19CSE201 :Advanced Programming

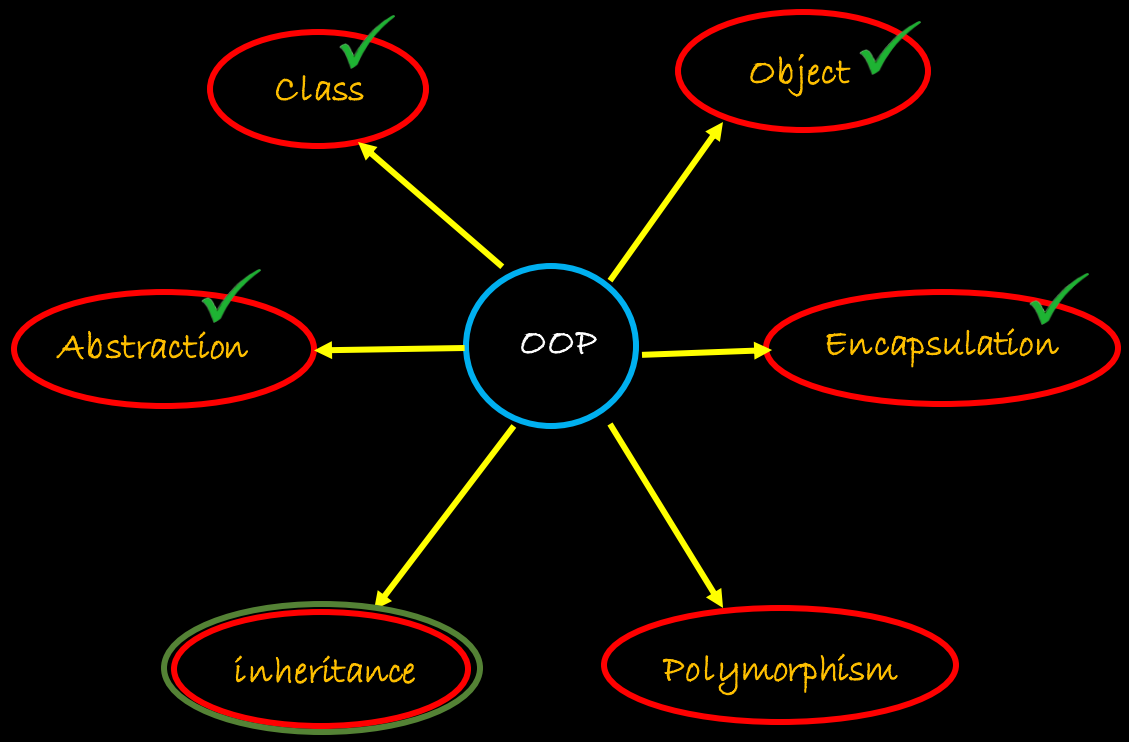
Lecture 12
More on Inheritance in C++

By
Ritwik M
Assistant Professor(SrGr)
Dept. Of Computer Science & Engg
Amrita Viswa Vidyapeetham - Coimbatore

Based on material from The Professor's C++Course by Linda W. Friedman and Stroustrup B. Programming: principles and practice using C++. Second edition, Addison Wesley; 2014.

A Quick Recap

- Inheritance
- Single inheritance
- Multiple Inheritance
- Multilevel Inheritance
- Access specifiers and inheritance
- Examples and Exercises

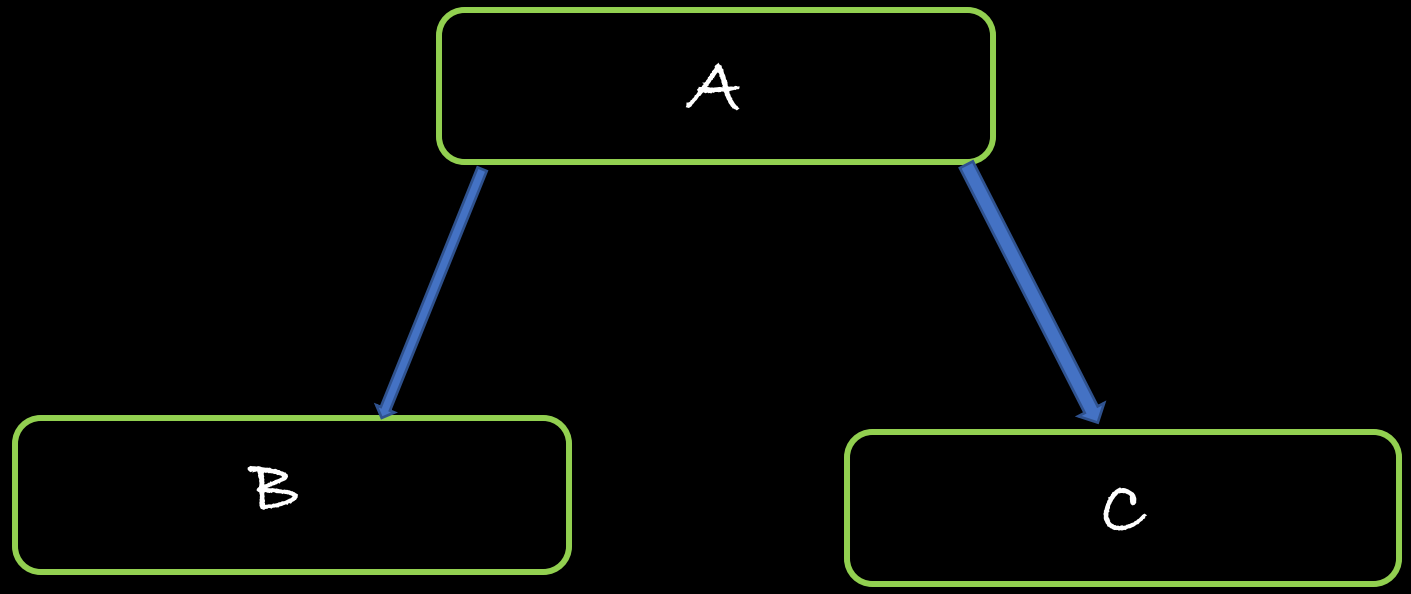


Types of Inheritance

- Single Inheritance ✓
- Multilevel Inheritance ✓
- Multiple Inheritance ✓
- Hierarchical Inheritance
- Hybrid Inheritance

Hierarchical Inheritance

- If more than one class is inherited from the base class, it's known as hierarchical inheritance.
- In hierarchical inheritance, all features that are common in child classes are included in the base class.



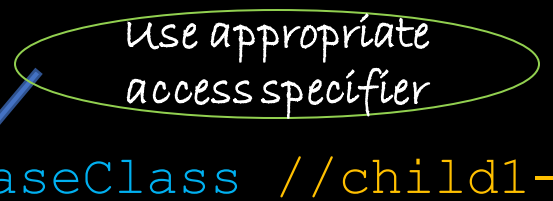
Hierarchical Inheritance Cont.

- Syntax

```
class baseClass //Single parent
   {
     ......;
   };

   class firstChild: public baseClass //child1-from baseClass
   {
     ......;
   };

   class second child:accces1 baseClass//child2-from baseClass
   {
      -----;
   };
```



Hierarchical Inheritance - Example

```
//Base class
class A {
    public:
      int x, y;
      void getdata()
      {
       cin >> x >> y;
      }
};
```

```
//Main function
int main()
{
      B obj1;
      C obj2;
      obj1.getdata();
      obj1.product();
      obj2.getdata();
      obj2.sum();
      return 0;
}
```

```
//B derived from A
class B : public A {
    public:
  void product()
  {
      cout << "\nProduct= " << x * y;
  }
};
```

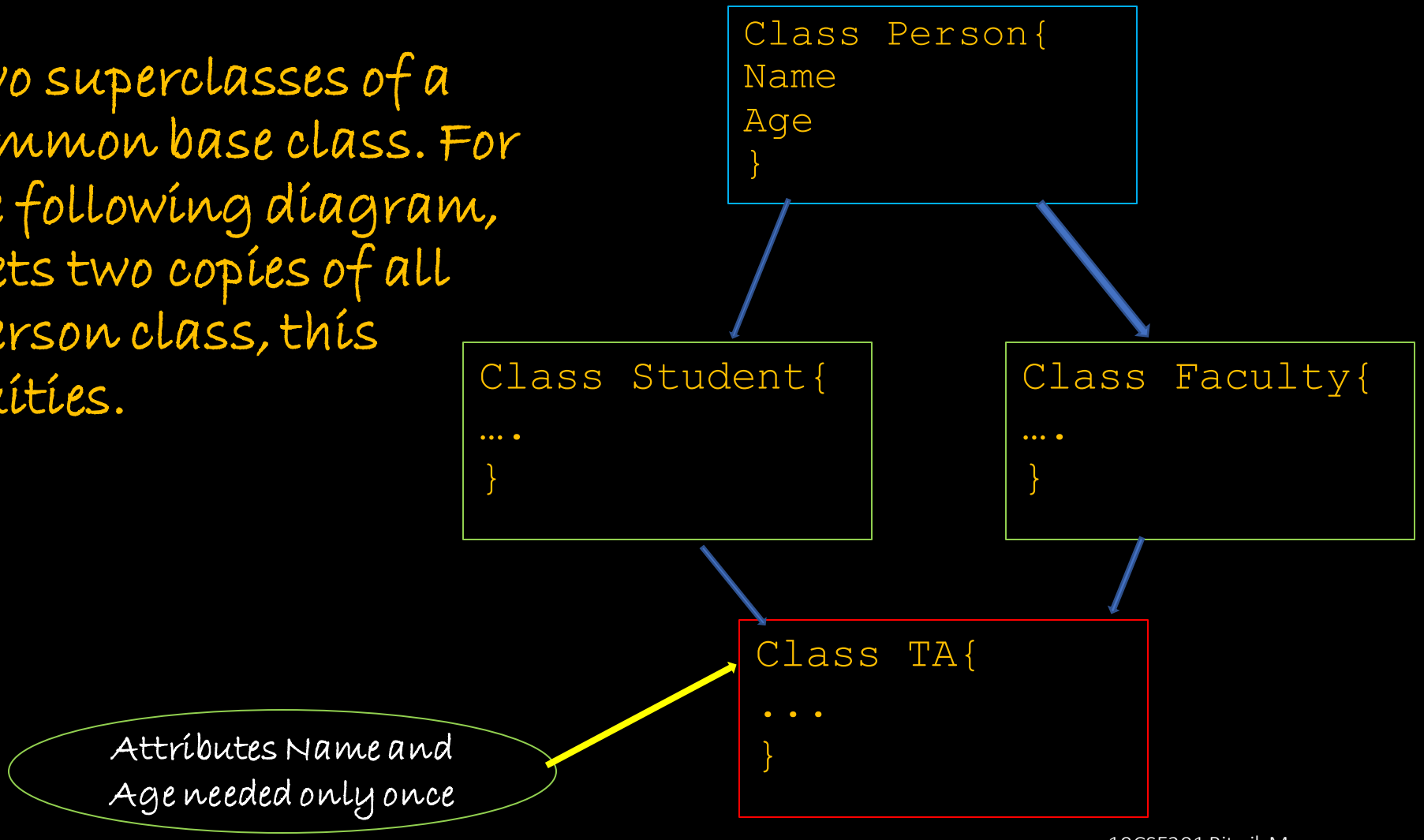
```
//C derived from A
class C : public A
{
    public:
  void sum()
  {
        cout << "\nSum= " << x + y;
  }
};
```

Hybrid Inheritance

- The inheritance where the derivation of a class involves more than one form of any inheritance is called hybrid inheritance.
- Essentially, in C++, hybrid inheritance is combination of two or more types of inheritance. It can also be called multi path inheritance.

Hybrid Inheritance – The Diamond Problem

- occurs when two superclasses of a class have a common base class. For example, in the following diagram, the TA class gets two copies of all attributes of Person class, this causes ambiguities.



Hybrid Inheritance – The Diamond Problem

```
class Person {
   // Data members of person
public:
  Person(int x)
  {
cout << "Person::Person(int ) called" << endl;   }
};
```

```
int main()   {
    TA ta1(30);
}
```

```
class Student : public Person {
   // data members of Student
public:
    Student(int x):Person(x) {
        cout<<"Student::Student(int ) called";
    }
};
```

```
class Faculty : public Person {
   // data members of Faculty
public:
    Faculty(int x):Person(x)   {
   cout<<"Faculty::Faculty(int ) called";
    }
};
```

```
class TA : public Faculty, public Student  {
public:
    TA(int x):Student(x), Faculty(x)   {
        cout<<"TA::TA(int ) called"<< endl;
    }
};
```

Solution: Virtual Keyword - Later

Order of Constructor Calls

- Whenever you create derived class object, first the base class default constructor is executed and then the derived class's constructor finishes execution.
- Points to Remember
  - Whether derived class's default constructor is called or parameterized is called, base class's default constructor is always called inside them.
  - To call base class's parameterized constructor inside derived class's parameterized constructor, we must mention it explicitly while declaring derived class's parameterized constructor.

Base class Default Constructor in Derived class Constructors

```
class Base {
   int x;
   public:
   base()
   {
         cout << "Base";
   }
};
```

```
int main()   {
     Base b;
     Derived d1;
     Derived d2(10);
}
```

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

     Derived()
     {
          cout << "Derived constructor\n";
     }

     // parameterized constructor
     Derived(int i)
     {
           cout << "Derived parameterized
constructor\n";
     }
};
```

Constructor and Destructor inheritance

```
class parent//parent class
{
public:
parent()//constructor
{
cout<<"Parent class Constructor\n";
}

~parent()//destructor
{
cout<<"Parent class Destructor\n";
}
};
```

```
class child : public parent//child class
{
 public:
   child() //constructor
   {
      cout<<"Child class Constructor\n";
   }

   ~ child() //destructor
   {
      cout<<"Child class Destructor\n";
   }
};
```

```
int main()   {
    child c;
    return 0;
}
```

Exercise 1 – What's the output?

```
class A
{float d;
   public:
    A()
{ cout<<"Constructor of class A\n";
}
};

class B: public A
{ int a = 15;
    public:
B(){
cout<<"Constructor of class B\n";
}
};
```

```
int main()
{
B b;
return 0;
 }
```

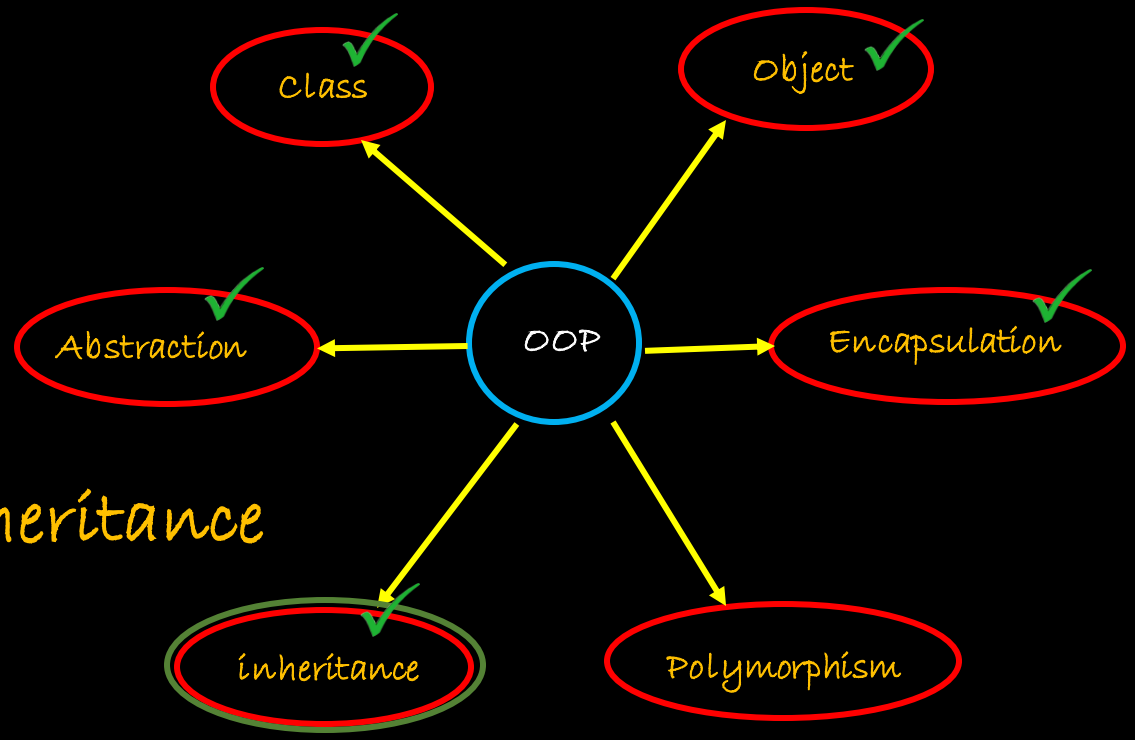
Exercise 2 – What's the output?

```
class Base1 {
 public:
     ~Base1()  { cout << " Base1's destructor" <<
endl; }
};
class Base2 {
 public:
     ~Base2()  { cout << " Base2's destructor" <<
endl; }
};
class Derived: public Base1, public Base2 {
   public:
     ~Derived()  { cout << " Derived's
destructor" << endl; }
};
```

```
int main()
{
Derived d;
return 0;
 }
```

Quick Summary

- Hierarchical Inheritance
- Hybrid Inheritance
- Constructor calls in inheritance
- Constructors and Destructors in inheritance
- Examples
- Exercises



Up Next

Memory Management in C++