Multiple Inheritance

Multiple inheritance is a concept in **Object-Oriented Programming (OOP)** where **one child class inherits properties and methods from more than one parent class**.

👉 In simple words:
One child, many parents.

🔷 **Simple Real-Life Example**
Imagine:

- Father gives **money**
- Mother gives **care**
- Child gets **both money and care**

This is **multiple inheritance**.

```
# Parent class 1
class Father:
    def money(self):
        print("Father provides money")

# Parent class 2
class Mother:
    def care(self):
        print("Mother provides care")

# Child class (inherits from both)
class Child(Father, Mother):
    pass

# Create object
c = Child()

# Access methods from both parents
c.money()
c.care()
```

🔹 **Output**

```
nginx

Father provides money
Mother provides care
```

🔷 When to Use Carefully ⚠

If both parents have same method name, Python follows MRO (Method Resolution Order).

🔷 What is MRO (Method Resolution Order)?

MRO is the order in which Python searches for a method when you call it on an object, especially in multiple inheritance.

👉 In simple words:

When a method is called**,** Python needs to decide from which class to take that method. The rule Python follows is called **MRO**.

🔷 How to see MRO in Python

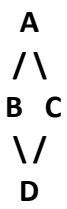
```
print(ClassName.mro())
```

🔷 What is the Diamond Problem?

The diamond problem occurs in **multiple inheritance** when:

- A class inherits from **two classes**
- Both of those classes inherit from **the same parent**
- The inheritance structure looks like a **diamond** 💎

Diamond Shape Structure :



🔷 Problem Question

If class A has a method show()
and both B and C inherit from A:

👉 **Which show() should class D use?**
This confusion is the **diamond problem.**

🔷 Python Solution: MRO (C3 Linearization)

Python solves this problem using C3 Linearization, which defines a clear and consistent order to search classes**.**

🔷 Simple Diamond Problem Example

```python
class A:
    def show(self):
        print("Class A")

class B(A):
    def show(self):
        print("Class B")

class C(A):
    def show(self):
        print("Class C")

class D(B, C):
    pass

obj = D()
obj.show()
```

Output:

Class B

🔷 Why Class B?

Python checks classes in this order (MRO):

print(D.mro())

Output:

[D, B, C, A, object]

Search order:

1. D
2. B ✅ (found show())
3. C
4. A

So Python uses B's method first.

📌 MRO

Method Resolution Order defines the order in which Python searches for methods in a class hierarchy.

📌 Diamond Problem

The diamond problem occurs in multiple inheritance when a class inherits from two classes that have a common parent, causing ambiguity in method resolution.

◈ What is Composition (in OOPs)?

Composition is an Object-Oriented Programming (OOP) concept where one class contains an object of another class to use its functionality.

👉 In simple words:
"Has-a relationship"

◈ Difference in One Line

- **Inheritance → *is-a* relationship**
- **Composition → *has-a* relationship**

◈ Simple Real-Life Example

- **A Car *has an* Engine**
- **An Employee *has an* Address**

This is composition.

```
# Part class
class Engine:
    def start(self):
        print("Engine started")

# Whole class
class Car:
    def __init__(self):
        self.engine = Engine()   # Car HAS an Engine

    def drive(self):
        self.engine.start()
        print("Car is moving")

# Object creation
car = Car()
car.drive()
```

Output:
Engine started
Car is moving

◈ Why Use Composition?

✓ Better **code reuse**
✓ Avoids complexity of multiple inheritance
✓ More flexible than inheritance
✓ Preferred in real-world designs