📚Lecture 03: Inheritance & Polymorphism in OOPs

🔶 1. INHERITANCE (Virasaat)

📌 Inheritance Kya Hai?

🌍 **Real World Observation:** Real life mein objects aapas mein related hote hain.
👨‍👩‍👧 **Parent-Child Relationship:** Objects mein mostly parent-child relationship hota hai.
💻 **Programming Need:** Is relationship ko programming mein represent karne ke liye inheritance use karte hain.

🧠 Real Life Example – Car Hierarchy

```
    CAR (Parent)

       /      \

Manual Car     Electric Car (Children)
```

🚗 Manual Car:

- Gear system hota hai
- Petrol/Diesel pe chalti hai
- **Special Feature:** Gear shifting

⚡ Electric Car:

- Electricity pe chalti hai
- Automatic hoti hai
- **Special Feature:** Battery charging

📌 Common Features (Har Car Mein Hote Hain)

⚙️ Characteristics:

- Brand 🏷️
- Model 📝
- Engine Status 🔧
- Current Speed 🏎️

🧩 Behaviors:

- Start Engine 🚀
- Stop Engine 🛑
- Accelerate ⚡
- Apply Brakes 🚦

📌 Special Features (Specific to Each Type)

Manual Car Ke Special Features:

- Current Gear ⚙️
- Shift Gear method

Electric Car Ke Special Features:

- Battery Percentage 🔋
- Charge Battery method

💡 **Inheritance Ka Basic Idea:**
Parent class mein common features, aur child classes mein specific features.

🔷 Inheritance Implementation – Code Structure

✅ Basic Syntax:

```
class ManualCar : public Car {

    // ManualCar specific features

};


class ElectricCar : public Car {

    // ElectricCar specific features

};
```

📌 Access Modifiers in Inheritance

3 Types of Inheritance:

1. **Public Inheritance:**
    - Parent ke public members child mein bhi public rahenge
    - Protected members protected rahenge
    - Private members accessible nahi honge
2. **Private Inheritance:**
    - Parent ke public/protected members child mein private ban jayenge
    - Mostly use nahi hota
3. **Protected Inheritance:**
    - Parent ke public members child mein protected ban jayenge
    - Rarely use hota hai

💡 **Practical Tip:**
👉 99% cases mein hum **public inheritance** hi use karte hain.

🔶 2. POLYMORPHISM (Bahurupita)

📌 Polymorphism Kya Hai?

Word Meaning:

- Poly → Many (Bahut sare)
- Morphism → Forms (Roops)

Simple Definition:
👉 “Ek hi cheez ke multiple forms”

🧠 Real Life Examples

🐅 Example 1: Animals (Different Objects, Same Behavior)

Stimulus: Run karna

- Duck: Apne tarike se bhagegi
- Human: Apne tarike se bhagega
- Tiger: Apne tarike se bhagega

🧍 Example 2: Human (Same Object, Different Situations)

- **Situation 1:** Normal – Dheere bhagega
- **Situation 2:** Danger – Tez bhagega
  Stimulus same: Run karna

📌 Types of Polymorphism

1️⃣ Dynamic Polymorphism (Runtime)

- **Also Known As:** Method Overriding
- **Concept:** Different objects react differently to same stimulus
- **Example:** ManualCar aur ElectricCar alag tarike se accelerate karti hain

2️⃣ Static Polymorphism (Compile Time)

- **Also Known As:** Method Overloading
- **Concept:** Same object reacts differently based on parameters
- **Example:** Car accelerate with parameter and without parameter

---

🔷 Polymorphism Implementation – Code Examples

✅ Dynamic Polymorphism (Method Overriding)

Parent Class (Car):

```
class Car {

protected:

    string brand;

    string model;

    bool isEngineOn;

    int currentSpeed;


public:

    virtual void accelerate() = 0;  // Abstract method

    virtual void brake() = 0;       // Abstract method

};
```

Child Class (ManualCar):

```
class ManualCar : public Car {
private:
    int currentGear;

public:
    void accelerate() override {
        currentSpeed += 20;  // Manual car specific acceleration
        cout << "Accelerating to " << currentSpeed << " km/hr" << endl;
    }
};
```

Child Class (ElectricCar)

```
class ElectricCar : public Car {
private:
    int batteryLevel;

public:
    void accelerate() override {
        currentSpeed += 15;  // Electric car specific acceleration
        batteryLevel -= 5;   // Battery decreases
        cout << "Accelerating to " << currentSpeed << " km/hr" << endl;
    }
};
```

✅ Static Polymorphism (Method Overloading)

Same Class Mein Multiple Methods:

```
class ManualCar {
public:
    void accelerate() {
        // Default acceleration

        currentSpeed += 20;

    }


    void accelerate(int speed) {

        // Parameterized acceleration → Jitni Speed se Push utna Speed

        currentSpeed += speed;

    }

};
```

💡 **Key Difference:**

- **Method Overriding:** Different classes, same method signature
- **Method Overloading:** Same class, different method parameters

---

🎯 Complete Example – All 4 Pillars Together

🎯 Ek Hi Code Mein All Concepts:

```
// ABSTRACTION + INHERITANCE
class Car {
protected:
    string brand;
    string model;
    bool isEngineOn;
    int currentSpeed;

public:
    // Common methods - every car has these
    void startEngine() {
        isEngineOn = true;
        cout << "Engine started!" << endl;
    }

    // POLYMORPHISM - Virtual methods for overriding
    virtual void accelerate() = 0;
    virtual void accelerate(int speed) = 0;
    virtual void brake() = 0;
};
```

```cpp
// INHERITANCE
class ManualCar : public Car {
private:
    int currentGear;  // ENCAPSULATION - Private member

public:
    // POLYMORPHISM - Method Overriding
    void accelerate() override {
        currentSpeed += 20;
        cout << "Manual car accelerating to " << currentSpeed << endl;
    }

    void accelerate(int speed) override {
        currentSpeed += speed;
        cout << "Manual car accelerating to " << currentSpeed << endl;
    }

    void brake() override {
        currentSpeed -= 20;
        cout << "Manual car braking" << endl;
    }

    // Manual car specific method
    void shiftGear(int gear) {
        currentGear = gear;
        cout << "Shifted to gear " << gear << endl;
    }

};
```

🚀 Usage in Main Function

```cpp
int main() {
    ManualCar wagonR;
    ElectricCar tesla;

    wagonR.startEngine();
    wagonR.accelerate();      // Output: 20 km/hr
    wagonR.shiftGear(2);
    wagonR.accelerate(30);    // Output: 50 km/hr

    tesla.startEngine();
    tesla.accelerate();       // Output: 15 km/hr
    tesla.chargeBattery();

    return 0;

}
```

🔷 Access Modifiers – Complete Understanding

📌 3 Types of Access Modifiers:

1. Public 🌐

Koi bhi access kar sakta hai. 🧩 Example: Car ka AC temperature

2. Private 🔒

Sirf class ke andar access hota hai.
Child classes bhi access nahi kar sakti.
🧩 Example: Car ki current speed

3. Protected 🛡️

Class aur uski child classes access kar sakti hain.
Bahar se koi access nahi kar sakta.
Inheritance ke liye perfect.

💡 **Protected Ka Smart Use:**
Parent class ke variables ko protected banaye taki child classes use kar sake but outside world na kar sake.

🔶 Method Overriding vs Method Overloading

📌 Method Overriding (Dynamic Polymorphism)

- Different Classes: Parent and Child classes
- Same Method Signature: Name, return type, parameters same
- Runtime Decision: Kaun sa method call hoga runtime mein decide hota hai
- Use Case: Different objects → different behavior

📌 Method Overloading (Static Polymorphism)

- Same Class: Ek hi class ke andar
- Different Parameters: Method name same but parameters different
- Compile Time Decision: Kaun sa method call hoga compile time mein decide hota hai
- Use Case: Same object → different situations

🧠 **Simple Analogy:**

- Overriding → Different animals running differently
- Overloading → Same human running fast or slow based on situation

---

📚 POWERFUL SUMMARY – Ek Nazar Mein Poora Content

🎯 **Inheritance (Virasaat):**

- Parent-child relationship between classes
- **Real Example:** Car → ManualCar, ElectricCar
- **Benefits:** Code reuse, better organization
- **Types:** Public, Private, Protected inheritance

🎯 **Polymorphism (Bahurupita):**

- Ek hi cheez ke multiple forms
- **Dynamic:** Method Overriding
- **Static:** Method Overloading

🏛️ **OOP Ke 4 Pillars Complete:**

1. Abstraction 🎭 – Implementation hide karna
2. Encapsulation 📦 – Data bundle aur secure karna
3. Inheritance 👨‍👩‍👧 – Code reuse
4. Polymorphism 🎭 – Flexibility aur multiple forms

💡 **Golden Rules:**

- "Inheritance se code reuse, Polymorphism se flexibility."
- "Parent class = Common features, Child class = Specific features."
- "Method Overriding = Different classes, same method."
- "Method Overloading = Same class, different parameters."

🚀 Practical Implementation Flow

```
Parent Class (Common Features)

       ↓

Child Classes (Specific Features + Inheritance)

       ↓

Method Overriding (Different Behavior)

       ↓

Method Overloading (Different Situations)
```