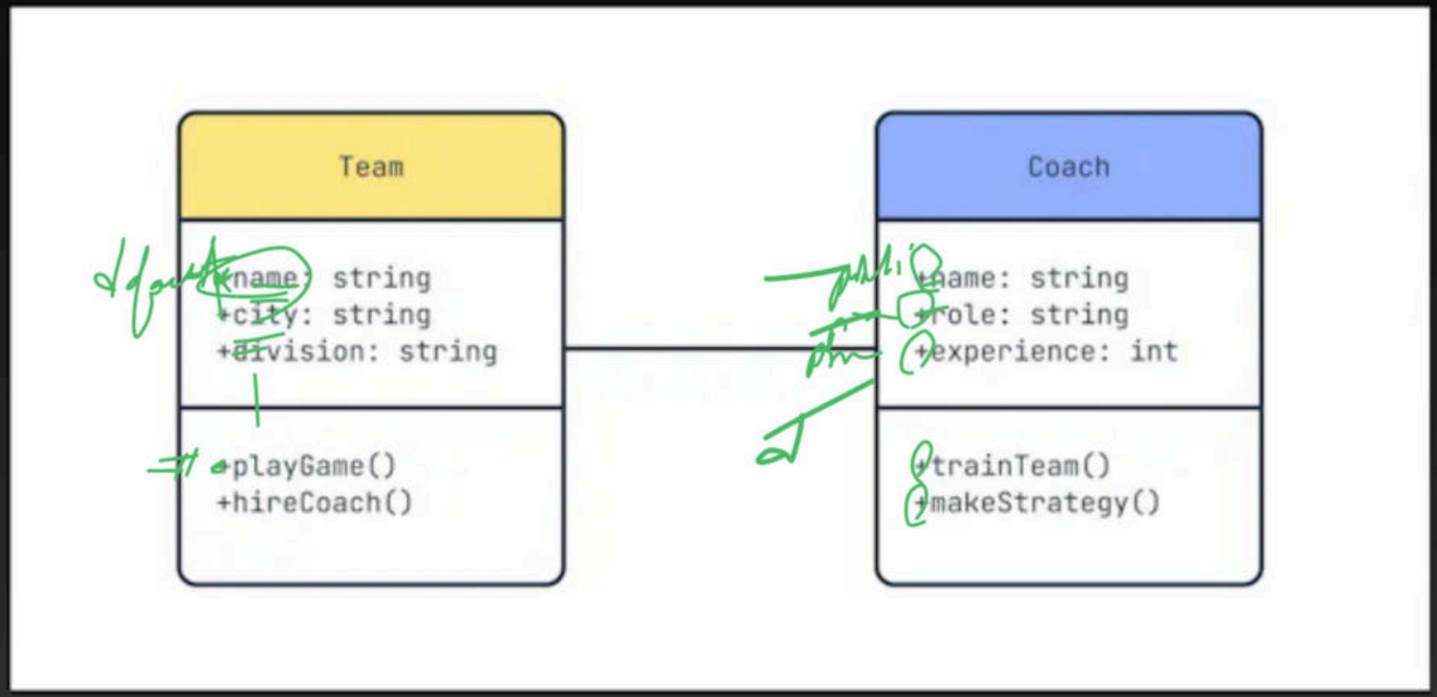
- Enumeration (Enum): An enum is a special data type that enables for a variable to be a set of predefined constants, improving type safety and readability.
- Abstract Classes: Abstract classes are classes that cannot be instantiated on their own and are designed to be extended by subclasses that implement the abstract methods contained within.
- Interfaces: An interface is a contract for a class that specifies what methods the class should implement, without providing the method implementation itself.
- Annotations: Annotations are a form of metadata that provide information about the code but do not change the code itself; they are used to give additional information to the compiler or to be used through reflection at runtime.



# Access Modifiers

- When you create methods or attributes in programming, you can use special symbols to show who can see or use them. Here's what the symbols mean:

- The "+" symbol means everyone can see it. This is called "public."
- The "-" symbol means only the code inside the same class can see it. This is called "private."
- The "#" symbol means only the class it's in and classes that are based on it can see it. This is called "protected."



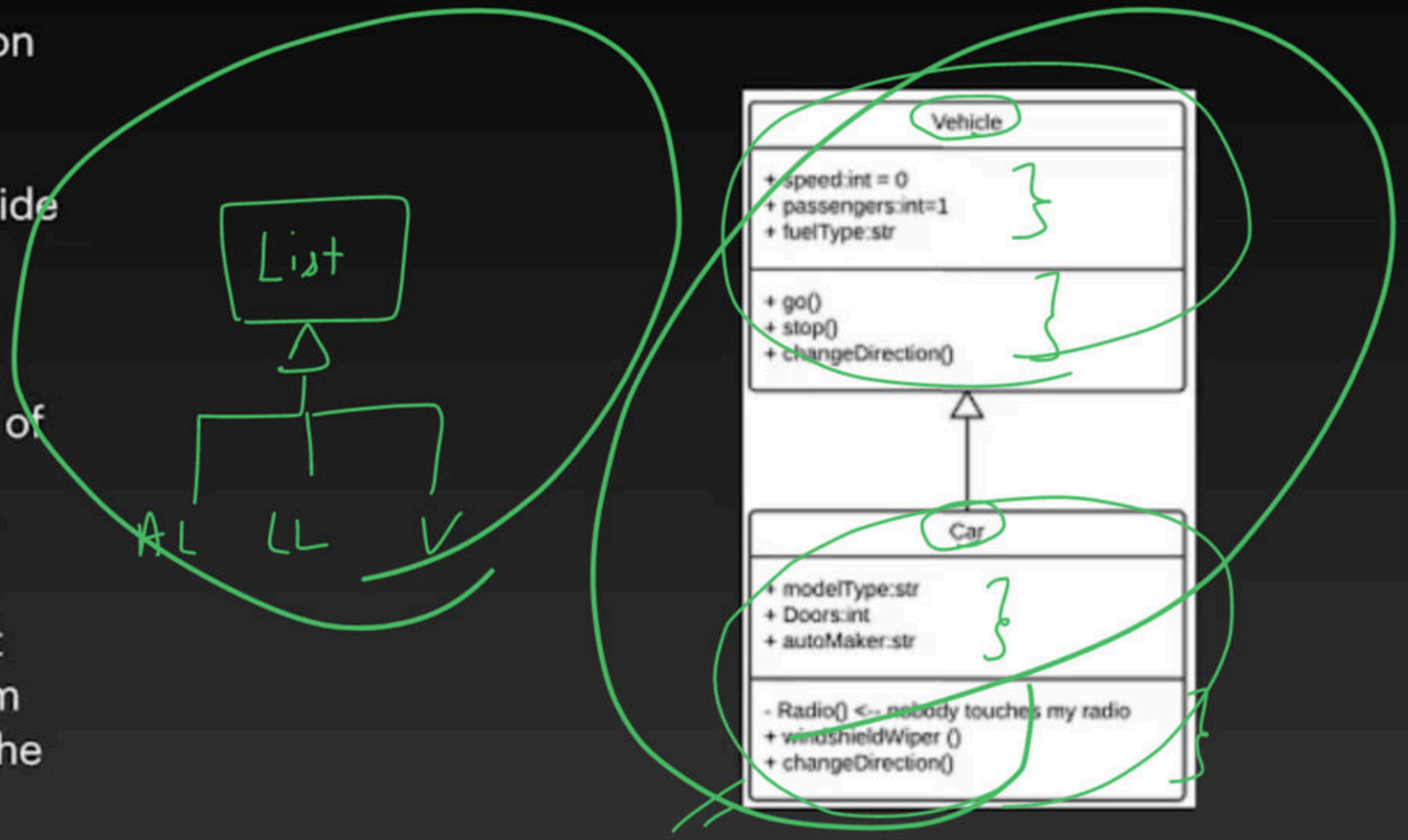
# Association

Association provides a mechanism to communicate one object with another object, or one object provides services to another object. Association represents the relationship between classes.

- The association can be divided into two categories:
  - Class association (Inheritance)
  - Object association: Simple Association, Composition & Aggregation

# Class Association → Inheritance

- Inheritance is like making a new version of something that already exists. Imagine you have a recipe from your family (the original class) and you decide to add your own twist to it (creating a new class). This new recipe (the child class) still keeps all the original ingredients and steps (characteristics of the parent class), but you might add something new or do something a bit differently. In pictures that show how classes are related, we draw a straight line with an empty arrow pointing from the new version (child class) back to the original (parent class) to show this connection.



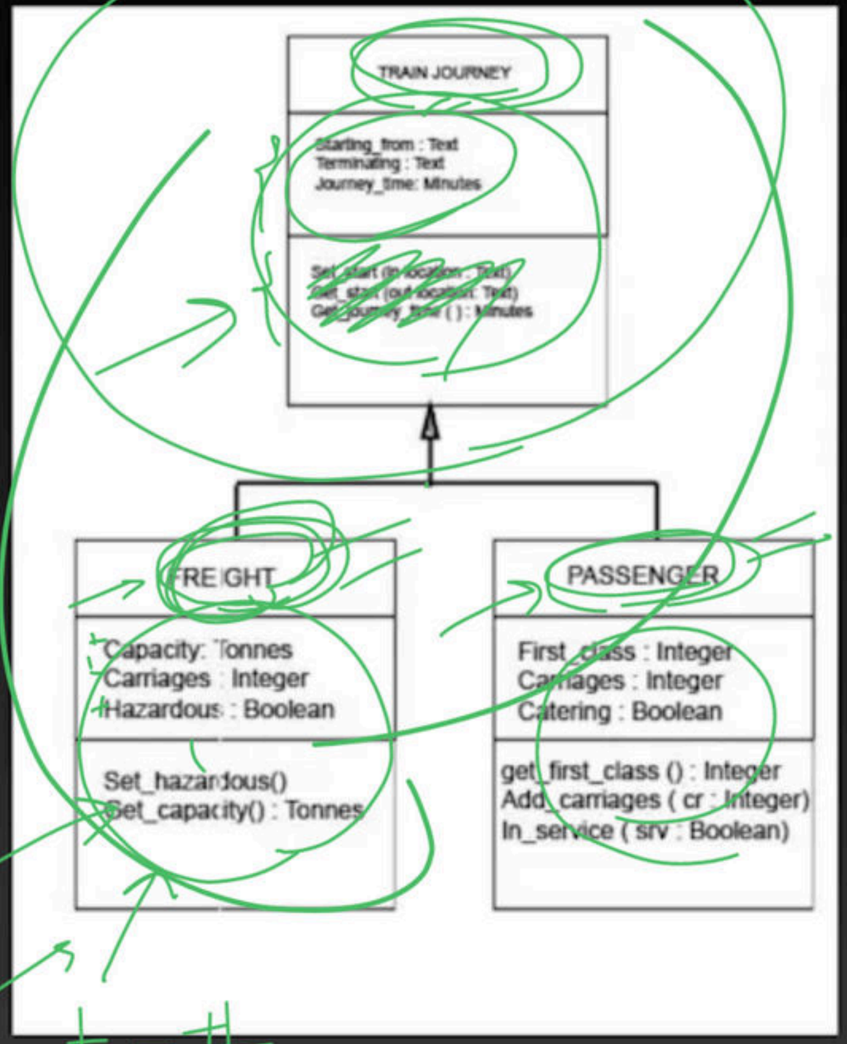
# Example:

The diagram you've provided is a UML class diagram representing an inheritance hierarchy for a train journey system. It shows three classes: `TRAIN JOURNEY`, `FREIGHT`, and `PASSENGER`.

- **TRAIN JOURNEY** is the base class with attributes:
  - `Starting_from`: Text
  - `Terminating`: Text
  - `Journey_time`: Minutes

  It has methods to:
  - `Set_start(in location: Text)`
  - `Get_start(out location: Text)`
  - `Get_Journey_time() : Minutes`
- **FREIGHT** is a subclass of `TRAIN JOURNEY` (indicated by the arrow pointing towards `TRAIN JOURNEY`), suggesting that a `FREIGHT` is a type of `TRAIN JOURNEY`. It has additional attributes:
  - `Capacity`: Tonnes
  - `Carriages`: Integer
  - `Hazardous`: Boolean

  And methods to:
  - `Set_hazardous()`
  - `Get_capacity() : Tonnes`

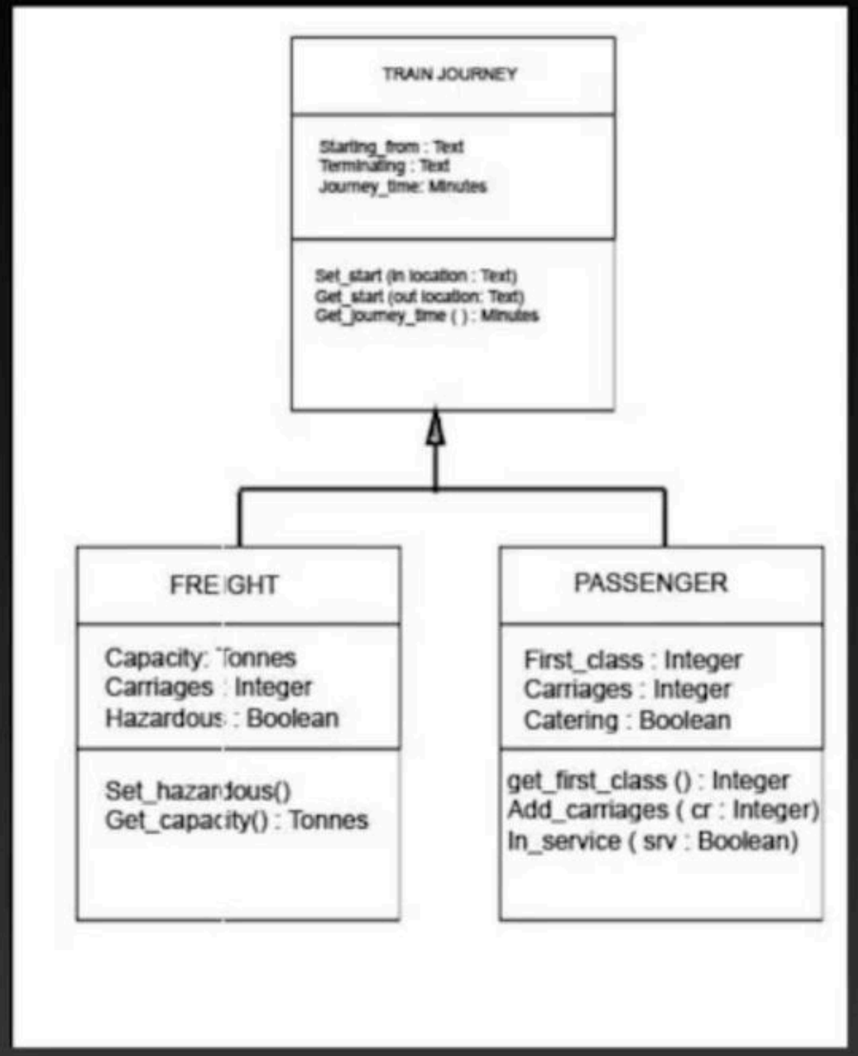


# Example:

- **PASSENGER** is another subclass of `TRAIN JOURNEY`, implying that a `PASSENGER` journey is another type of `TRAIN JOURNEY`. This class includes attributes:
  - `First_class`: Integer
  - `Carriages`: Integer
  - `Catering`: Boolean

  With methods to:
  - `get_first_class() : Integer`
  - `Add_carriages(cr : Integer)`
  - `In_service(srv : Boolean)`

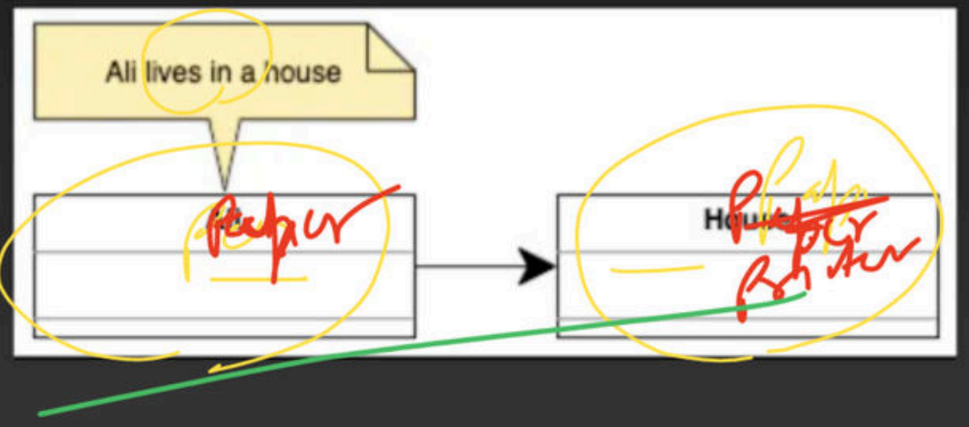
The diagram indicates an object-oriented design where `FREIGHT` and `PASSENGER` trains share common features from `TRAIN JOURNEY` but also have their specific attributes and methods that define their unique properties and behaviors. The `TRAIN JOURNEY` class acts as a general template, while the `FREIGHT` and `PASSENGER` classes specify these templates further for different types of train journeys.

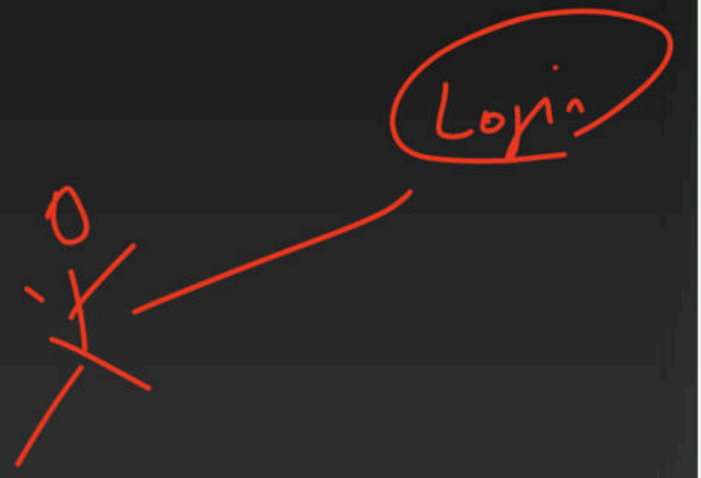


# Object Association

## Simple Association

- The weakest connections between objects are made through simple association. It is achieved through reference, which one object can inherit from another. The following is an example of a simple association:



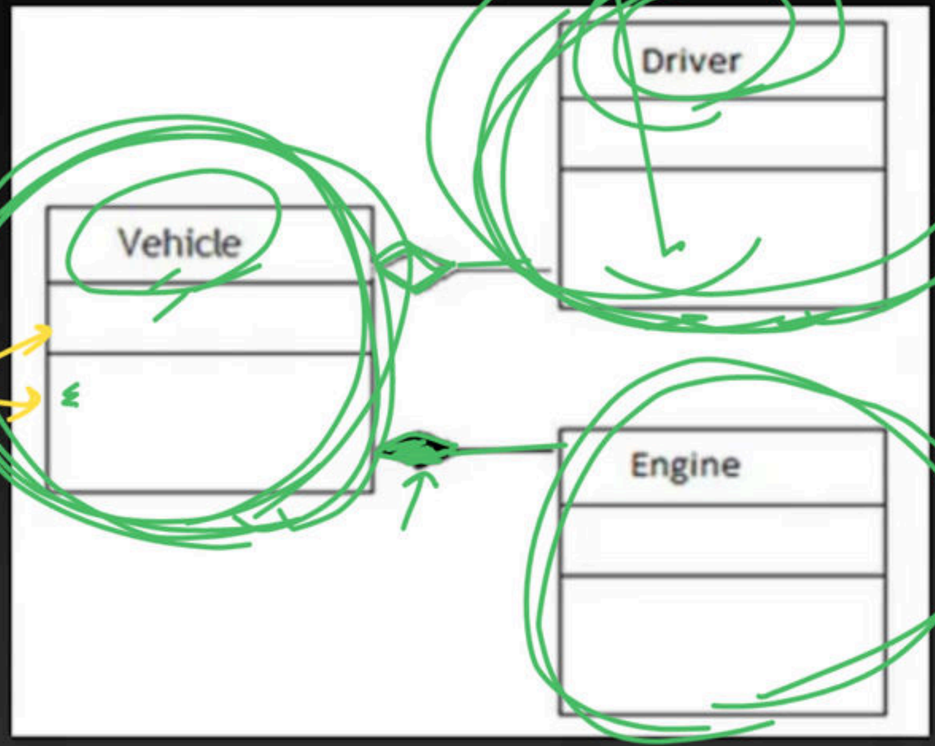
# More Info

- Object association, or simple association, is a concept where two or more objects are connected or related to each other in a straightforward manner. Here are five examples of simple associations:

- Pen and Paper: This is a classic example where the pen is used to write or draw on the paper. The association is based on the function of the pen and the utility of the paper.
- Key and Lock: A key is associated with a lock because the key is designed to open or close the lock. This relationship is direct and is based on the functionality of the key matching the lock mechanism.
- Shoes and Laces: Shoes and laces are associated because laces are used to secure shoes on the feet. The laces go through the eyelets of the shoes and can be tied to adjust the fit of the shoe.
- Charger and Smartphone: The charger is associated with a smartphone because it is used to recharge the smartphone's battery. This is a straightforward relationship where the charger's output matches the smartphone's charging requirements.
- Teapot and Tea Cup: A teapot is associated with a tea cup because the teapot is used to hold and pour tea into the cup. This association is based on the complementary functions of serving (teapot) and receiving (tea cup) the tea.

# Aggregation

- Aggregation is like having a box where you can put things inside. The box is one object, and the things inside are other objects. We show this relationship by drawing a line with a diamond shape at one end that points to the box.
- This relationship is not very strong because:
  - The things inside the box are not actually part of the box itself.
  - The things inside can exist on their own, even if you take them out of the box.




Composition

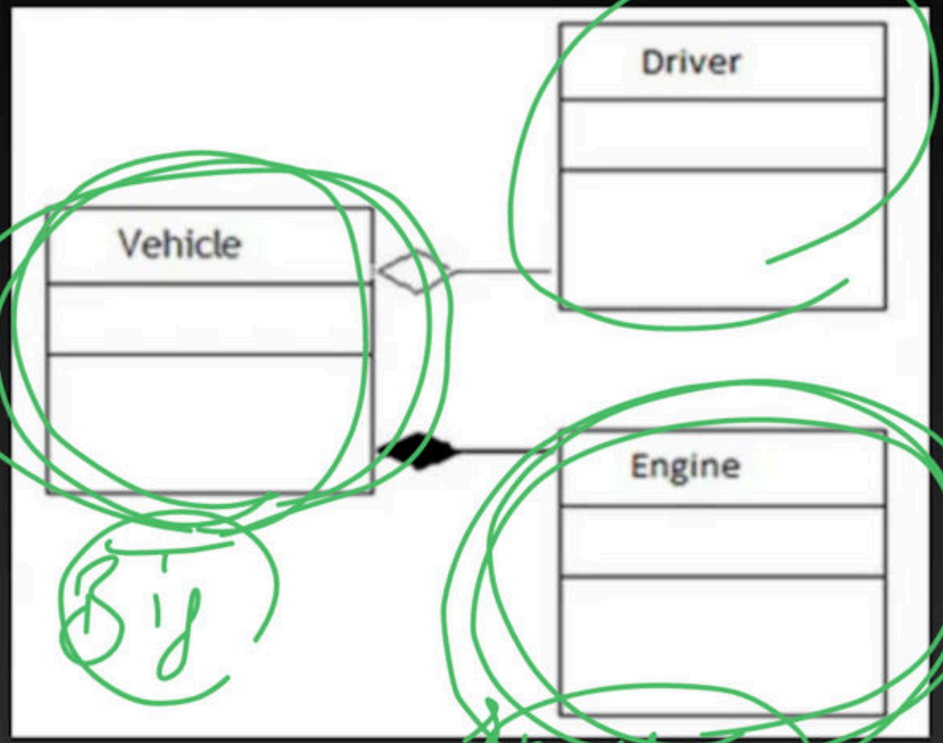
# More Info

- Aggregation in object association represents a "has-a" relationship where one entity (the whole) is made up of other entities (the parts), but the parts can also exist independently of the whole. Here are five examples of aggregation:

- Library and Books: A library contains books, but books can exist outside of the library as well. The library is an aggregation of books and possibly other resources like magazines, DVDs, and so on.
- Computer System and Hardware Components: A computer system is an aggregation of various hardware components like the CPU, RAM, hard drive, motherboard, etc. Each of these components can exist independently of the computer system and can be used in different configurations or systems.
- Car and Wheels: A car has wheels, but wheels are not exclusive to the car and can be used in other applications like motorcycles, carts, etc. The car is an aggregation of wheels along with other parts like the engine, seats, and steering wheel.
- University and Departments: A university is comprised of various departments like Mathematics, English, Engineering, etc. Each department functions within the university structure but can also exist independently as specialized entities or in other university settings.
- Flower Bouquet and Flowers: A flower bouquet is an aggregation of various flowers. Each flower contributes to the overall appearance of the bouquet, but each flower can also exist independently, not bound to the bouquet configuration.

# Composition

- Composition is when a big thing is made up of smaller parts. Think of a chair as the big thing. It's made up of smaller parts like arms, a seat, and legs. We show this by drawing a line with a solid diamond shape at one end, pointing from the big thing to its parts.
- This is a strong connection because:
  - The smaller parts become a part of the big thing.
  - The smaller parts can't be on their own without the big thing.



# More Info

- Composition is a strong form of aggregation where the parts cannot exist independently of the whole; if the whole is destroyed, the parts are destroyed or no longer functional as well. Here are five examples of composition:

- House and Rooms: A house is composed of multiple rooms such as the kitchen, living room, bedrooms, and bathrooms. The rooms are defined by their existence within the structure of the house and do not have an independent existence outside of it.
- Human Body and Organs: The human body is a composition of various organs like the heart, lungs, liver, and kidneys. These organs are integral to the body's functioning, and outside of the body, they lose their function and context.
- Airplane and Wings: An airplane is composed of several parts, including wings. The wings are crucial for the airplane's ability to fly and are designed specifically for a particular airplane model. Without the wings, the airplane cannot function as intended, and the wings themselves have no purpose without the airplane.
- Book and Pages: A book is composed of pages. The pages are bound together in a specific order and cannot function as a book if they are separated from the binding. The integrity of the book as an object is dependent on the pages being combined in a fixed arrangement.