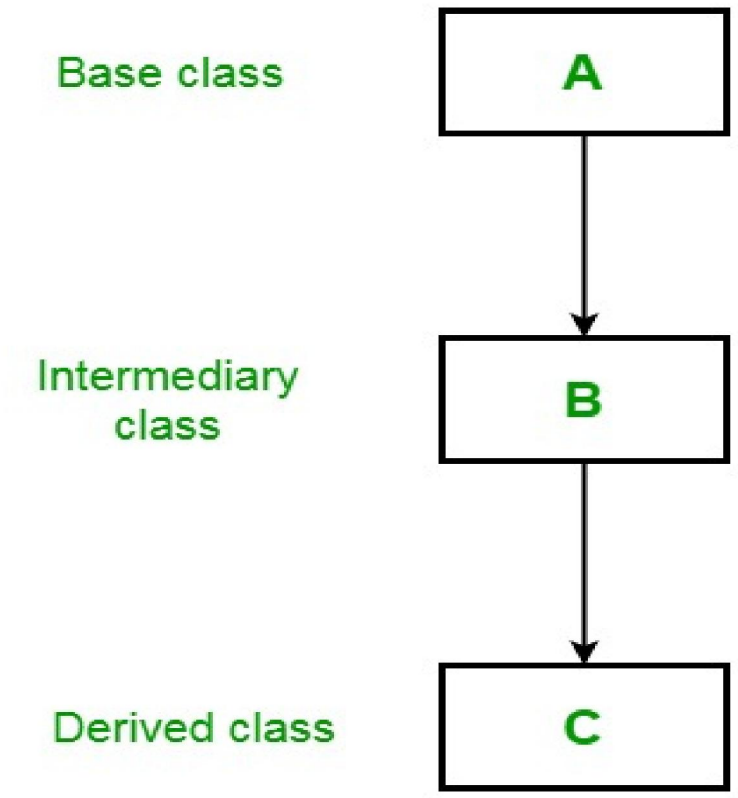
LECTURE-13

Multilevel Inheritance

When the inheritance is such that, the class A serves as a base class for a derived class B which in turn serves as a base class for the derived class C. This type of inheritance is called 'MULTILEVEL INHERITENCE'. The class B is known as the 'INTERMEDIATE BASE CLASS' since it provides a link for the inheritance between A and C. The chain ABC is called 'INHERITENCE*PATH' for e.g.



The declaration for the same would be:

```
Class A
{
      //body
}
Class B : public A
{
      //body
}
Class C : public B
{
     //body
}
```

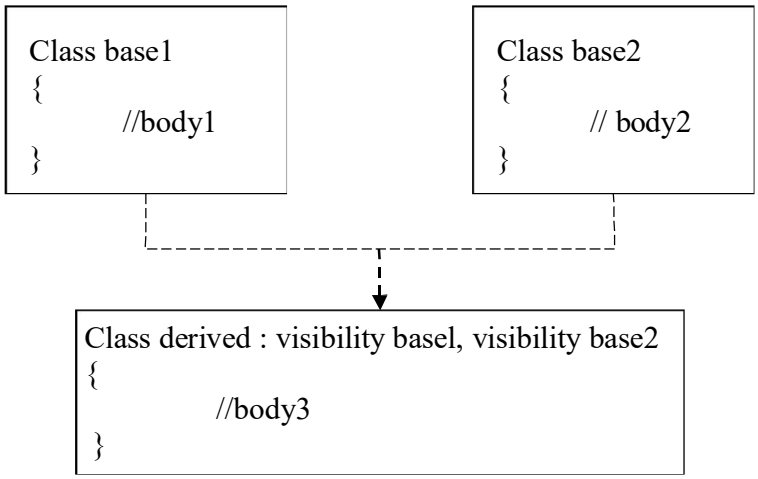
This declaration will form the different levels of inheritance. Following program exhibits the multilevel inheritance.

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

// Base class
class Animal {
public:
    void eat() {
        cout << "Animal is eating." << endl;
    }
};
// Intermediate class derived from Animal
class Mammal : public Animal {
public:
    void breathe() {
        cout << "Mammal is breathing." << endl;
    }
};
// Derived class derived from Mammal
class Dog : public Mammal {
public:
    void bark() {
        cout << "Dog is barking." << endl;
    }
};
```

```
int main() {
    // Create an object of the derived class
    Dog myDog;

    // Access methods from the base class
    myDog.eat();

    // Access methods from the intermediate class
    myDog.breathe();
   // Access methods from the derived class
    myDog.bark();
}
```

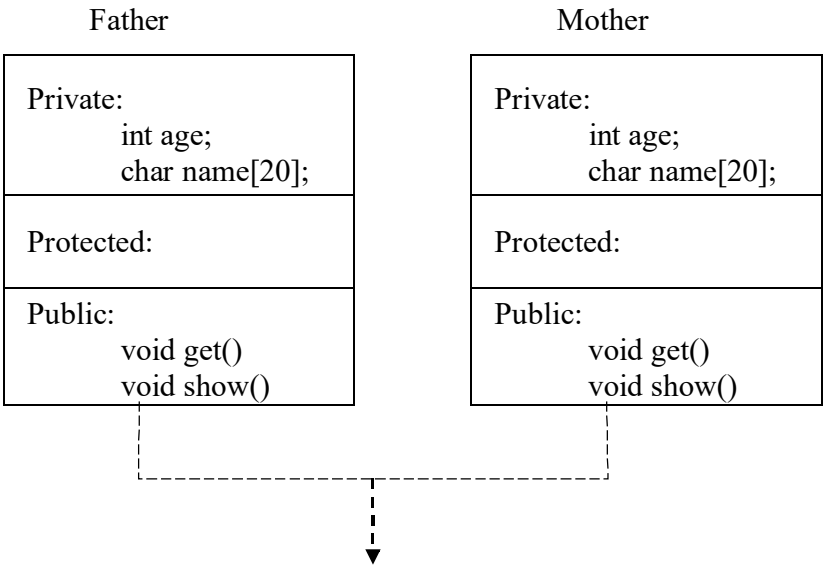
Multiple Inheritances

A class can inherit the attributes of two or more classes. This mechanism is known as 'MULTIPLE INHERITENCE'. Multiple inheritance allows us to combine the features of several existing classes as a starting point for defining new classes. It is like the child inheriting the physical feature of one parent and the intelligence of another. The syntax of the derived class is as follows:

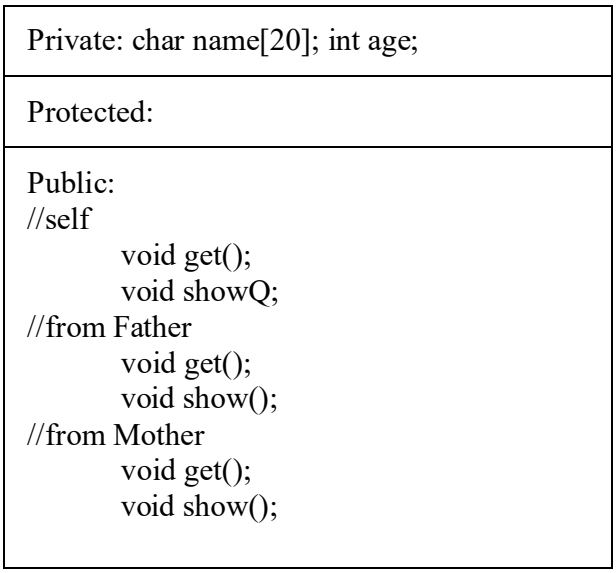


Where the visibility refers to the access specifiers i.e. public, private or protected. Following program shows the multiple inheritance.

Diagrammatic Representation of Multiple Inheritance is as follows:



Class daughter: public Father, public Mother



```
#include<iostream>
using namespace std;
class father          //Declaration of base classl
{
        int age ;
        char name   [20] ;
        public:
                void get ();
                void show () ;
};
void father :: get ()
{
        cout << "your father name please";
        cin>> name;
        cout<< "Enter the age";
        cin>> age;
}
void father :: show ()
{
        cout<< "In  my father's  name  is: "<<name<< "In  my father's  age is:"<<age;
}
class mother          //Declaration of base class 2
{
        char name [20] ;
        int age ;
        public:
                void get ()
                {
                        cout << "mother's name please" << "In";
                        cin >> name;
                        cout << "mother's age please" << "in";
                        cin >> age;
                }
                void show ( )
                {
        cout << "In my mother's name is: " <<name; cout << "In my mother's age is: " <<age;
                }
};
class daughter: public father, public mother //derived class inheriting
{       //publicly
        char name [20];     //the features of both the base class;
        public:
                void get ();
```

```
                    void show ();
};
void daughter :: get ()
{
        father :: get () ;
        mother :: get () ;
        cout << "child's name: ";
        cin >> name;
        cout << "child's standard";
}
void daughter :: show ()
{
        father :: show ();
        father :: show () ;
        cout << "In child's name is : " <<name;
}
main ()
{
        daughter d1;
        d1.get () ;
        d1.show () ;
}
```

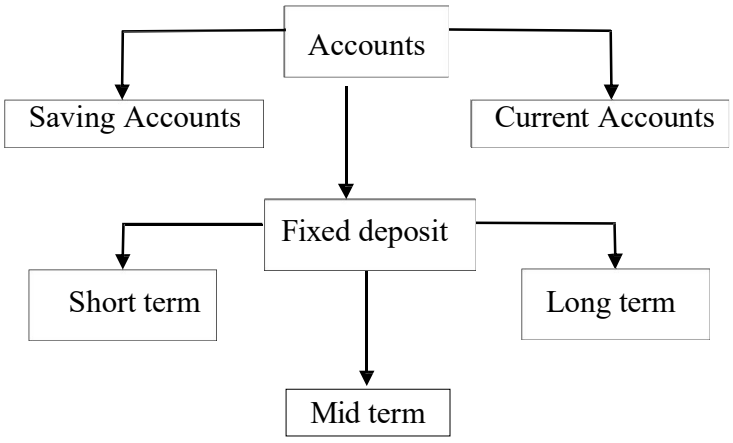
Example

```
#include <iostream>
Using namespace std;
class ElectronicDevice {
public:
    void powerOn() {
        cout << "Electronic device powered on\n";
    }
    void powerOff() {
        cout << "Electronic device powered off\n";
    }
};
class Camera {
public:
    void takePhoto() {
        cout << "Photo taken\n";
    }
};
```

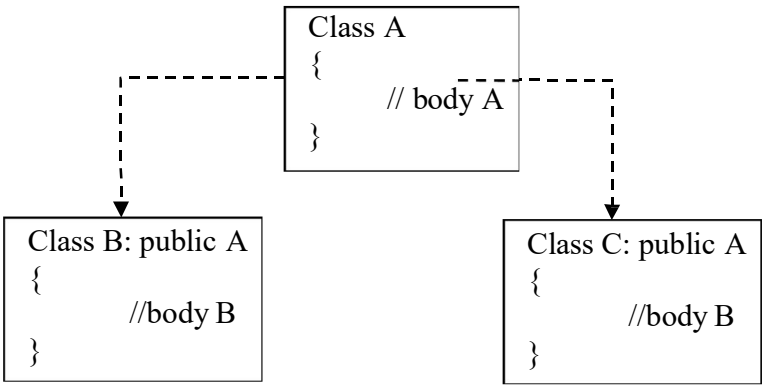
```
class Smartphone : public ElectronicDevice, public Camera {
public:
    void makeCall() {
        cout << "Call made\n";
    }

    void sendText() {
        cout << "Text message sent\n";
    }
};

int main() {
    Smartphone myPhone;
    myPhone.powerOn();
    myPhone.powerOff();
    myPhone.takePhoto();
    myPhone.makeCall();
    myPhone.sendText();
}
```

Hierarchical Inheritance

Another interesting application of inheritance is to use is as a support to a hierarchical design of a class program. Many programming problems can be cast into a hierarchy where certain features of one level are shared by many others below that level for e.g.



In general, the syntax is given as



In C++, such problems can be easily converted into hierarchies. The base class will include all the features that are common to the subclasses. A sub-class can be constructed by inheriting the features of base class and so on.

```
#include <iostream>
Using namespace std;
class Animal {
public:
    void eat() {
        cout << "Animal is eating\n";
    }
    void sleep() {
        cout << "Animal is sleeping\n";
    }
};
class Dog : public Animal {
public:
    void bark() {
        cout << "Dog is barking\n";
    }
};
class Cat: public Animal {
public:
    void meow() {
        cout << "Cat is meowing\n";}
};
```

```
int main() {
    Dog myDog;
    Cat myCat;
    // Accessing functions from the base class Animal
    myDog.eat();
    myDog.sleep();
    myCat.eat();
    myCat.sleep();
    // Accessing functions specific to the derived classes
    myDog.bark();
    myCat.meow();
}
```

Hybrid Inheritance

There could be situations where we need to apply two or more types of inheritance to design a program. Basically, Hybrid Inheritance is the combination of one or more types of the inheritance. Here is one implementation of hybrid inheritance.

```
#include <iostream>
// Base class 1
class Animal {
public:
    void eat() {
        std::cout << "Animal is eating\n";
    }

    void sleep() {
        std::cout << "Animal is sleeping\n";
    }
};
// Base class 2
class Mammal {
public:
    void giveBirth() {
        std::cout << "Mammal giving birth\n";
    }
};
class Dog : public Animal, public Mammal {
public:
    void bark() {
```

```
        std::cout << "Dog is barking\n";
    }
};
class Cat : public Animal {
public:
    void meow() {
        std::cout << "Cat is meowing\n";
    }
};
class Pet : public Dog, public Cat {
public:
    void play() {
        std::cout << "Pet is playing\n";
    }
};
int main() {
    Pet myPet;
    // Accessing functions from Animal (via Dog and Cat)
    myPet.eat();
    myPet.sleep();
    // Accessing functions from Mammal (via Dog)
    myPet.giveBirth();
    // Accessing functions specific to Dog and Cat
    myPet.bark();
    myPet.meow();
    // Accessing function specific to Pet
    myPet.play();
}
```

Student Activity

1. What is the major use of multilevel Inheritance?
2. How are arguments sent to the base constructors in multiple inheritance? Whose responsibility is it.
3. What is the difference between hierarchical and hybrid Inheritance.
4. Write C++ program to demonstrate example of single inheritance to check if the number is even or odd.