3

In object-oriented programming, particularly in languages like C++ and Java, a derived class is a class that is based on another class, known as the base or parent class. When you create an object of the derived class, its constructor and destructor are responsible for initializing and cleaning up the resources associated with that object.

Let's discuss the concepts of derived class constructors and destructors, along with the order of their calling:

### Derived Class Constructor:

1. \*\*Initialization of Base Class:\*\*

- When a derived class object is created, the constructor of the base class is called first. This ensures that the base class's data members are initialized properly.

2. \*\*Derived Class Specific Initialization:\*\*

- After the base class constructor is executed, the derived class constructor is called. Here, you can perform additional initialization specific to the derived class, such as initializing its own data members.

3. \*\*Order of Constructor Invocation:\*\*

- The order of constructor invocation is from the topmost base class to the most derived class. This sequence ensures that the base class is fully initialized before the derived class.

### Derived Class Destructor:

1. \*\*Derived Class Cleanup:\*\*

- When an object of the derived class goes out of scope or is explicitly deleted, the destructor of the derived class is called first. This allows the derived class to release any resources it has acquired.

2. \*\*Base Class Cleanup:\*\*

- After the derived class destructor completes its cleanup, the destructor of the base class is called. This ensures proper cleanup of base class resources.

3. \*\*Order of Destructor Invocation:\*\*

- The order of destructor invocation is from the most derived class to the topmost base class. This sequence ensures that each level of the class hierarchy has an opportunity to release its resources before moving up the hierarchy.

### Example (C++ Syntax):

```cpp

class Base {

public:

Base() {

// Base class constructor code

}

~Base() {

// Base class destructor code

}

};

class Derived : public Base {

public:

Derived() {

// Derived class constructor code

}

~Derived() {

// Derived class destructor code

}

};

```

In this example, if you create an object of the `Derived` class, the order of execution would be as follows:

1. `Base` class constructor

2. `Derived` class constructor

3. `Derived` class destructor

4. `Base` class destructor