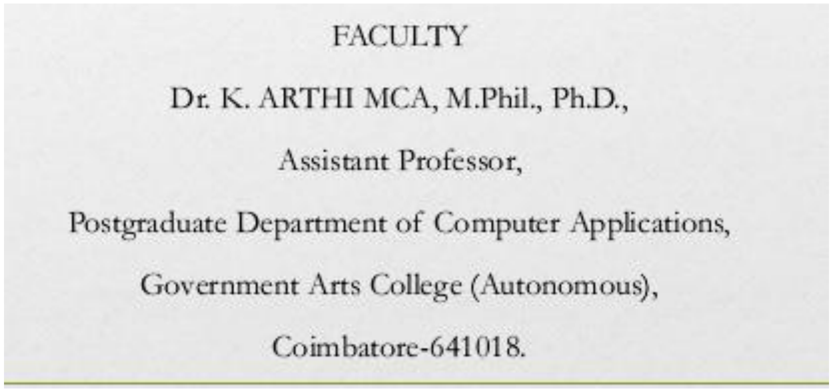
20MCA11C OBJECT ORIENTED PROGRAMMING WITH C++

UNIT IV: Inheritance



FACULTY

Dr. K. ARTHI MCA, M.Phil., Ph.D.,

Assistant Professor,

Postgraduate Department of Computer Applications,

Government Arts College (Autonomous),

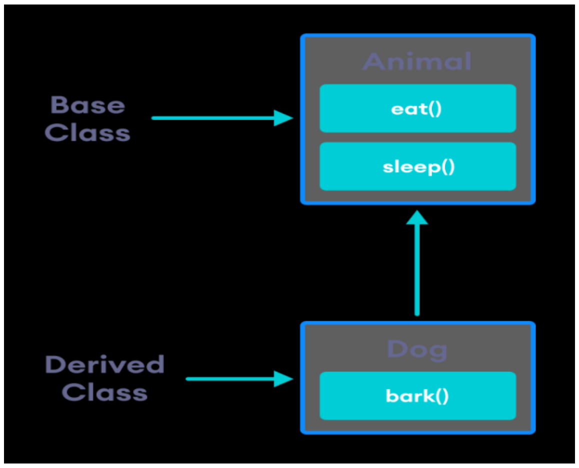
Coimbatore-641018.

Inheritance

Inheritance is one of the key features of Object-oriented programming in C++. It allows us to create a new class (derived class) from an existing class (base class).

Inheritance is a mechanism for

- building class types from existing class types
- defining new class types to be a
  - specialization
  - augmentation of existing types
  -



Define a Class Hierarchy

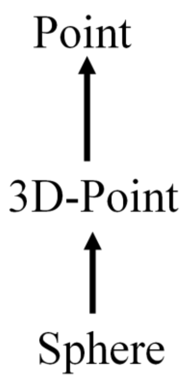
- Syntax:

  class *DerivedClassName* : access-level *BaseClassName*

  where

  – access-level specifies the type of derivation
    - private by default, or
    - public
- Any class can serve as a base class
  – Thus a derived class can also be a base class

Class Derivation



```
class Point{
    protected:
      int x, y;
    public:
      void set (int a, int b);
};
```

```
class 3D-Point : public Point{
    private:
      double z;
    … …
};
```

```
class Sphere : public 3D-Point{
    private:
      double r;
    … …
};
```

Point is the base class of 3D-Point, while 3D-Point is the base class of Sphere

Access Rights of Derived Classes

Type of Inheritance

Access Control for Members

	private	protected	public
private	-	-	-
protected	private	protected	protected
public	private	protected	public

- The type of inheritance defines the access level for the members of derived class that are inherited from the base class

```cpp
// C++ program to demonstrate inheritance

#include <iostream>
using namespace std;

// base class
class Animal {

   public:
    void eat() {
        cout << "I can eat!" << endl;
    }

    void sleep() {
        cout << "I can sleep!" << endl;
    }
};
```

```
// derived class
class Dog : public Animal {

   public:
    void bark() {
        cout << "I can bark! Woof woof!!" << endl;
    }
};

int main() {
    // Create object of the Dog class
    Dog dog1;

    // Calling members of the base class
    dog1.eat();
    dog1.sleep();

    // Calling member of the derived class
    dog1.bark();

    return 0;
}
```

C++ Multiple, Multilevel and Hierarchical Inheritance

C++ Multilevel Inheritance

In C++ programming, not only you can derive a class from the base class but you can also derive a class from the derived class. This form of inheritance is known as multilevel inheritance.

```
class A
```

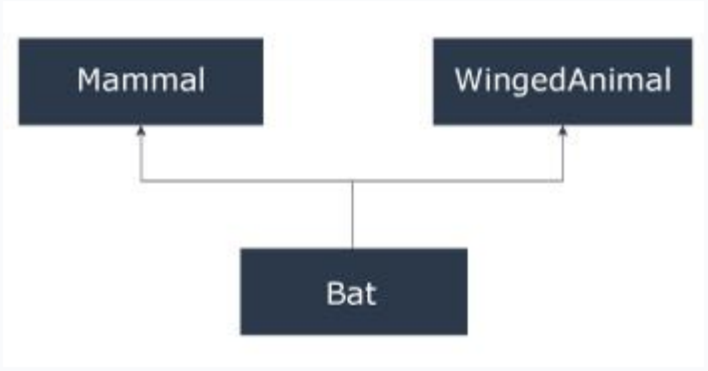
```
{

... .. ...

};

class B: public A

{

... .. ...

};

class C: public B

{

... ... ...

};
```

Example 1: C++ Multilevel Inheritance

```
#include <iostream>
using namespace std;

class A
{
    public:
      void display()
      {
          cout<<"Base class content.";
      }
};

class B : public A
{

};

class C : public B
{
};

int main()
{
    C obj;
    obj.display();
    return 0;
}
```

C++ Multiple Inheritance

In C++ programming, a class can be derived from more than one parents. For example: A class `Bat` is derived from base classes `Mammal` and `WingedAnimal`. It makes sense because bat is a mammal as well as a winged animal.



Multiple Inheritance in C++ Programming

```
#include <iostream>
using namespace std;

class Mammal {
  public:
    Mammal()
    {
      cout << "Mammals can give direct birth." << endl;
    }
};

class WingedAnimal {
  public:
    WingedAnimal()
    {
      cout << "Winged animal can flap." << endl;
    }
};

class Bat: public Mammal, public WingedAnimal {

};

int main()
{
    Bat b1;
    return 0;
}
```

C++ 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.

For example: Physics, Chemistry, Biology are derived from Science class.

Syntax of Hierarchical Inheritance

```
class base_class {

     ... .. ...

}



class first_derived_class: public base_class {

     ... .. ...
}



class second_derived_class: public base_class {

     ... .. ...

}



class third_derived_class: public base_class {

     ... .. ...

}
```

Pure Virtual Functions and Abstract Classes in C++

A pure virtual function (or abstract function) in C++ is a virtual function for which we don't have implementation, we only declare it. A pure virtual function is declared by assigning 0 in declaration. See the following example.

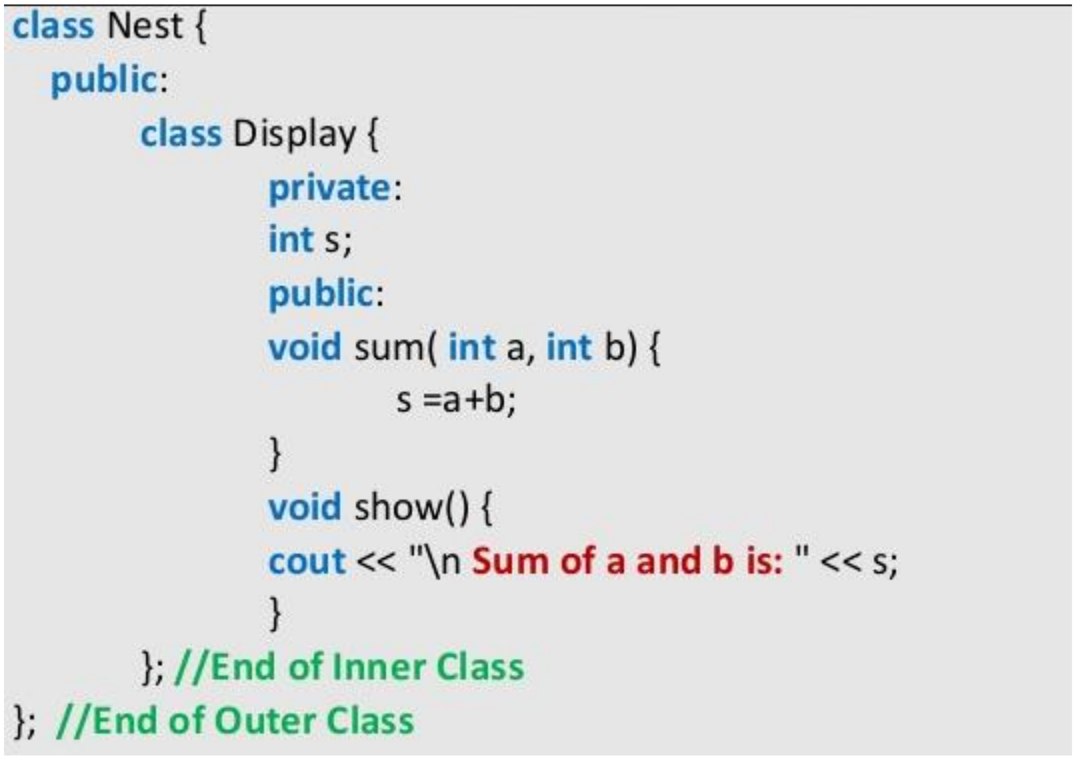
```
// An abstract class
class Test
{
    // Data members of class
public:
    // Pure Virtual Function
    virtual void show() = 0;

   /* Other members */
};
```

A pure virtual function is implemented by classes which are derived from a Abstract class. Following is a simple example to demonstrate the same.

```
#include<iostream>
using namespace std;

class Base
{
   int x;
public:
    virtual void fun() = 0;
    int getX() { return x; }
};

// This class inherits from Base and implements fun()
class Derived: public Base
{
    int y;
public:
    void fun() { cout << "fun() called"; }
};

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

Nested Classes in C++

A nested class is a class which is declared in another enclosing class. A nested class is a member and as such has the same access rights as any other member. The members of an enclosing class have no special access to members of a nested class.



```
class Nest {
   public:
           class Display {
                   private:
                   int s;
                   public:
                   void sum( int a, int b) {
                           s =a+b;
                   }
                   void show() {
                   cout << "\n Sum of a and b is: " << s;
                   }
           }; //End of Inner Class
}; //End of Outer Class
```

THANK YOU