CodInClub
Powered By BridgeLabz

Week7 /S1/ Lecture #: DSOOPS-30

Topics Covered

- Inheritance Introduction
- Defining Derived Classes
- Forms of Inheritance (Single, Multilevel, Multiple, Hierarchical, Hybrid)
- Diamond Problem in Inheritance
- Practice Problems (Programs)

Inheritance Introduction

Inheritance is a key feature of object-oriented programming (OOP) that allows a new class to acquire properties and behaviors from an existing class.

- The existing class is called the base class (or parent/superclass).
- The new class is called the derived class (or child/subclass).
- Inheritance supports code reusability and models “is-a” relationships.

Benefits of Inheritance:

- Avoids code duplication.
- Facilitates hierarchical classification.
- Enables extension and modification of existing functionality.

Defining Derived Classes

Derived classes inherit members from base classes and can add or override members.
Syntax (C++):

```
class Base {
    // base class members
};

class Derived : public Base {
    // derived class members
```

```
};
```

Example:

```
class Animal {
public:
    void eat() { std::cout << "Eating...\n"; }
};

class Dog : public Animal {
public:
    void bark() { std::cout << "Barking...\n"; }
};
```

Here, `Dog` is derived from `Animal`, inherits `eat()`, and adds `bark()`.

Forms of Inheritance

1. Single Inheritance

A derived class inherits from exactly one base class.

Example:

```
class Vehicle { /* ... */ };
class Car : public Vehicle { /* ... */ };
```

2. Multilevel Inheritance

A derived class inherits from another derived class, forming a chain.

Example:

```
class Grandparent { /* ... */ };
class Parent : public Grandparent { /* ... */ };
class Child : public Parent { /* ... */ };
```

3. Multiple Inheritance

A derived class inherits from more than one base class.

Example:

```
class Father { /* ... */ };
class Mother { /* ... */ };
class Child : public Father, public Mother { /* ... */ };
```

4. Hierarchical Inheritance

Multiple derived classes inherit from a single base class.

Example:

```
class Animal { /* ... */ };
class Dog : public Animal { /* ... */ };
class Cat : public Animal { /* ... */ };
```

5. Hybrid Inheritance

A combination of two or more types of inheritance, often combining hierarchical and multiple inheritance.

Example:

```
class A { /* ... */ };
```

```cpp
class B : public A { /* ... */ };
class C : public A { /* ... */ };
class D : public B, public C { /* ... */ };
```

Diamond Problem in Inheritance

The diamond problem arises in hybrid/multiple inheritance when two base classes inherit from the same ancestor, and a derived class inherits from both, causing ambiguity.
Example:
cpp

```cpp
class A {
public:
    void show() { std::cout << "Class A\n"; }
};

class B : public A { };
class C : public A { };
class D : public B, public C { };
```

Attempting:

```cpp
D obj;
obj.show(); // Error: 'show' is ambiguous
```

This happens because `D` inherits two copies of `A`.
Solution: Use virtual inheritance to ensure only one shared base class instance.

Practice Problems (Programs)

Easy 1:
Implement Hierarchical Inheritance with a base class `Shape` and derived classes `Circle` and `Rectangle`. Each derived class should display a specific message.

Easy 2:
Create a program demonstrating Single Inheritance: class `Person` with `introduce()` method, and derived class `Student` with `study()` method. Instantiate `Student` and call both methods.

Medium:
Write a program showing Multilevel Inheritance: `Device` → `Computer` → `Laptop`. Each class has a unique method. Instantiate `Laptop` and call all methods.

Hard:
Write code demonstrating the Diamond Problem using classes `A`, `B`, `C`, and `D`. Then resolve it by applying virtual inheritance and show calling the shared base class method without ambiguity.

Wrap-Up & Key Takeaways

- C++ supports five main types of inheritance: Single, Multilevel, Multiple, Hierarchical, and Hybrid.
- Diamond problem is a key complexity in multiple/hybrid inheritance, solved by virtual inheritance.
- Choosing the right inheritance type simplifies design and maintenance.
- Practice coding these inheritance forms to master their use and resolve related issues.