Tutorial 46 - Constructors in Derived Class in C++

Overview:

1. **Constructors in Derived Classes**:
   - If the base class constructor has no arguments, no need for a constructor in the derived class.
   - If the base class constructor has arguments, the derived class must pass arguments to it.
   - When both base and derived classes have constructors, **base class constructor** is executed first.
2. **Constructors in Multiple Inheritance**:
   - Base classes are constructed in the **order they appear** in the class declaration.
     - Example:
       - `class C : public A, public B {}`
         - **Order of constructor execution**:
           1. Constructor of `A`.
           2. Constructor of `B`.
3. **Constructors in Multilevel Inheritance**:
   - Constructors execute in the **order of inheritance**.
     - Example:
       - `class B : public A {}; class C : public B {};`
         - **Order of constructor execution**:
           1. Constructor of `A`.
           2. Constructor of `B`.
           3. Constructor of `C`.

Special Syntax:

- C++ allows a concise syntax to pass arguments to multiple base classes:

```
DerivedConstructor(arg1, arg2): Base1Constructor(arg1), Base2Constructor(arg2) {
    // Body of the derived class constructor
}

```

Special Case: Virtual Base Classes:

1. Virtual base class constructors are executed **before non-virtual base classes**.
2. If multiple virtual base classes exist, they are executed in the **order of declaration**.
3. After virtual and non-virtual base class constructors, the **derived class constructor** is executed.

Code Example:

Single Inheritance:

```
#include <iostream>
using namespace std;
class Base {
```

```cpp
public:
    Base(int a) {
        cout << "Base class constructor called with value: " << a << endl;
    }
};
class Derived : public Base {
public:
    Derived(int x, int y) : Base(x) {
        cout << "Derived class constructor called with value: " << y << endl;
    }
};
int main() {
    Derived obj(10, 20);
    return 0;
}

```

Output:

```
Base class constructor called with value: 10
Derived class constructor called with value: 20

```

Multiple Inheritance:

```cpp
#include <iostream>
using namespace std;
class A {
public:
    A() { cout << "Constructor of A" << endl; }
};
class B {
public:
    B() { cout << "Constructor of B" << endl; }
};
class C : public A, public B {
public:
    C() { cout << "Constructor of C" << endl; }
};
int main() {
    C obj;
    return 0;
}

```

Output:

```
Constructor of A
Constructor of B
Constructor of C

```

Virtual Base Class:

```cpp
#include <iostream>
using namespace std;
class Base {
public:
    Base() { cout << "Virtual Base class constructor" << endl; }
};
class A : virtual public Base {};
class B : virtual public Base {};
class Derived : public A, public B {
public:
    Derived() { cout << "Derived class constructor" << endl; }
};
int main() {
    Derived obj;
    return 0;
}

```

Output:

```
Virtual Base class constructor
Derived class constructor

```

Short Notes for Notebook

1. **Derived Class Constructor Execution**:
   - **Single Inheritance**: Base class constructor executes before derived class constructor.
   - **Multiple Inheritance**: Base classes are constructed in the order they appear in the declaration.
   - **Multilevel Inheritance**: Constructors are executed in inheritance order.
2. **Special Syntax for Arguments**:

   ```
   DerivedConstructor(arg1, arg2): Base1(arg1), Base2(arg2) {
       // Body
   }

   ```
3. **Virtual Base Class**:
   - Virtual base class constructors execute **before** non-virtual base classes.
   - Ensures a single instance of the virtual base class is constructed.
4. **Examples**:
   - **Single Inheritance**:
     - Base → Derived.
   - **Multiple Inheritance**:
     - `class C : public A, public B { };`
       - A → B → C.
   - **Virtual Base Class**:
     - Virtual base → Non-virtual → Derived.