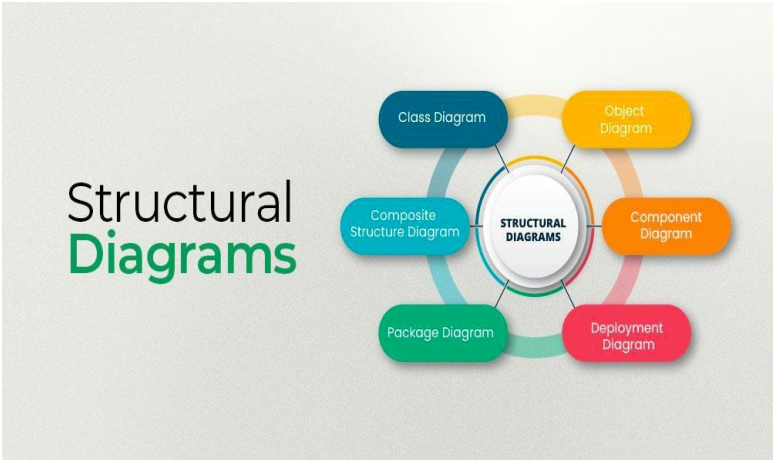
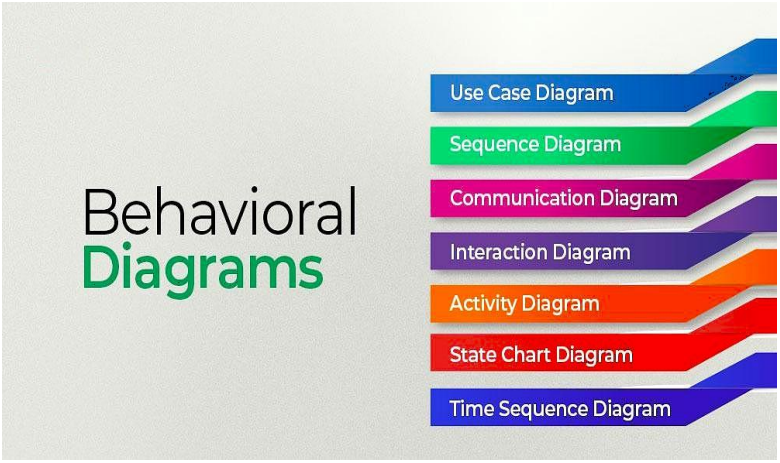
Unified Modelling Language (UML) diagram:

- A ***Unified Modelling Language*** (UML) ***diagram*** is a ***visual representation*** of a ***system***, showing its ***structure***, ***behaviour***, and ***relationships***.
- It's not a programming language itself but a ***standardized way*** for software engineers to ***visualize*** and ***communicate*** the ***design*** and ***implementation*** of ***software systems***.
- UML diagrams help in understanding, ***modelling***, and ***documenting complex software systems***.
- It divided into ***two*** types,
  - ***Structural*** : Focuses on the ***static aspects*** of a ***system***, like its ***components*** and ***relationships***.
  - ***Behavioural*** : Shows the ***dynamic interactions*** and ***changes*** within a ***system***.

Types of Structural UML diagram



Types of Behavioural UML diagram



Class Diagram:

- A ***class diagram*** is a ***static structural*** diagram within the ***UML*** that ***visually represents*** the ***structure*** of a ***system*** by showing its ***classes***, their ***attributes***, ***operations***, and ***relationships*** among objects.
- It's a fundamental tool in object-oriented design, used to ***model classes*** and their ***relationships.***
- We have to express only two things,
  1) Class structure: Refers to the ***overall organization*** and ***arrangement*** of classes within a system, including their ***attributes***, ***operations***, and ***relationships***.
  2) Association / Connection: Describes a ***relationship between two classes***, specifying ***how they are connected*** and ***how they interact***. It can be a ***one-to-one***, ***one-to-many***, or ***many-to-many*** relationship.

UML representation of Access modifiers:

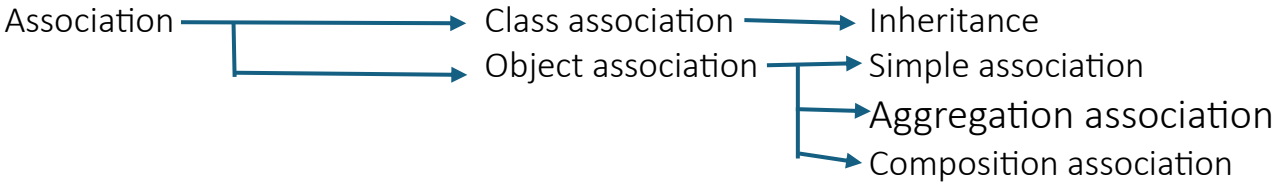
Access Modifier	Within the class	Child class	Outside the class	UML representation
public	✅	✅	✅	+
private	✅	✅	❌	−
protected	✅	❌	❌	#

Class structure sample code:

```
abstract class Students {
    // Variables
    protected Long id;
    protected String name;
    protected String dept;
    private Float collegeFee;

    // Methods
    public abstract void changeDept();

    public abstract Integer remFee();

    public abstract String getName();
}
```

<<abstract>>

Student
(#) id: Long (#) name: String (#) dept: String (-) collegeFee: Float
(+) changeDept: void (+) remFee: Integer (+) getName: String



Inheritance: *Inheritance* allows a *class* (the *subclass*) to inherit *properties* and *behaviours* from *another class* (the superclass), establishing an "*is-a*" relationship (e.g., a car *is a* vehicle).

UML representation: —▷

Simple Association:

- A *general relationship* between *objects*, often described as "*A uses B*". It represents *a connection between objects*, but *doesn't define ownership* or *lifecycle dependency*.
- For example, a *doctor* and a *patient* have an *association* as they interact, but their *lives* are *independent*.
- UML representation: —>

Aggregation:

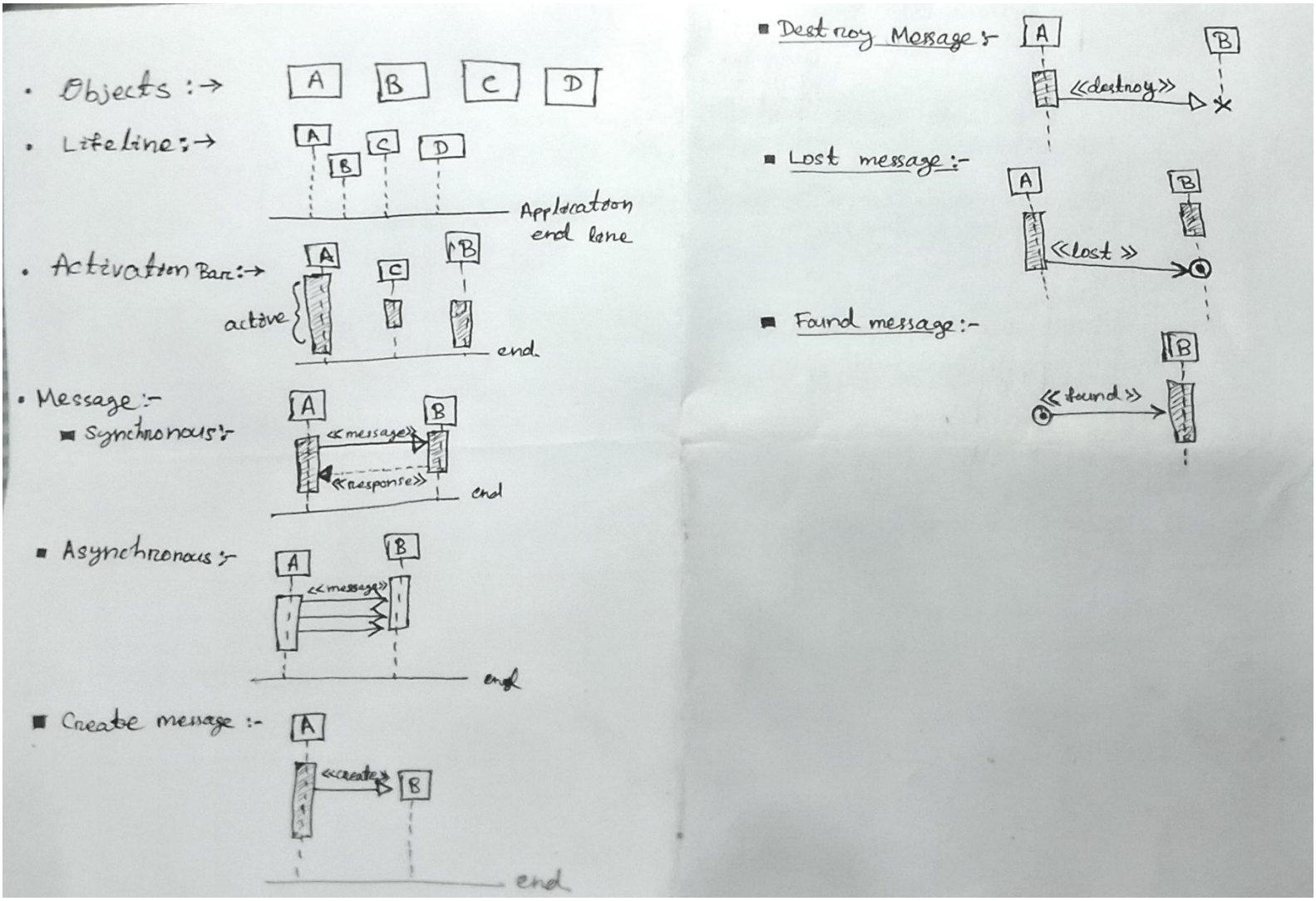
- A "*part-of*" relationship where *one object* (the whole) is *composed* of *other objects* (the parts). The parts can *exist independently of the whole*, even if they are *part* of it *at a given time*.
- A *car* and its *wheels* have an *aggregation relationship*; the wheels can *exist independently of the car.*
- UML representation: —◇

Composition:

- A *strong form* of *aggregation* where *the parts are integral to the whole* and *cannot exist independently*. The parts are essentially owned by the whole, and *if the whole is deleted*, the *parts are also deleted*.
- A *window* and its *glass pane* have a *composition relationship*; the *glass cannot exist without* the *window*.
- UML representation: —◆

Sequence Diagrams:

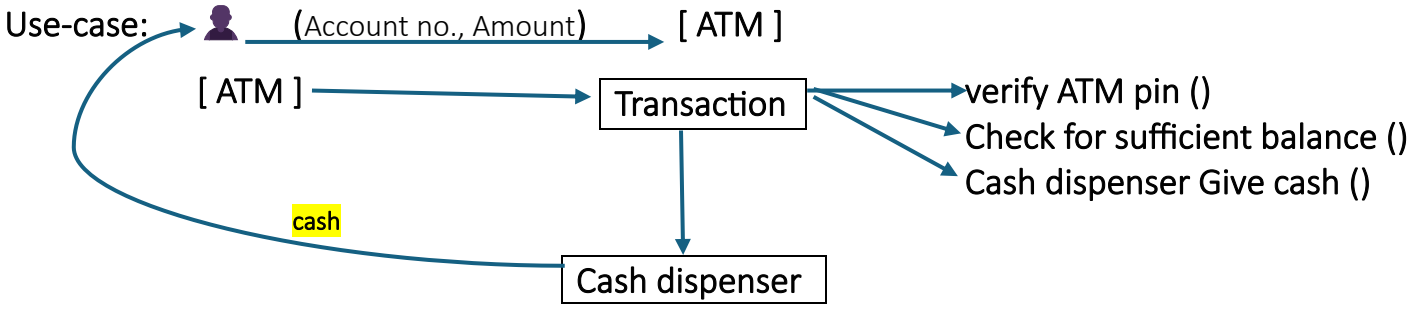
A ***sequence diagram***, also known as a ***timeline diagram*** or ***interaction diagram***, is a ***graphical representation*** that shows the ***interaction between objects in a system over time***. It visually depicts the ***sequence*** of ***messages exchanged between objects***, highlighting the ***flow of actions*** and ***the order in which they occur***.



How to Draw?

Let suppose there is a user went to ATM with account number and amount to withdraw. Then the question is the role of ATM, here ATM simply creates a transaction and give cash to the user. Let's draw a sequence diagram for this...

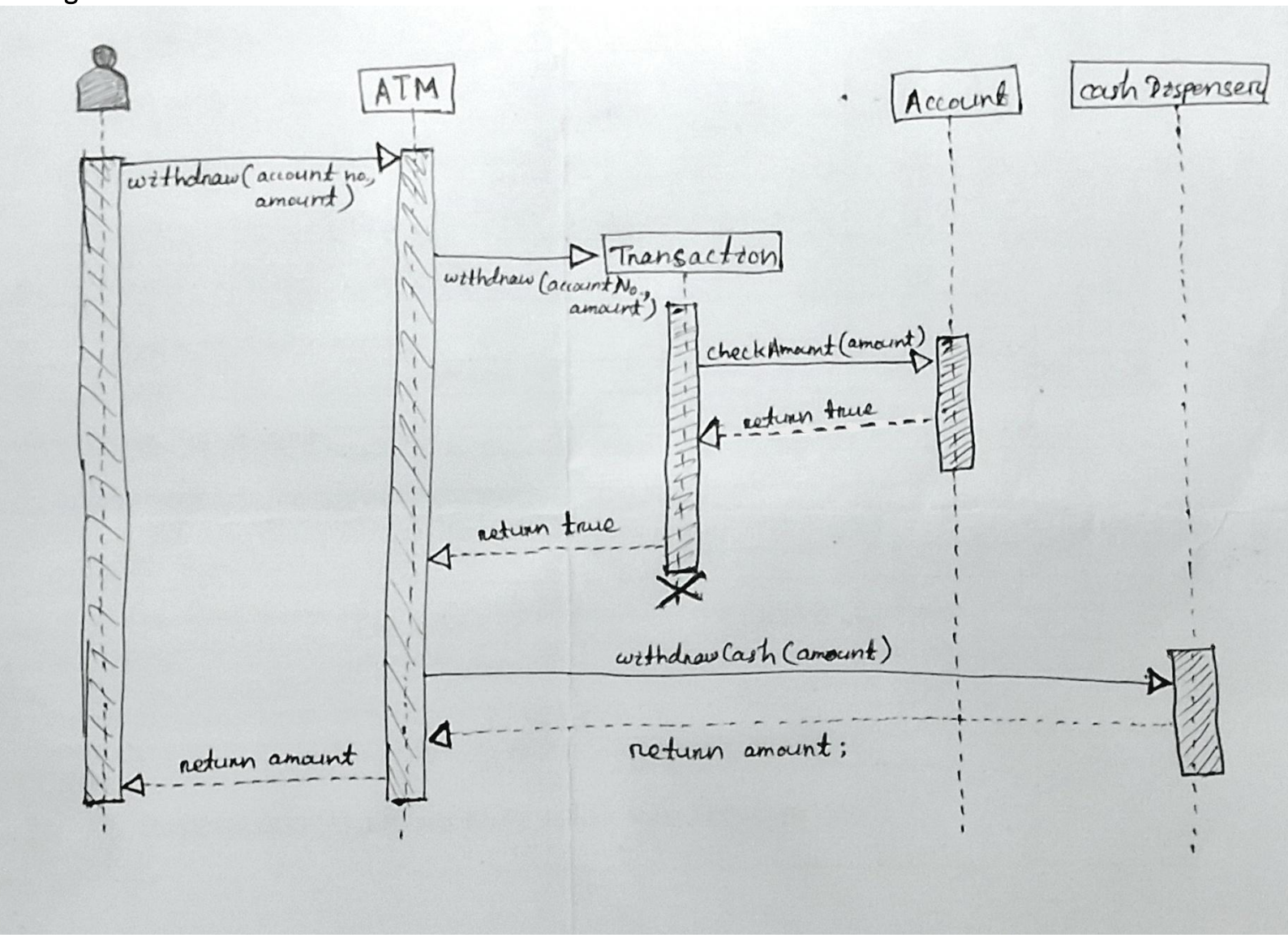
First, we have to get the use-case then we can draw the diagram.



Identifying the objects:

1. ATM
2. User
3. Transaction
4. Account
5. Cash_dispenser

Diagram:



Alt: [if – else]
Option: [if]
Loop: [for / while]