Inheritance Using C++

Introduction

- The capability of a class to derive properties and characteristics from another class is called Inheritance.

Syntax:

```
class  derived_class_name : access-specifier  base_class_name
{
      //    body ....
};
```

Where,
Class: Keyword to create a new class.
derived_class_name: name of the new class, which will inherit the base class.
access -specifier: Specifies the access modes. Private is default.
base_class_name: name of the base class.

Syntax of Object Creation:

Child *c = new Child();
Parent *p = new Child(); // To discuss Polymorphism.

Modes Of Inheritance

- Public Mode
- Protected Mode
- Private Mode

Public Inheritance Mode

- If we derive a subclass from a public base class. Then the public member of the base class will become public in the derived class and protected members of the base class will become protected in the derived class.
  ```
  class ABC : public XYZ {...}              //  public derivation
  ```

Protected Inheritance Mode

- If we derive a subclass from a Protected base class. Then both public members and protected members of the base class will become protected in the derived class.

  ```
  class ABC : protected XYZ {...}      //  protected derivation
  ```

Private Inheritance Mode

- If we derive a subclass from a Private base class. Then both public members and protected members of the base class will become private in the derived class. They can only be accessed by the member functions of the derived class.

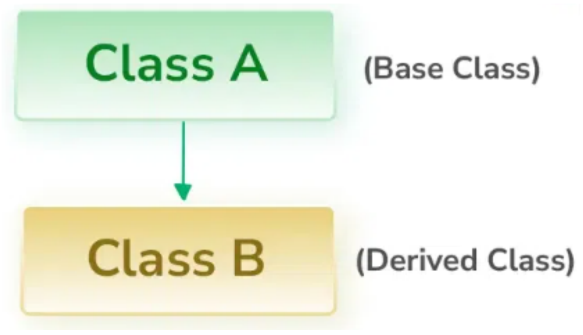
```
class ABC : private XYZ {...}      //  private derivation
class ABC: XYZ {...}               //  private derivation by default
```

Types Of Inheritance

1. Single Inheritance
2. Multilevel Inheritance
3. Multiple Inheritance
4. Hierarchical Inheritance
5. Hybrid Inheritance

1. Single Inheritance

- In single inheritance a class is allowed to inherit from only one class. I.e one base class is inherited by one derived class.



2. Multiple Inheritance

- Multiple Inheritance in C++ is a feature that allows a class to inherit from more than one base class
- This is a powerful feature but this can lead to complexity, especially with issues like ambiguity, the diamond problem, and object slicing.

Basic Syntax:

```
Class BAse1{
        // Base1 members.
};
Class Base2 {
        // Base2 members
};
Class Derived : public Base1, public Base2{
        // Derived members
};
```

Access Control:

- When using Multiple Inheritance each base class can have its own access control (public, protected or private).

```
Class Derived: public Base1, private Base2{
        // Derived Members
};
```

Problems:

1. **Ambiguity**
   - A common issue with multiple inheritance is ambiguity, where the compiler can't determine which base class member to use if there are multiple candidates.
   - Ex: Base1 and Base2 both have common method foo();

```
Class Base1{
        void foo(){}
}
Class Base2{
        void foo(){}
}
Class Derived: public Base1, public Base2{
        // Derived Members
}
```

Simply calling an object of Derived i.e obj.foo() will give an error as it is not able to decide on which base it has to call. ***Solution:***

```
Derived obj;
obj.Base1::foo();
// Calls Base1's version of foo obj.Base2::foo();
// Calls Base2's version of foo
```

2. Diamond Problem

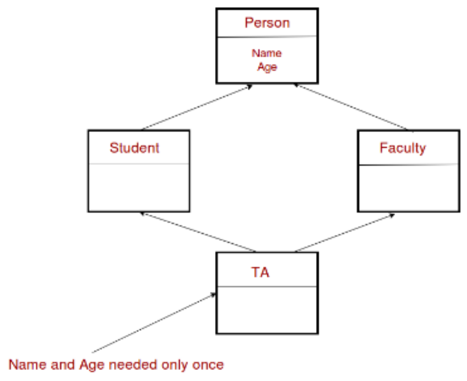
- Diamond Problem occurs when two base classes inherit from the same class, and then another class inherits from both of these base classes.
- Example:

```
Class A{
        Public:
                Int x;
}
Class B: public A{
        // Members of B
}
Class C: public A{
        // Members of C
}
Class D: public B, public C{
        // Members of D
}
```

Solution:

To solve this problem c++ provides a feature called **Virtual Inheritance.** Which ensures there is only one copy of the base class members.

```
class A {
        public:
                int x;
};

class B : virtual public A {
         // Members of B
};

class C : virtual public A {
        // Members of C
};

class D : public B, public C {
        // Members of D
};
```



Only one copy of A’s member will be present in D.

3. **Object Slicing**

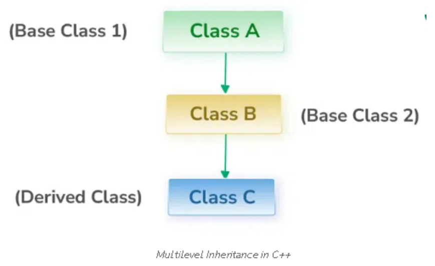
- Object slicing occurs when a derived class object is assigned to a base class object.
  In case of multiple Inheritance, this can lead to only part of the derived object being copied.
- Example:

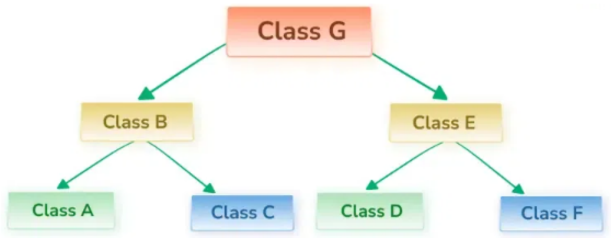
```
class Base {
public:
    int x;
};

class Derived : public Base {
public:
    int y;
};

Base b;
Derived d;
b = d; // Object slicing occurs here
```

Best Practices

- **Use multiple inheritance cautiously:** It can increase complexity and lead to difficult to diagnose issues.
- **Prefer Composition Over Inheritance:** Often, It's better to use composition {holding an object of one class within another} rather than inheritance.
- **Use virtual inheritance if necessary.**

Other Types Of Inheritance:





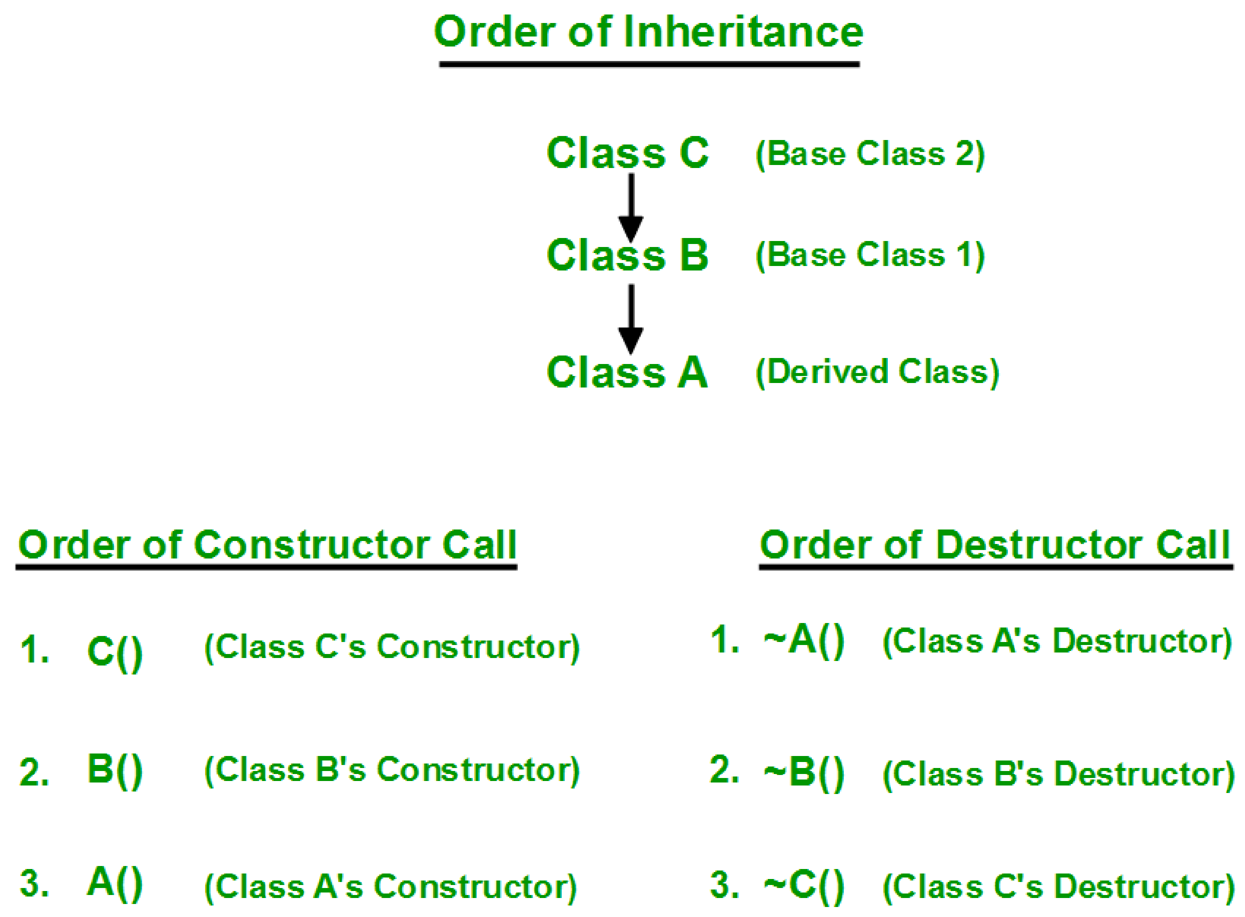
Multilevel   Hierarchical
And more …

Constructor And Destructor Creation Deletion In Inheritance

Constructors and Destructors are generally defined by the programmer and if not, the compiler automatically creates them so they are present in every class in C++. Now, the question arises what happens to the constructor and destructor when a class is inherited by another class.

In C++ inheritance, the **constructors and destructors are not inherited by the derived class,** but we can call the constructor of the base class in the derived class.

- The constructors will be called by the compiler in the order in which they are inherited. It means that base class constructors will be called first, then derived class constructors will be called.
- The destructors will be called in reverse order in which the compiler is declared.
- We can also call the constructors and destructors manually in the derived class.


Order of Inheritance
Class C (Base Class 2)
Class B (Base Class 1)
Class A (Derived Class)
Order of Constructor Call
1. C() (Class C's Constructor)
2. B() (Class B's Constructor)
3. A() (Class A's Constructor)
Order of Destructor Call
1. ~A() (Class A's Destructor)
2. ~B() (Class B's Destructor)
3. ~C() (Class C's Destructor)