DAY 3

1. What is Inheritance? (The Parent-Child Relationship)

Inheritance is like receiving genetics from your parents. Just as you inherit traits (like eye color or height) from your parents, in programming, a **Child Class** inherits properties and behaviors from a **Parent Class**.

- **Core Concept:** Reusability. You write the common code once in the Parent, and all Child classes get it for free.
- **Parent Class (Base Class):** The generic version (e.g., Car).
- **Child Class (Derived Class):** The specific version (e.g., Manual Car or Electric Car).

Hinglish Analogy:

Real Life Example: Socho ek Car hai. Real life mein "Car" ek generic word hai.

Actual mein sadak par ya to **Manual Car** hoti hai ya **Electric Car**.

- Dono cars mein Brand, Model, aur Speed hota hai.
- Bajaye iske ki hum dono ke liye alag-alag code likhein, hum ek Parent Class Car banate hain jisme common cheezein daal dete hain.
- Manual aur Electric car bas usko "Inherit" kar leti hain.

---

2. Access Modifiers (The Rules of Inheritance)

When inheriting, we need to decide *who* can access the data. This is controlled by **Access Modifiers**.

The following table explains the 3 types:

Term	Meaning	Role in Inheritance
Public	Open to everyone.	Can be accessed by Child class and the outside world.
Private	Strictly personal.	Can **NOT** be accessed by the Child class. Only the Parent knows about it.
Protected	Family secrets.	Can be accessed by the **Child Class** but NOT by the outside world.

Hinglish Analogy:

- **Public:** Jaise park ka water tap—koi bhi use kar sakta hai.
- **Private:** Jaise aapka personal toothbrush—sirf aap use kar sakte ho, aapka beta bhi nahi.
- **Protected:** Jaise ghar ki tijori (Safe)—bahar wala nahi khol sakta, par family members (Children) access kar sakte hain.

---

3. What is Polymorphism? (Many Forms)

Polymorphism comes from two Greek words:

- **Poly** = Many
- **Morphism** = Forms

It means "One Object, Many Forms". It allows objects to respond to the same trigger (method call) in different ways.

Hinglish Analogy:

Scenario 1 (Different Objects, Same Action):

Imagine main bolu "Bhaago" (Run).

- Ek **Cheetah** alag tarike se bhagega (bohot tez).
- Ek **Insaan** alag tarike se bhagega (normal).
- Ek **Duck** alag tarike se bhagegi (waddle karte hue).
- **Action Same hai (Run)**, par karne ka tareeka sabka alag hai.

Scenario 2 (Same Object, Different Situation):

Ek Insaan ko bolo "Bhaago".

- Agar wo morning jog pe hai -> Dheere bhagega.
- Agar uske peeche **Sher (Tiger)** pad gaya -> Jaan bacha ke tez bhagega.
- **Insaan wahi hai**, bas situation (parameter) badal gayi.

---

4. Types of Polymorphism

In programming, we implement Polymorphism in two ways:

Type	Technical Name	When it happens?	Example
Static Polymorphism	Method Overloading	Compile Time	accelerate() vs accelerate(int speed)
Dynamic Polymorphism	Method Overriding	Run Time	ManualCar accelerates differently than ElectricCar

A. Static Polymorphism (Method Overloading)

When you have multiple methods with the **Same Name** but **Different Parameters** (Arguments) in the same class.

- **Example:**
  - accelerate() -> Car speeds up normally (by 20km/h).
  - accelerate(50) -> Car speeds up specifically to 50km/h.

Hinglish Analogy:

Jaise mummy ko bolo "Khaana do".

- Sirf "Khaana do" bola -> Mummy normal dal-chawal dengi.
- "Khaana do" + (Pizza) bola -> Mummy Pizza dengi.
- Request same thi, bas input (argument) badalne se output badal gaya.

B. Dynamic Polymorphism (Method Overriding)

When the Child Class changes the definition of a method that it inherited from the Parent Class.

- **Example:**
  - Parent Car has a method accelerate().
  - Manual Car says: "Main gear shift karke accelerate karungi." (Override).
  - Electric Car says: "Main battery power se accelerate karungi." (Override).

---

5. Deep Dive: Input to Output Flow (The Code Logic)

The following table explains the journey of the accelerate function:

Step	Action	Explanation
1. Parent Definition	virtual void accelerate()	Parent class (Car) declares the function but keeps it "Virtual" (flexible).
2. Inheritance	ElectricCar : public Car	Electric car inherits the function.
3. Overriding	void accelerate() { ... }	Electric car changes the logic: "Decrease Battery, Increase Speed".
4. Execution	myTesla.accelerate()	Computer checks: "Is this a generic car or a Tesla?" It sees it's a Tesla and runs the **New** logic, not the old one.

---

6. Summary & Key Points

- **Inheritance:** Don't Repeat Yourself (DRY). Move common code to a Parent class.
- **Polymorphism:** Flexibility. Treat different objects (Manual/Electric) as the same type (Car) but let them behave differently.
- **Virtual Functions:** The magic keyword in C++ that enables Dynamic Polymorphism (Overriding).
- **Real World Mapping:**
    - **Car** = Class
    - **Tesla** = Object
    - **Speed** = Variable (Data)
    - **Accelerate** = Method (Behavior)

---

7. Interview Questions

1. What is the difference between Method Overloading and Overriding?

- **Answer:**
  - **Overloading (Static):** Same function name, different parameters (e.g., add(2,3) vs add(2,3,4)). Happens in the same class.
  - **Overriding (Dynamic):** Same function name, SAME parameters. Child class replaces the Parent's method logic.

2. Why do we use "Protected" instead of "Private"?

- **Answer:** If we make a variable Private, the Child class cannot touch it. If we want the Child class to use it but keep it hidden from the rest of the world, we use Protected.

3. What is a Virtual Function?

- **Answer:** It is a function in the Parent class that allows Child classes to provide their own specific version of that function. It is essential for Dynamic Polymorphism.

4. (Homework Question) What is Operator Overloading?

- **Answer:** Changing the behavior of standard operators (like +, -, *) for user-defined objects. Example: Using + to add two Car objects together (conceptually). Java/Python handle this differently or don't support it fully like C++.