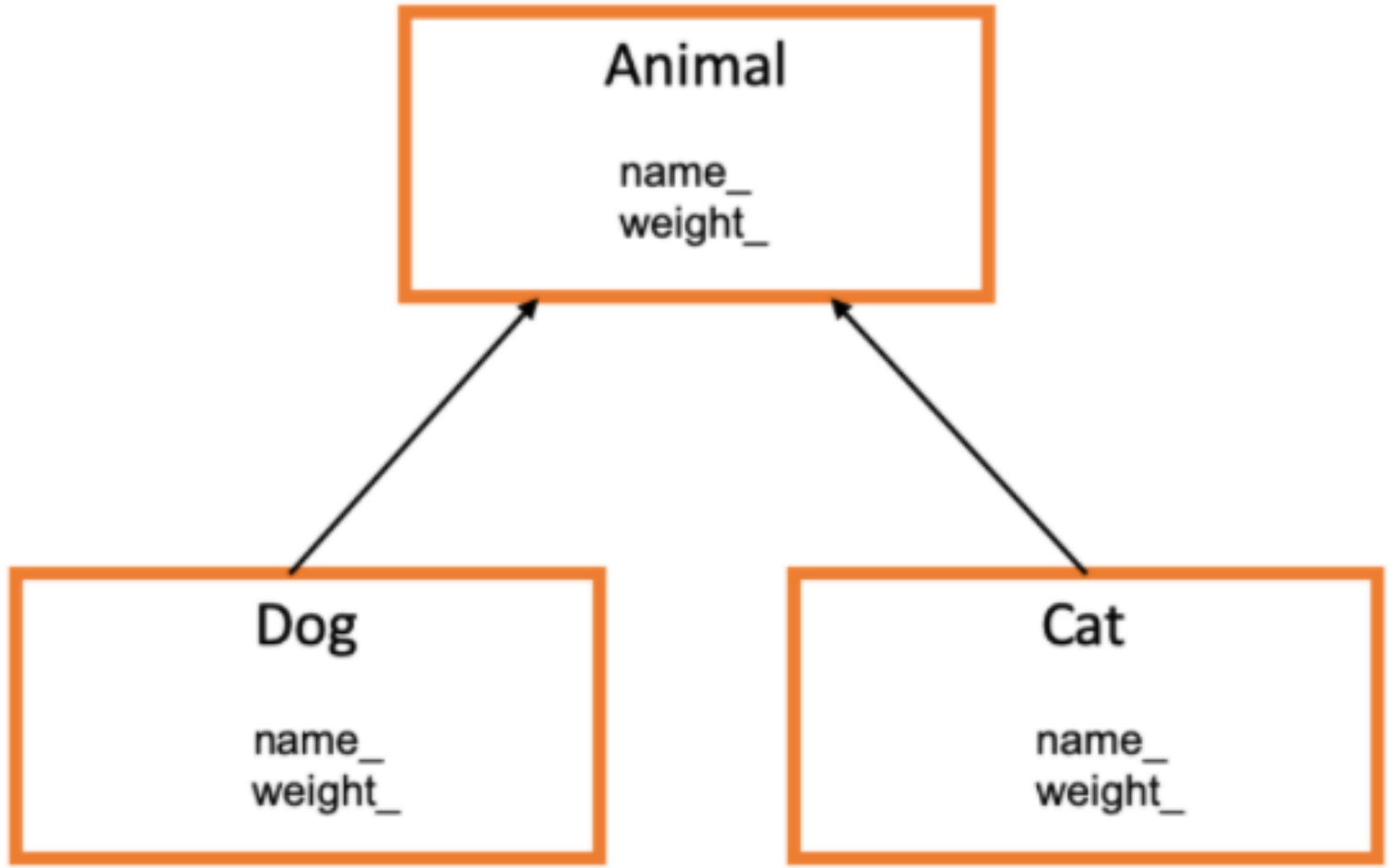
Inheritance

Inheritance

- C++ strongly supports the concept of reusability.
- Once a class has been written and tested, it can be adopted by another programmers. This is basically created by defining the new classes, reusing the properties of existing ones.
- The mechanism of deriving a new class from an old one is called 'INHERITANCE’.
- The old class is called 'BASE' (parent)class and the new one is called 'DERIVED’ (child)class.


Animal
name_
weight_
Dog
name_
weight_
Cat
name_
weight_

- In the inheritance, some of the base class data elements and member functions are inherited into the derived class.
- We can add our own data and member functions and thus extend the functionality of the base class.
- Inheritance, when used to modify and extend the capabilities of the existing classes, becomes a very powerful tool for incremental program development.

When a class inherits from another class, there are **three** benefits:

(1) You can reuse the methods and data of the existing class

(2) You can extend the existing class by adding new data and new methods

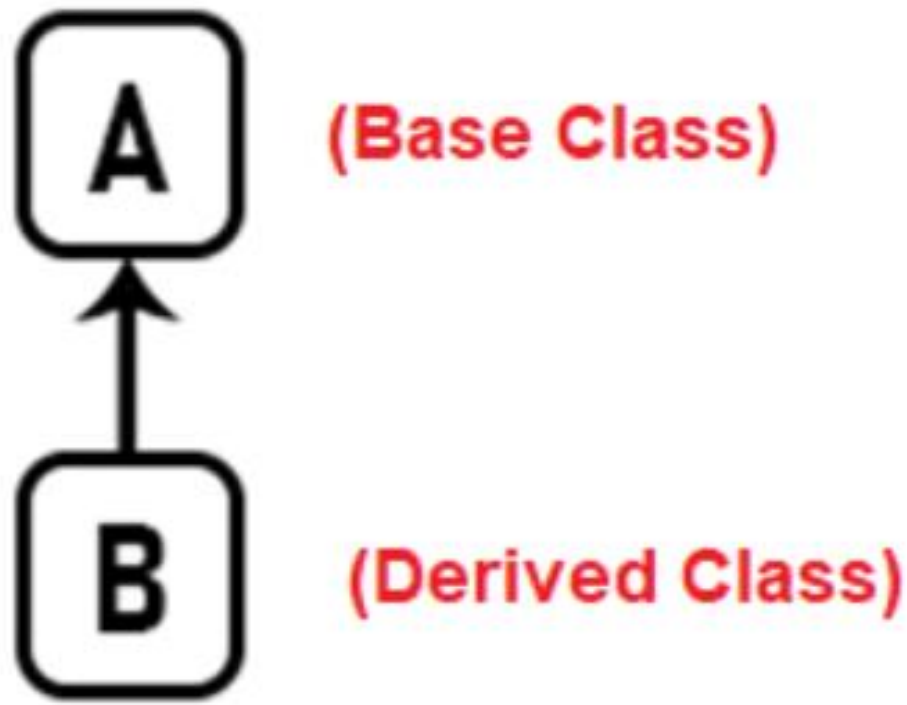
(3) You can modify the existing class by overloading its methods with your own implementations

In C++, we have 5 different types of Inheritance.

- Single Inheritance
- Multiple Inheritance
- Hierarchical Inheritance
- Multilevel Inheritance
- Hybrid Inheritance (also known as Virtual Inheritance)

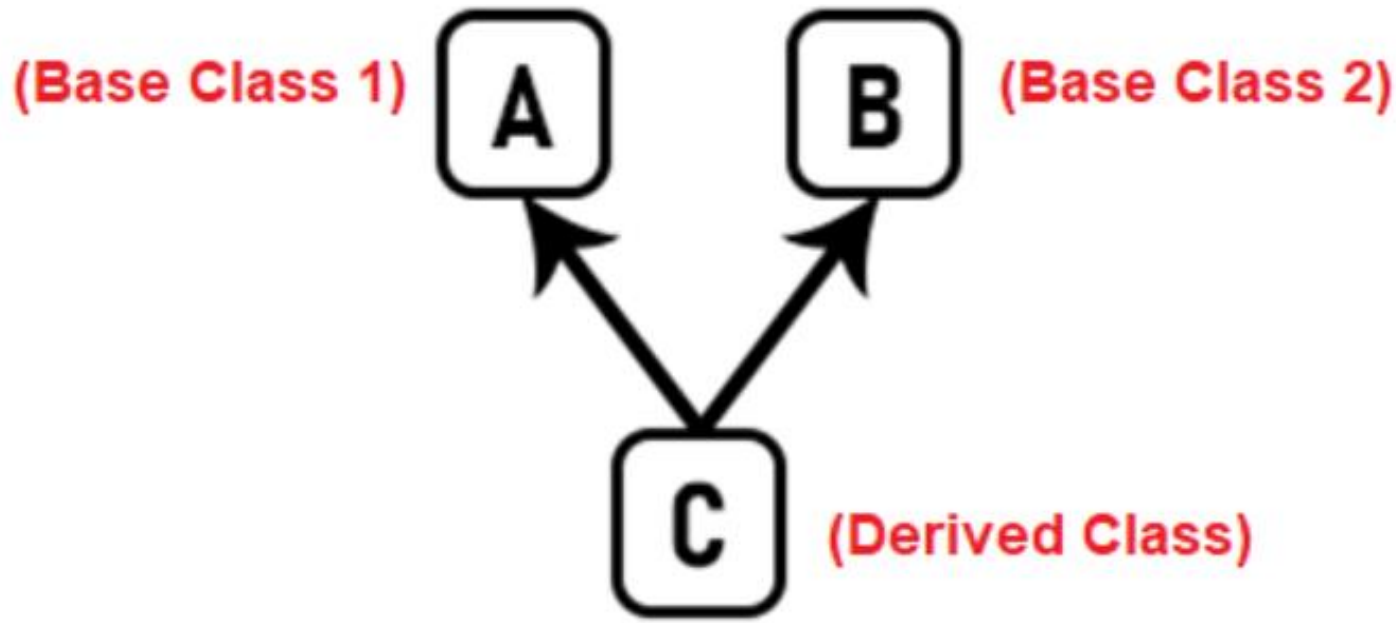
Single Inheritance in C++

In this type of inheritance one derived class inherits from only one base class. It is the most simplest form of Inheritance.



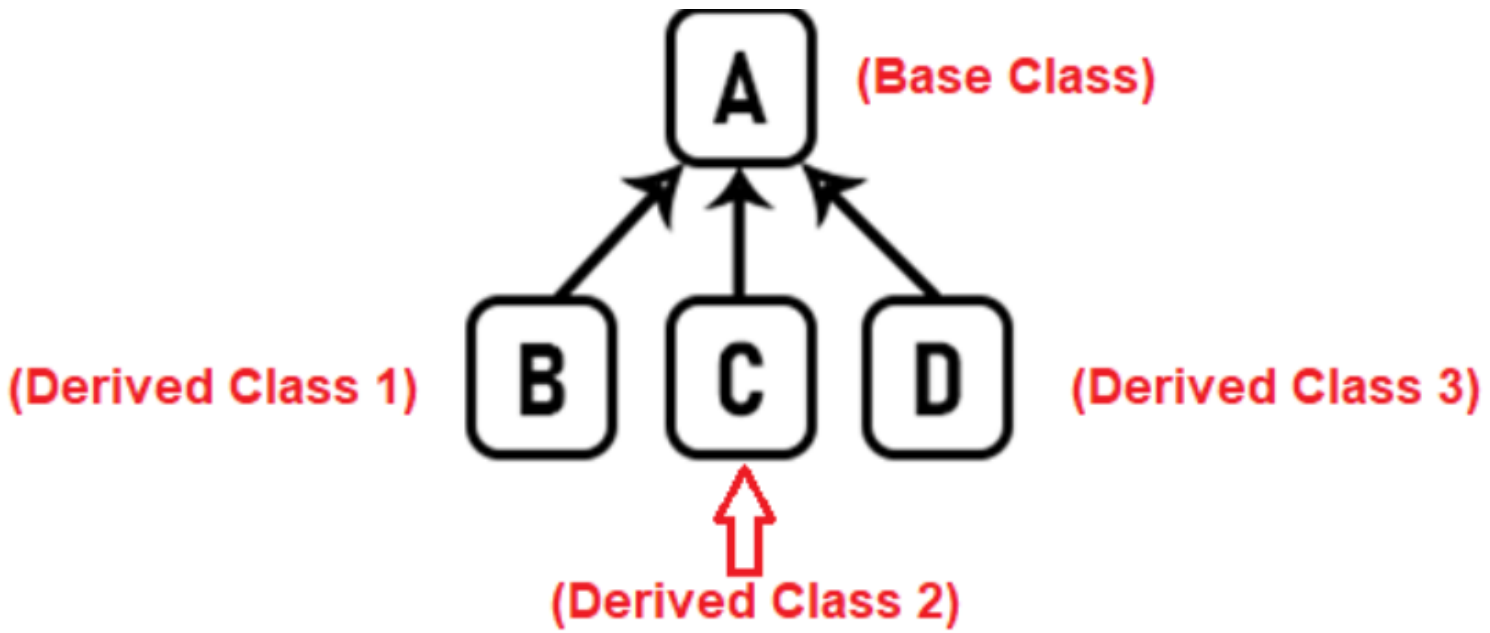
Multiple Inheritance in C++

In this type of inheritance a single derived class may inherit from two or more than two base classes.



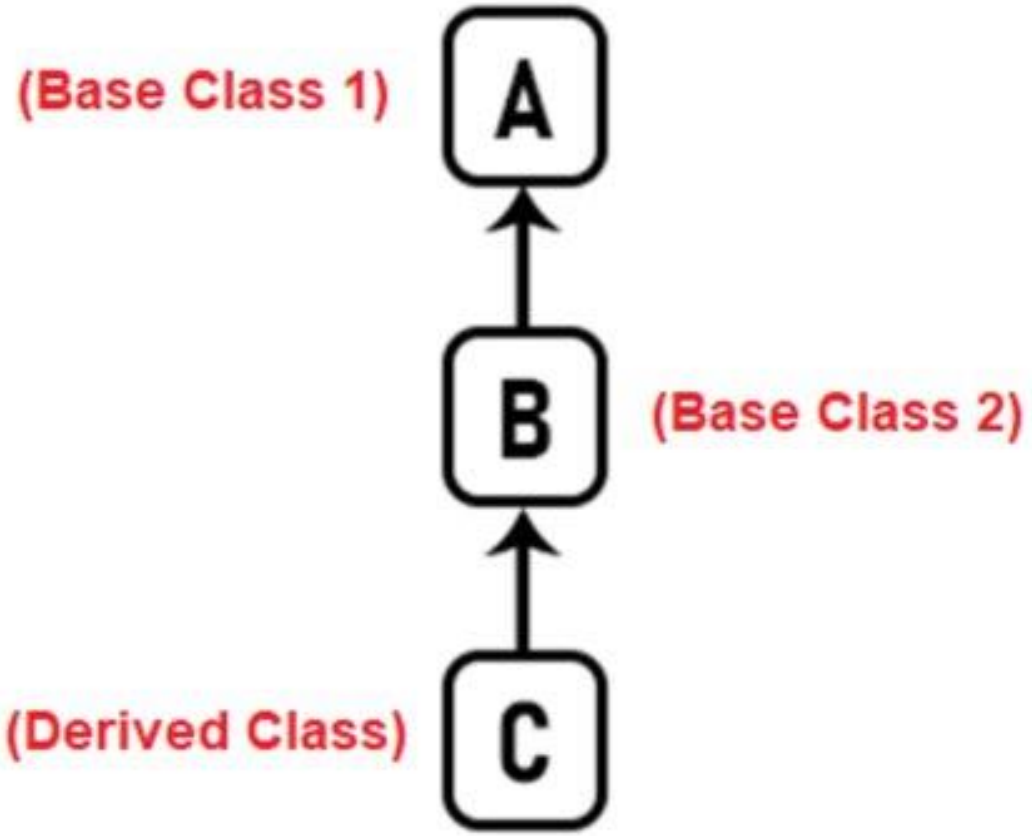
Hierarchical Inheritance in C++

In this type of inheritance, multiple derived classes inherits from a single base class.



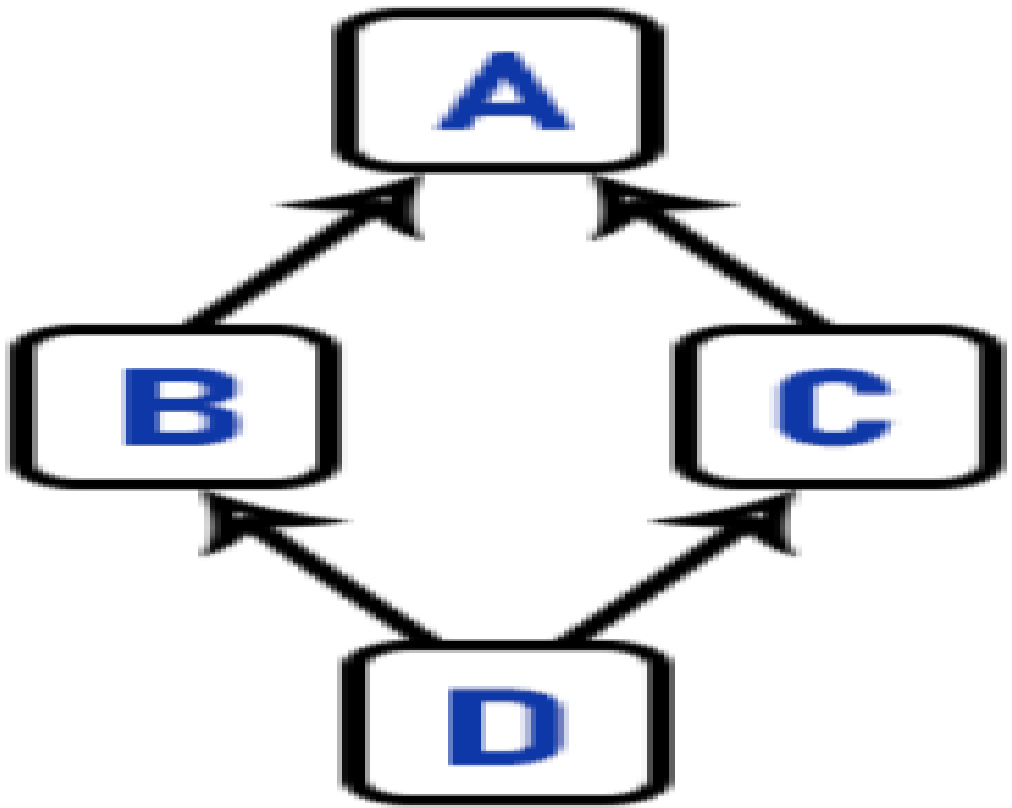
Multilevel Inheritance in C++

In this type of inheritance the derived class inherits from a class, which in turn inherits from some other class. The Super class for one, is sub class for the other.



Hybrid (Virtual) Inheritance in C++

Hybrid Inheritance is combination of Hierarchical and Mutilevel Inheritance.



Defining Derived Classes

- A derived class is specified by defining its relationship with the base class in addition to its own details.
- The general syntax of defining a derived class is as follows:

class derived_classname : Access specifier baseclass name

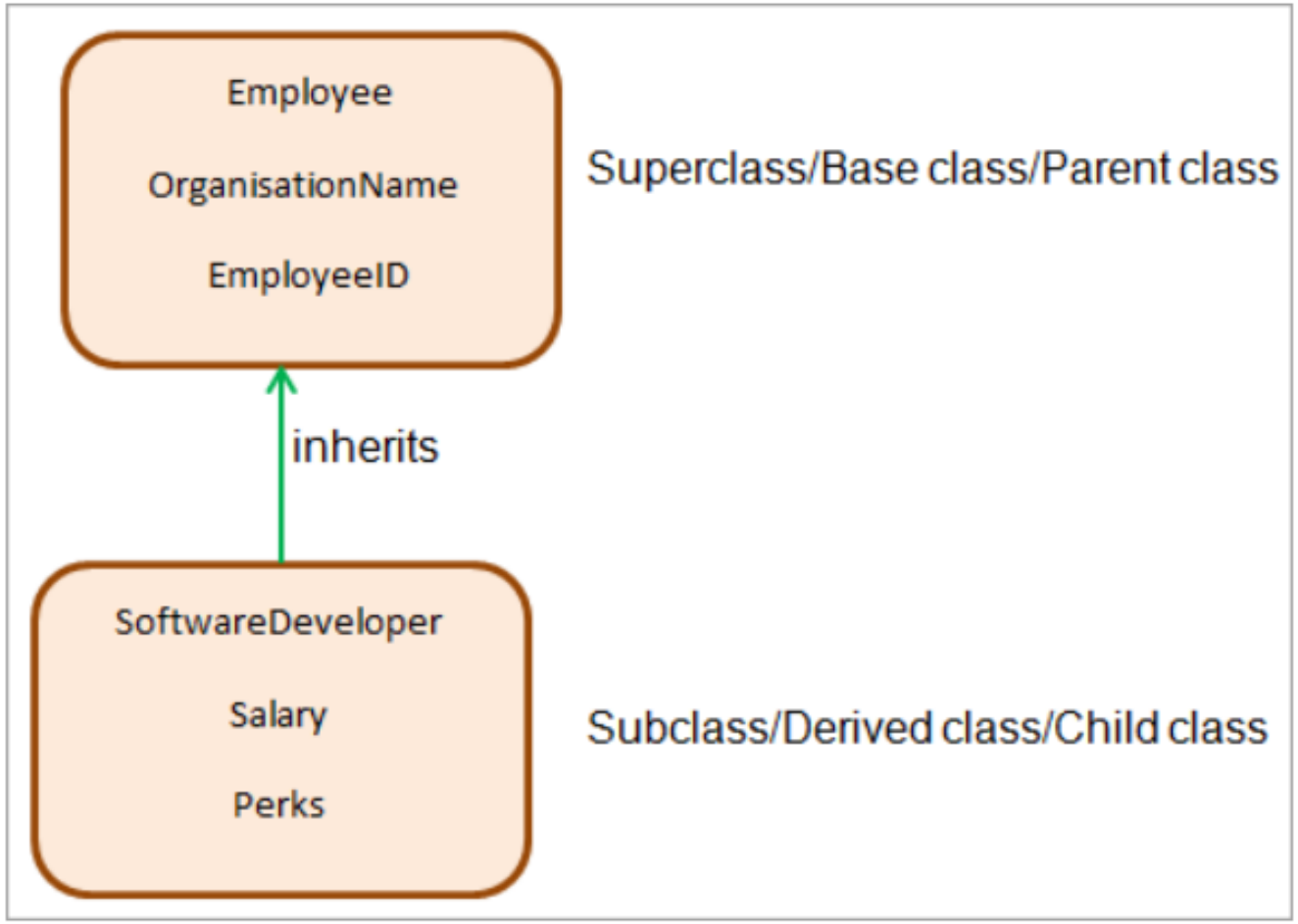
{ __

__ // members of derived class

};

- The access specifier or the visibility mode is optional
- By default it is private.

Example


Employee
OrganisationName
EmployeeID
Superclass/Base class/Parent class
inherits
SoftwareDeveloper
Salary
Perks
Subclass/Derived class/Child class

- **Access specifiers** in C++ define how the members of the class can be accessed.
  - public
  - private
  - Protected

- Private - Accessible only inside a class
- Protected - Accessible inside a class and inside derived classes
- Public - accessible inside class, inside derived class and upon object

Specifiers	Within Same Class	In Derived Class	Outside the Class
Private	Yes	No	No
Protected	Yes	Yes	No
Public	Yes	Yes	Yes

```
class employee //base class
{
   members of employee
};
class SoftwareDeveloper : public employee //public derivation
{
   Extra members of software developer
};
class  Testers: employee //private derivation (by default)
{
  Extra members of Testers
};
```

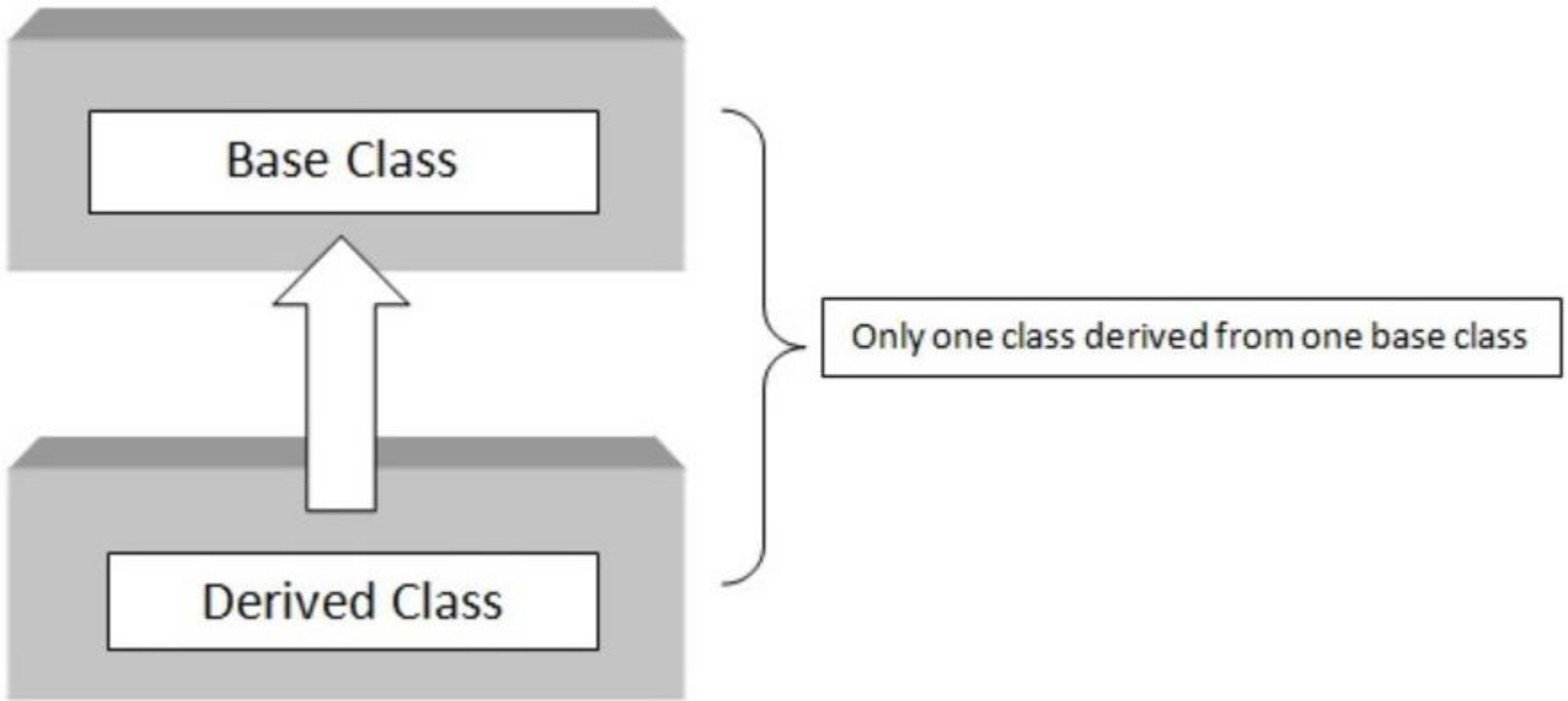
```
#include <iostream> //Within the derived class
using namespace std;
class A
{
int a;
public: int b;
 protected: int c;
public: A(){ a=1;b=2;c=3;}
public:
void fun()
{
cout<<"from function "<<a<<endl;
}};
class B:public A
{
public:
void dfun()
{
//cout<<a; //cant access
cout<<b;
cout<<c;
}};
int main()
{
B b1;
b1.dfun();}
```

```
 class Animal
{
   char color[10];    //can't inherit
public:
   int legs = 4;
   char type[10];  // omnivorous // carnivorous etc..
};

// Dog class inheriting Animal class
class Dog : public Animal
{
   public:
   int tail = 1;
   Dog() {  .....}
};
```

```
int main()
{
   Dog d;
   cout << d.legs;
   cout<<d.type;
   cout << d.tail;
}
```

Single Inheritance in C++

In this type of inheritance one derived class inherits from only one base class. It is the most simplest form of Inheritance.



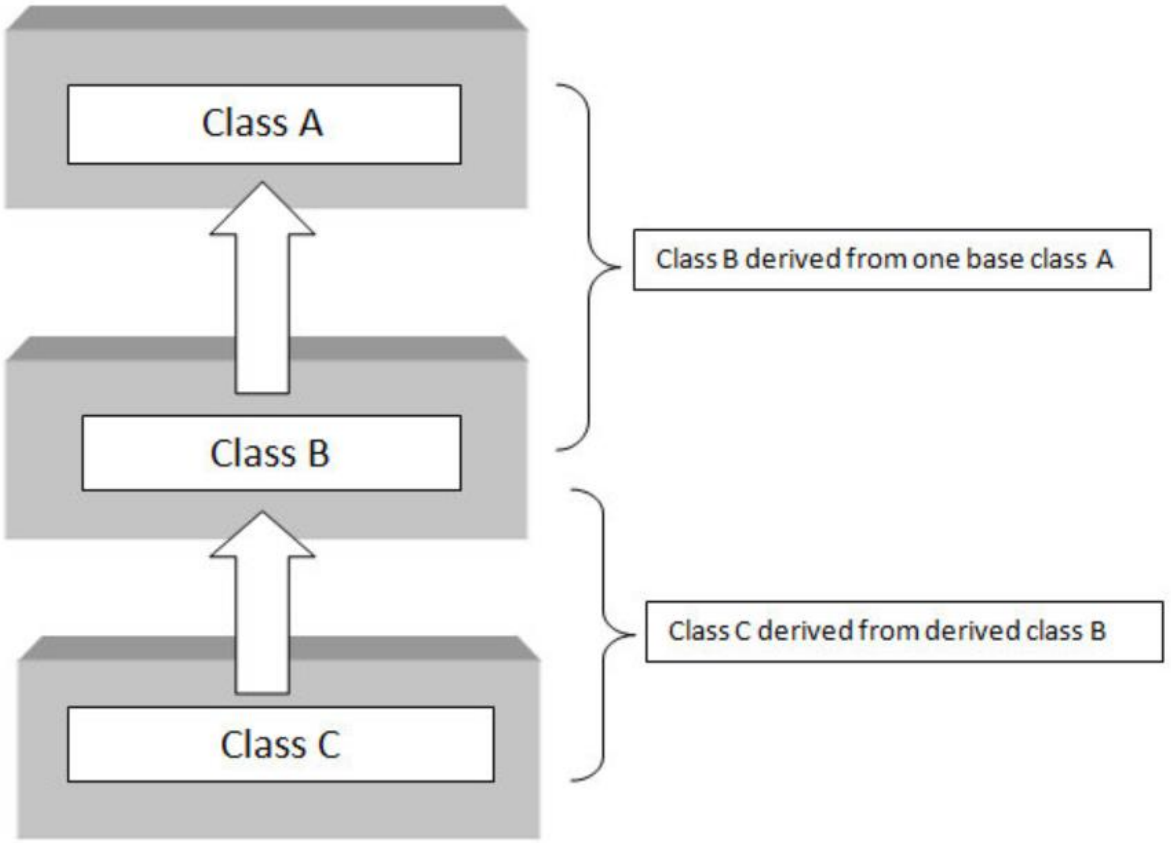
```
class A   // base class
{
    ..........
};
class B : access_specifier A   // derived class
{
    ..........
} ;
```

Getdata-X

Get Y & Product of X and Y

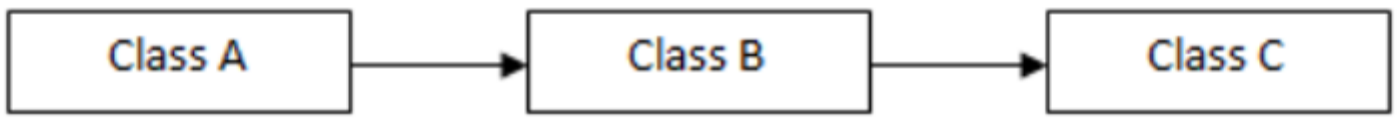
```
// inheritance.cpp
#include <iostream>
using namespace std;
class base    //single base class
{
  public:
    int x;
  void getdata()
  {
    cout << "Enter the value of x = "; cin >> x;
  }
 };
class derive : public base    //single derived class
{
  private:
   int y;

  public:
void readdata()
  {
    cout << "Enter the value of y = "; cin >> y;
  }
  void product()
  {
    cout << "Product = " << x * y;
  }
};


int main()
{
  derive a;     //object of derived class
  a.getdata();
  a.readdata();
  a.product();
  return 0;
}
```

Multilevel inheritance



```
class A // base class
{
    ..........
};
class B : acess_specifier A // derived class
{
    ..........
 } ;
 class C : access_specifier B // derived from derived class B
 {
    ..........
 } ;
```

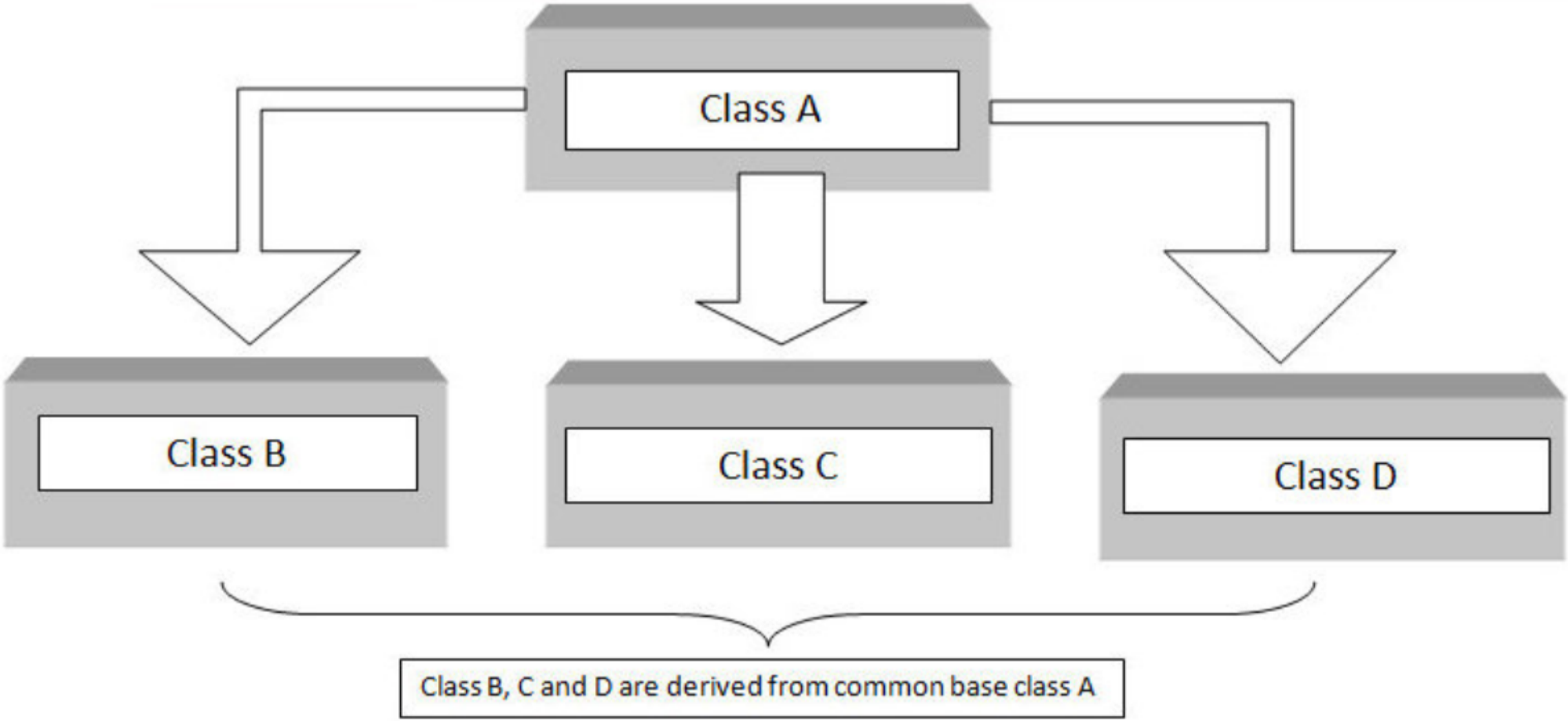
Multilevel Inheritance:



Get data x | Get data y | Get data Z and find product

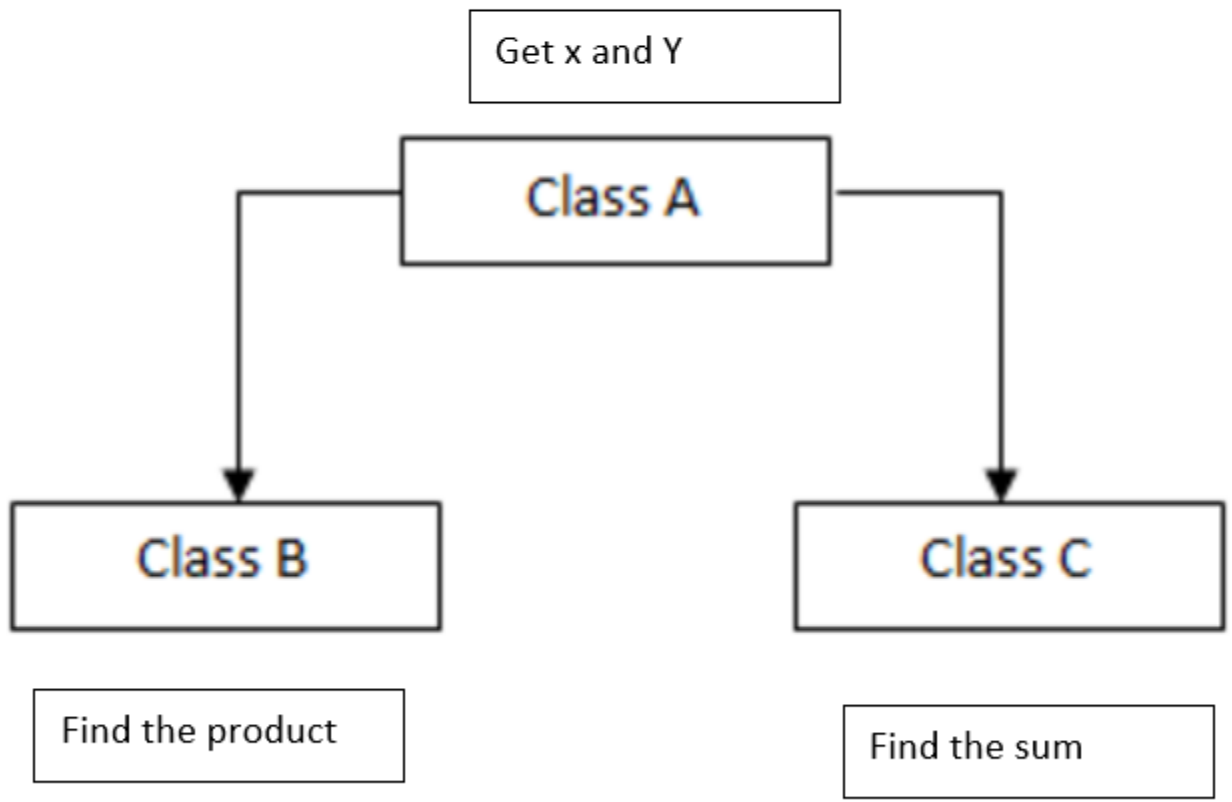
```
class base //single base class
{       public:
        int x;
        void getdata()
        {
        cout << "Enter value of x= "; cin >> x;
        }
};
class derive1 : public base
{       public:
        int y;
        void readdata()
        {
           cout << "\nEnter value of y= "; cin >> y;
        }
};
class derive2 : public derive1   // derived from class derive1
{
                private:
                int z;
                public:
                void indata()
                {               cout << "\nEnter value of z= "; cin >> z;
                }
                void product()
                {                       cout << "\nProduct= " << x * y * z;
                }
};
int main()
{
    derive2 a;      //object of derived class
    a.getdata();
    a.readdata();
    a.indata();
    a.product();
    return 0;
}               //end of program
```

C++ Hierarchical Inheritance Block Diagram



```
class A // base class
{
   ..............
};
class B : access_specifier A // derived class from A
{
   ...........
 } ;
 class C : access_specifier A // derived class from A
 {
   ...........
 } ;
 class D : access_specifier A // derived class from A
 {
    ...........
 } ;
```

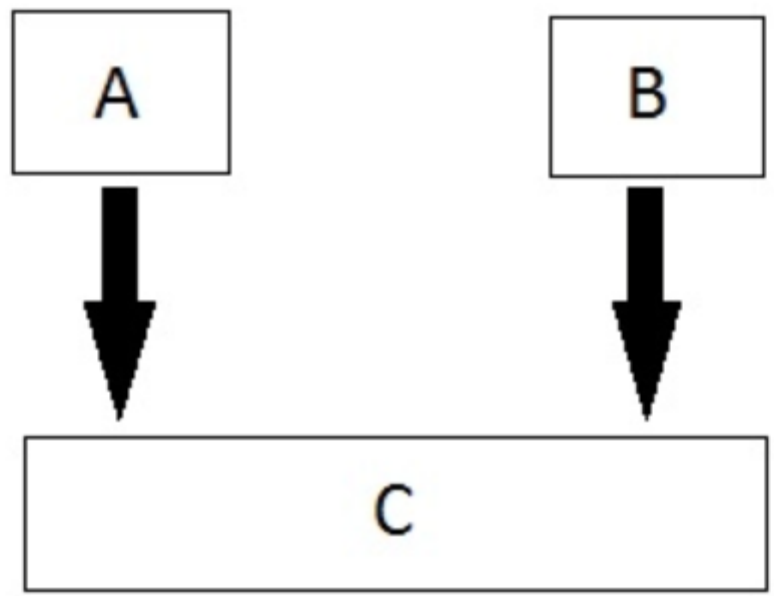
Hierarchical Inheritance:



```
// hierarchical inheritance
#include <iostream>
using namespace std;

class A //single base class
{
   public:
                int x, y;
                void getdata()
                {
                   cout << "\nEnter value of x and y:\n"; cin >> x >> y;
                }};
class B : public A //B is derived from class base
{    public:
                void product()
                {                   cout << "\nProduct= " << x * y;
                }
};
class C : public A //C is also derived from class base
{    public:
     void sum()
        {
          cout << "\nSum= " << x + y;
         }
};
int main()
{   B obj1;          //object of derived class B
    C obj2;         //object of derived class C
    obj1.getdata();
    obj1.product();
    obj2.getdata();
    obj2.sum();
    return 0;
}  //end of program
```

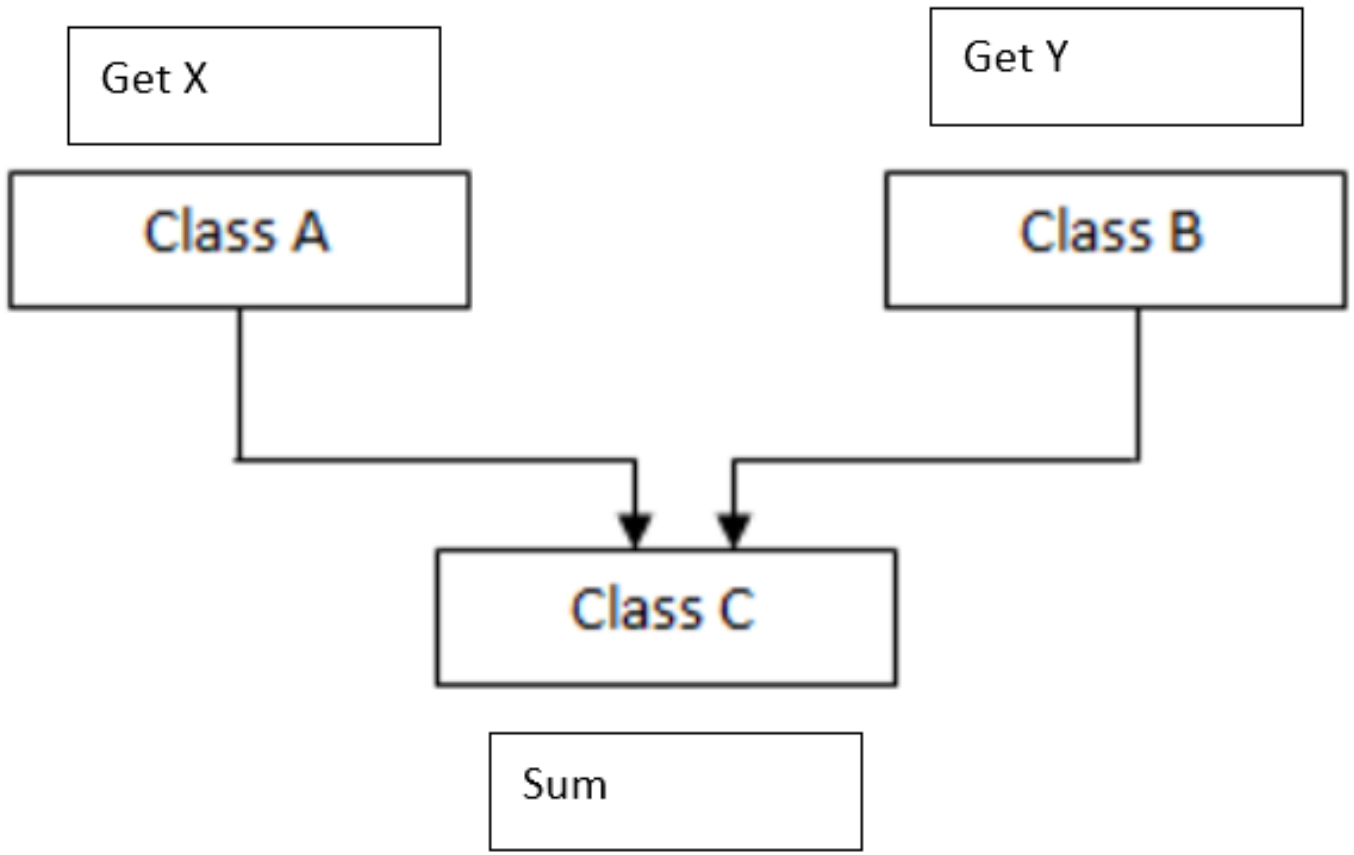
Multiple Inheritance in C++

In this type of inheritance a single derived class may inherit from two or more than two base classes.

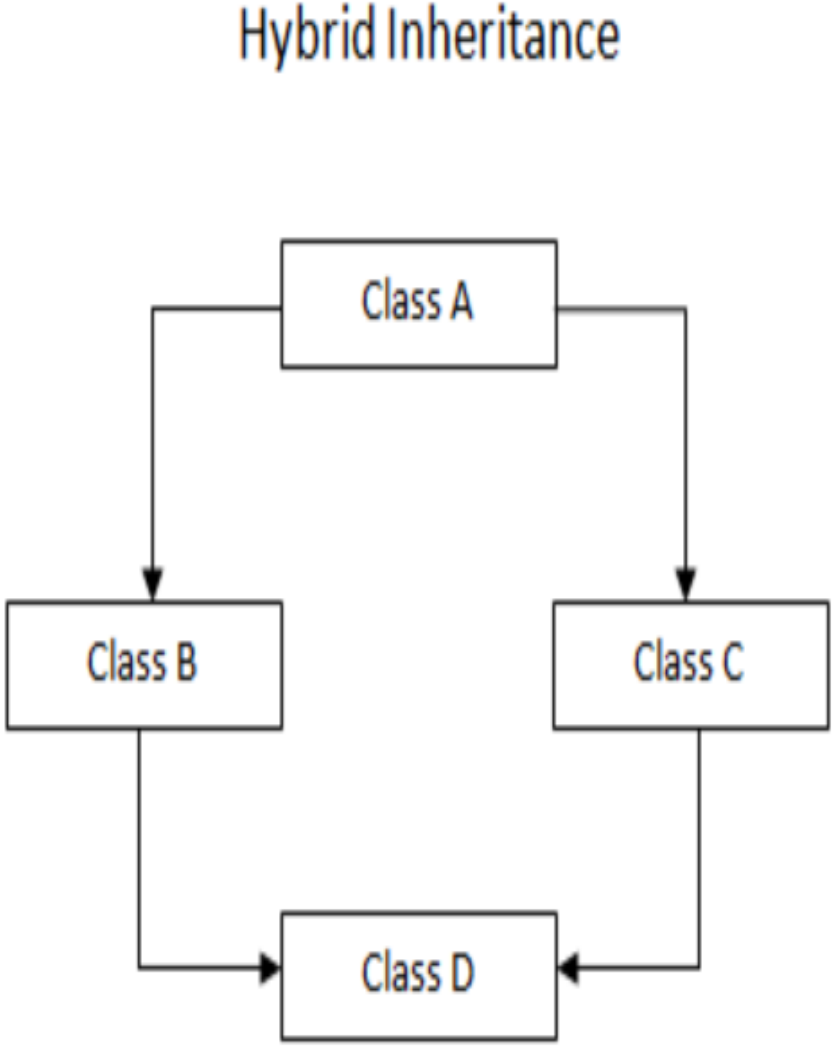


```
class A
{
  ..........
};
class B
{
  ..........
 } ;
class C : access_specifier A,access_specifier B // derived class from A and B
{
  ..........
} ;
```

Multiple Inheritances:



```
// multiple inheritance.cpp
class A
{       public:
        int x;
        void getx()
   {
          cout << "enter value of x: "; cin >> x;
   }
};
class B
{       public:
        int y;
        void gety()
        {
          cout << "enter value of y: "; cin >> y;
        }
};
class C : public A, public B   //C is derived from class A and class B
{
        public:
        void sum()
        {
          cout << "Sum = " << x + y;
        }
};

int main()
{
         C obj1; //object of derived class C
         obj1.getx();
         obj1.gety();
         obj1.sum();
         return 0;
}        //end of program
```

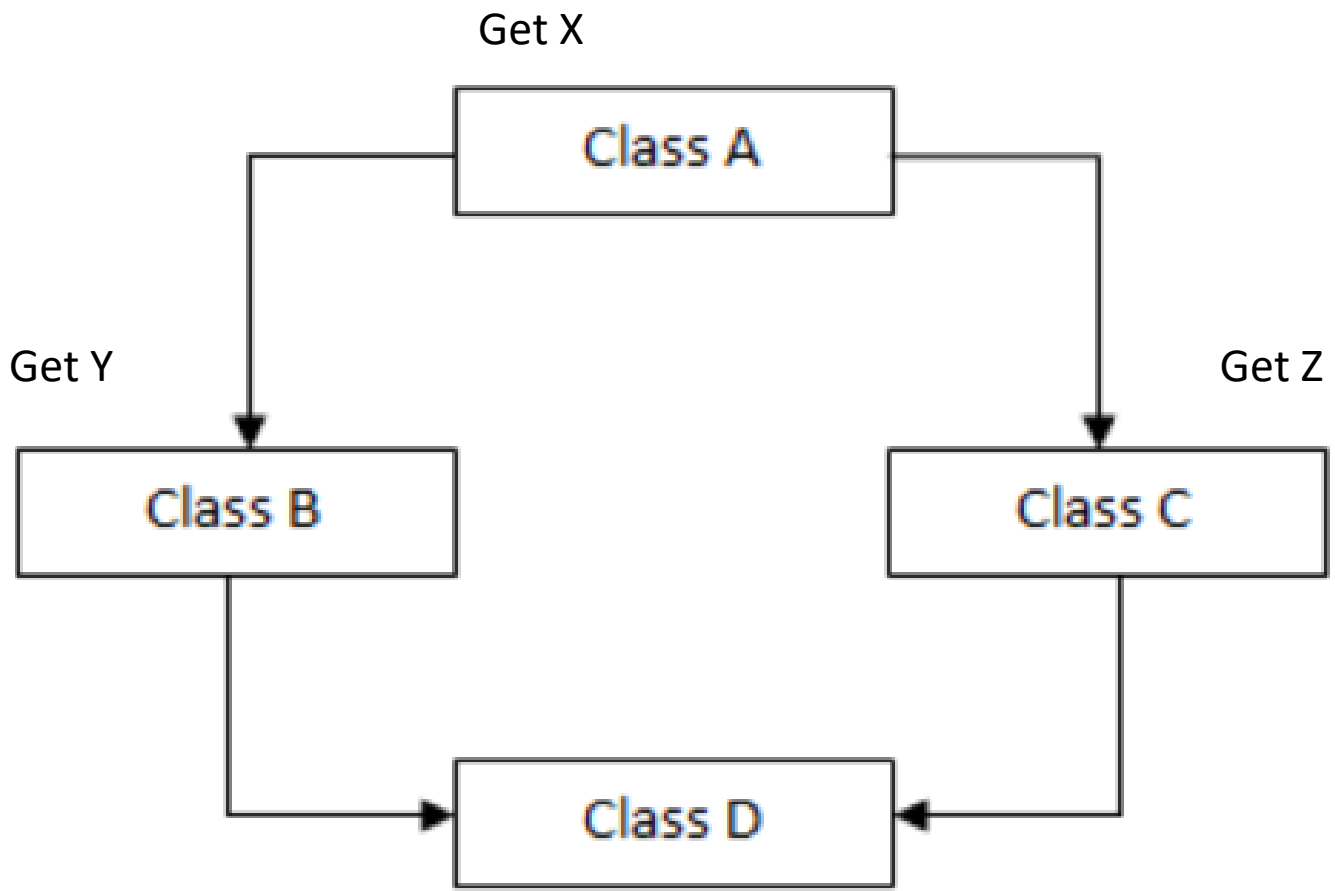


- Class B and class C are inherited from class A.
- Class D is inherited from two class B and class C.
- Indirectly class D is inherited from class A, twice, through class B and class C.
- Hence, class D will have two sets of copies of data members of class A. One through class B and the other through class C.

```
class A
{
  Body of class A
};
class B : virtual public A
{
  Body of class B
};

class C : virtual public A
{
  Body of class C
};
class D : public B, public C
{
  Body of class C
};
```

When class is made virtual base class then special care is taken by c++ to make sure that only one copy of that class is inherited regardless of how many inheritance paths exists between the virtual base class and derived class.

Hybrid Inheritance



Find product X*Y*Z

Access Control and Inheritance

Table showing all the Visibility Modes

	Derived Class	Derived Class	Derived Class
Base class	**Public Mode**	**Private Mode**	**Protected Mode**
Private	Not Inherited	Not Inherited	Not Inherited
Protected	Protected	Private	Protected
Public	Public	Private	Protected

•**public inheritance** makes public members of the base class public in the derived class, and the protected members of the base class remain protected in the derived class.

```
class Base {
    public:
        int x;
    protected:
        int y;
    private:
        int z;
};
class PublicDerived: public Base {
    // x is public
    // y is protected
    // z is not accessible from PublicDerived
};
class ProtectedDerived: protected Base {
    // x is protected
    // y is protected
    // z is not accessible from ProtectedDerived
};

class PrivateDerived: private Base {
    // x is private
    // y is private
    // z is not accessible from PrivateDerived
}
```

- How to access private member of base class from derived class?

```
// C++ program to demonstrate the working of
public inheritance
#include <iostream>
using namespace std;
class Base {
   int pvt ;
   protected:
    int prot ;
   public:
    int pub;
      Base(){pvt=prot=pub=1;}
    // function to access private member
    int getPVT() {
        return pvt;
    }
};
class PublicDerived : public Base {
   public:
    // function to access protected member from Base
    int getProt() {
        return prot;
    }
};

int main() {
    PublicDerived object1;
    cout << "Private = " << object1.getPVT() << endl;
    cout << "Protected = " << object1.getProt() << endl;
    cout << "Public = " << object1.pub << endl;
    return 0;
}
```

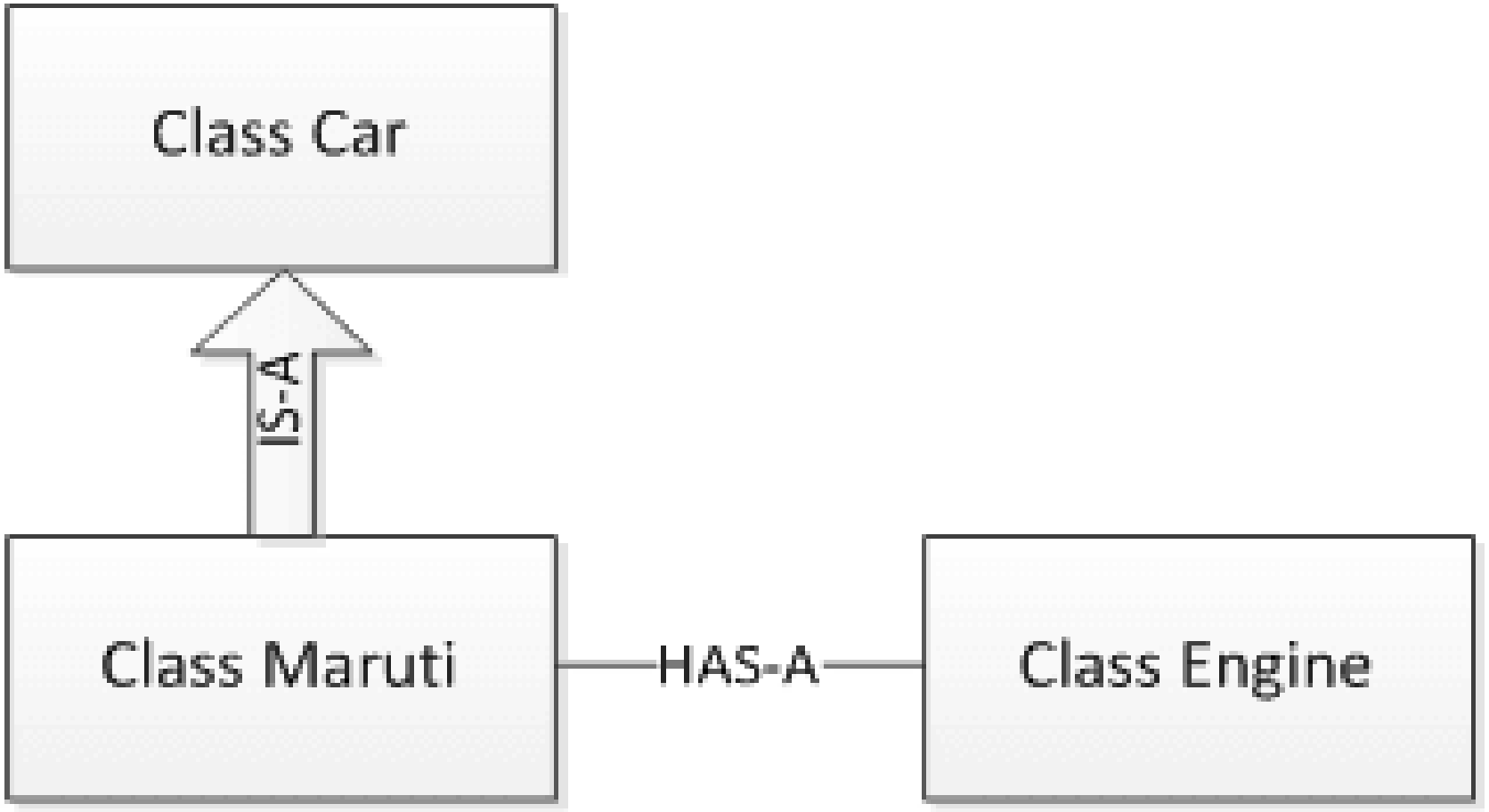
IS- A relationship

- the concept of IS-A is a totally based on Inheritance.
- It is just like saying "A is a B type of thing".
- For example, Apple is a Fruit, Car is a Vehicle etc.
- Inheritance is uni-directional.
- For example, House is a Building. But Building is not a House.

HAS-A Relationship:

- Composition(HAS-A) simply mean the use of instance variables that are references to other objects. For example Maruti has Engine, or House has Bathroom.

IS-A and HAS-A



```
#include <iostream>
using namespace std;
class car
{
   private:
   string color;
   int maxspeed;
   public:
   void read()
   {
      cin>>color>>maxspeed;
   }
   void carInfo(){
      cout<<color<<maxspeed;
   }
   };
```

```
class Engine {
   public:
   void start(){
      cout<<"Engine Started:";
   }
   void stop(){
      cout<<"Engine Stopped:";
   }
};
class Maruti:public car{

   public:
   void MarutiStartDemo(){
   Engine MarutiEngine;
   MarutiEngine.start();
      }
   };
```

```
int main()
{
   Maruti M;
   M.read();
   M.carInfo();
   M.MarutiStartDemo();
   return 0;
}
```

```
class Circle
{
  // 1. Internal private fields of class
private:
  double x, y; // Circle center coordinates
  double r; // radius of the circle


class CircleColor : public Circle
{
  // 1. Internal variables of the class
private:
  unsigned int color = 0;
```

```
// Class that describes a triangle
class Triangle
{
  // Methods and fields of Triangle class
  // ...
};

// A class that implements a circle
class Circle
{
  // Methods and fields of Circle class
  // ...
};

// A class that implements various geometric figures.
// Used type of relationship - aggregation.
class Figures
{
  Triangle tr[10]; // array of triangles
  unsigned int n_tr; // the number of triangles in the array tr
  Circle cr[10]; // array of circles
  unsigned int n_cr; // the number of circles in the array cr

  // Other fields and class methods
  // ...
};
```