

Object Oriented  
Programming with C++

# Unit-5

## Inheritance

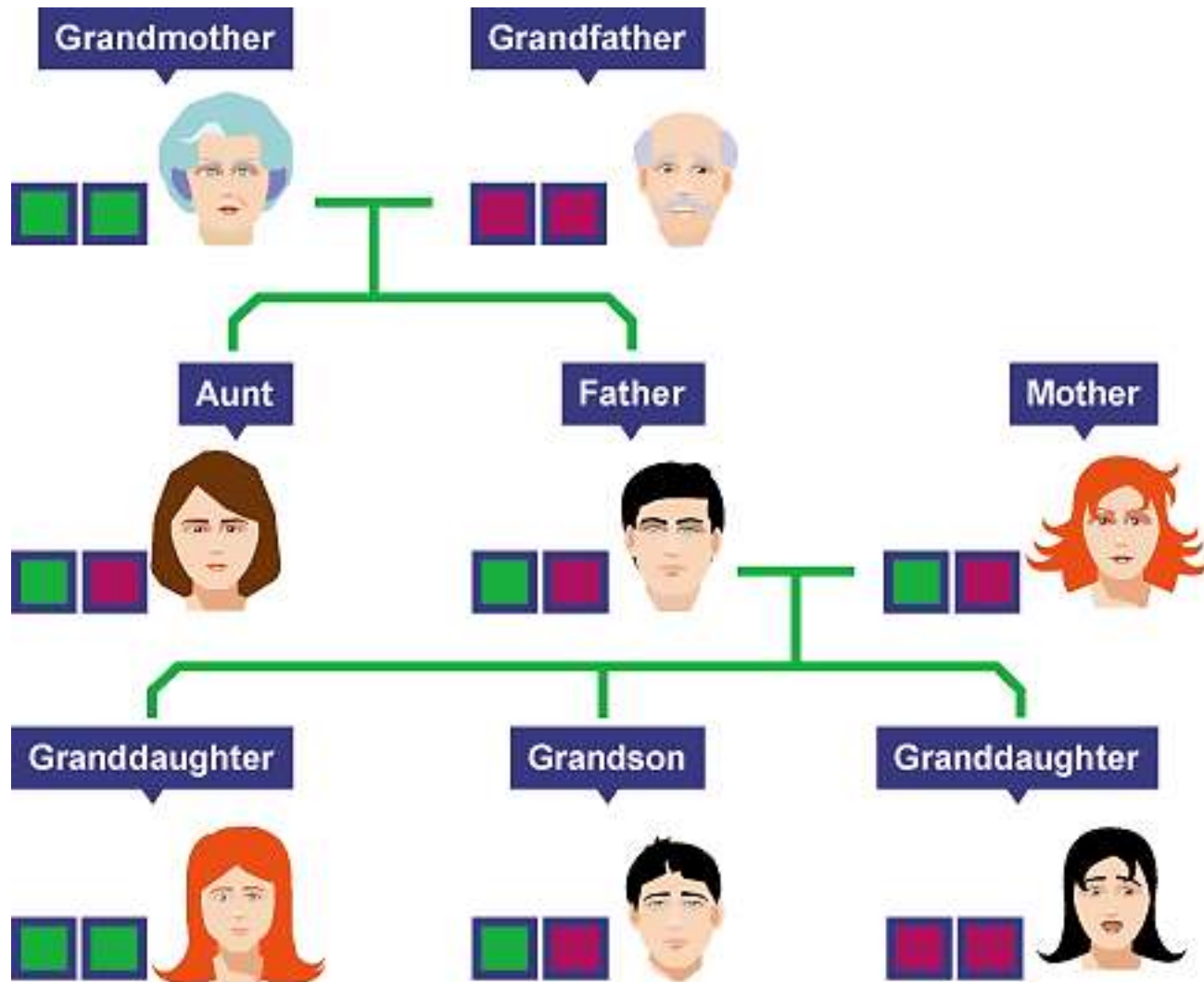


# Outline

---

- Concept of inheritance
- Types of inheritance
  1. Single
  2. Multiple
  3. Multilevel
  4. Hierarchical
  5. Hybrid
- Protected members
- Overriding
- Virtual base class

# Inheritance



# Concept of Inheritance

## class Doctor

### Attributes:

Age, Height, Weight

### Methods:

Talk()

Walk()

Eat()

Diagnose()

## class Footballer

### Attributes:

Age, Height, Weight

### Methods:

Talk()

Walk()

Eat()

Playfootball()

## class Businessman

### Attributes:

Age, Height, Weight

### Methods:

Talk()

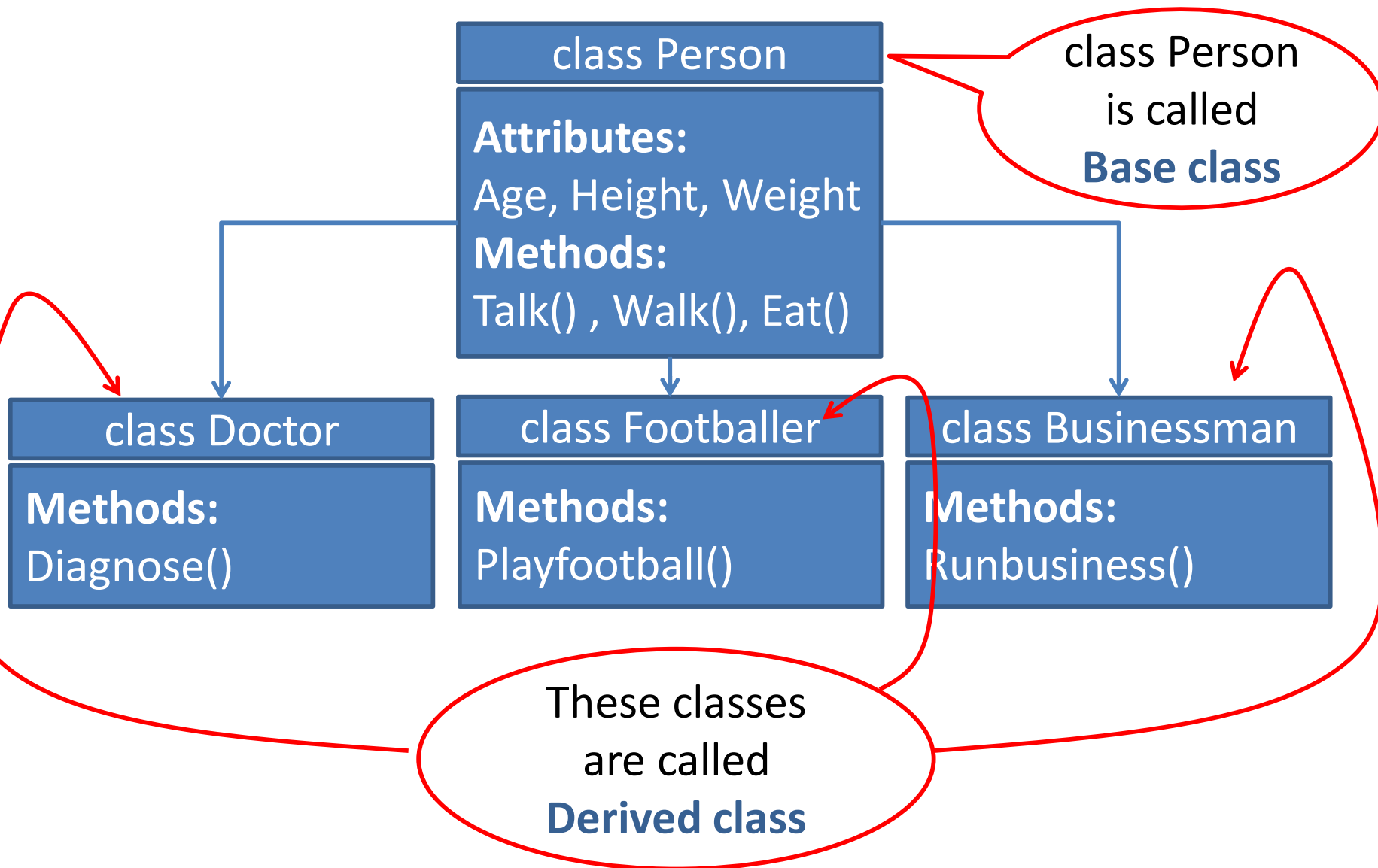
Walk()

Eat()

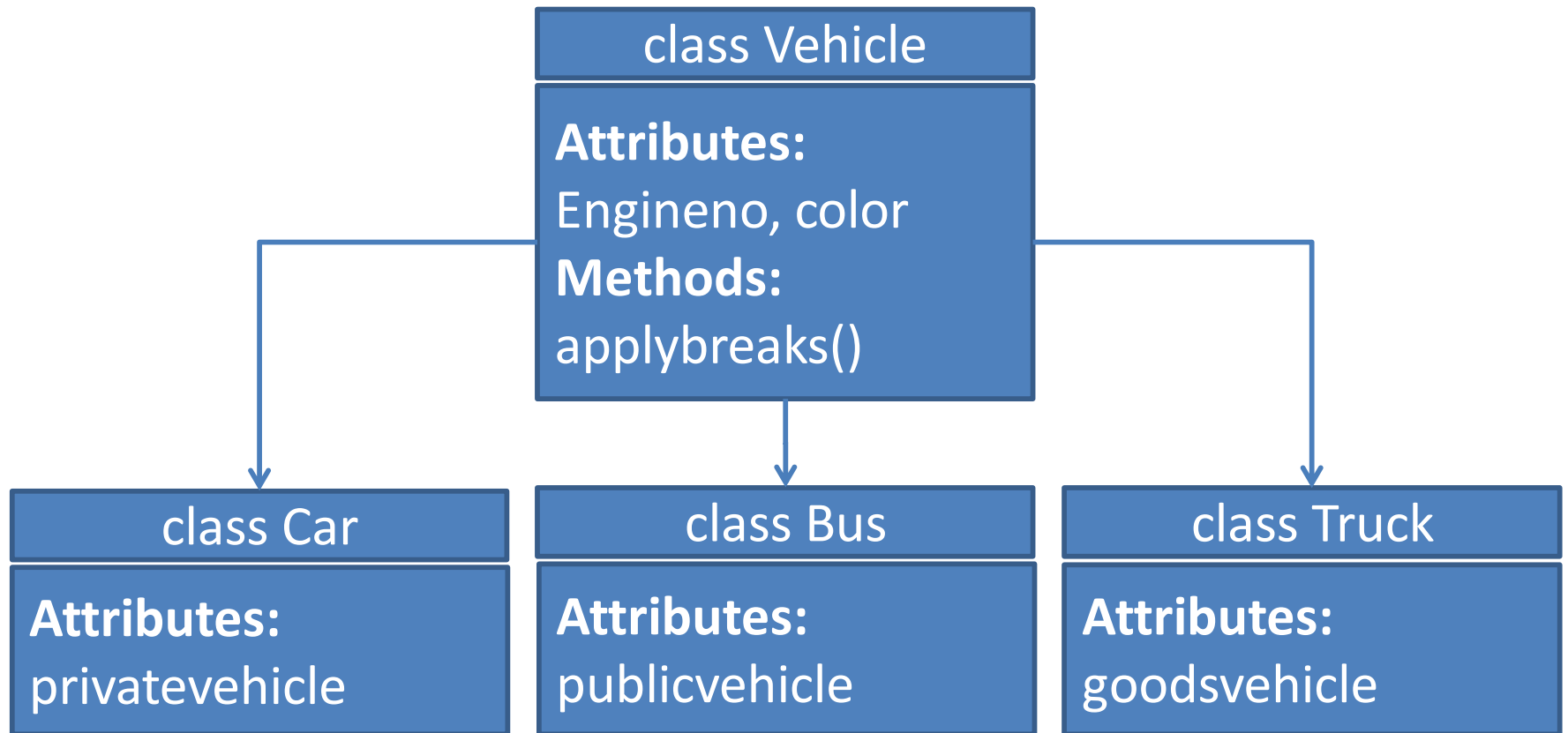
Runbusiness()

- All of the classes have common attributes (Age, Height, Weight) and methods (Walk, Talk, Eat).
- However, they have some special skills like Diagnose, Playfootball and Runbusiness.
- In each of the classes, you would be copying the same code for Walk, Talk and Eat for each character.

# Concept of Inheritance(Cont...)



# Concept of Inheritance(Cont...)



# Inheritance

- **Inheritance** is the process, by which class can acquire(reuse) the properties and methods of another class.
- The mechanism of deriving a new class from an old class is called **inheritance**.
- The new class is called **derived class** and old class is called **base class**.
- The derived class may have all the features of the base class and the programmer can add new features to the derived class.

# Syntax to Inherit class

Syntax:

```
class derived-class-name : access-mode base-class-name
{
    // body of class
};
```

Example:

```
class person{
    //body of class
};
class doctor:private person
{
    //body of class
}
```

The diagram illustrates the syntax of class inheritance with the following annotations:

- Base Class:** A bracket points to the `person` class definition.
- Access Mode:** A bracket points to the `private` access specifier in the `doctor` class definition.
- Derived Class:** A bracket points to the `doctor` class definition.
- By default access mode is private:** A red box highlights this text, indicating the default access mode for inheritance.

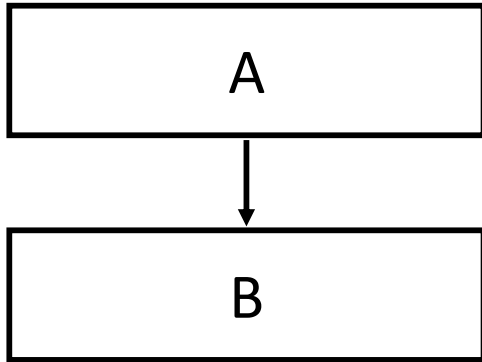


# Types of Inheritance

---

1. Single Inheritance
2. Multilevel Inheritance
3. Multiple Inheritance
4. Hierarchical Inheritance
5. Hybrid Inheritance (also known as Virtual Inheritance)

# 1. Single Inheritance



- If a class is derived from a single class then it is called **single inheritance**.
- Class **B** is derived from class **A**

Example:

```
class Animal
{   public:
    int legs = 4;
};
class Dog : public Animal
{   public:
    int tail = 1;
};
```

# Single Inheritance Program

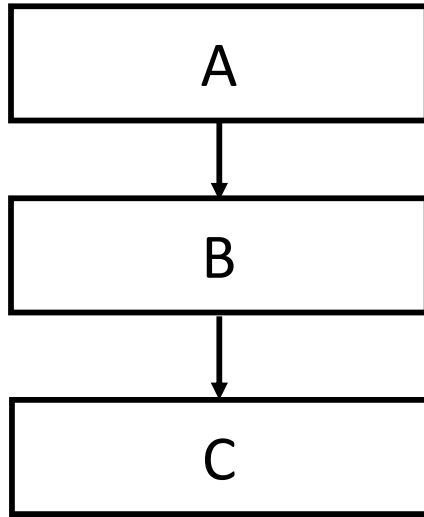
```
class Animal{
    int legs=4;
    public:
        void display1(){
            cout<<"\nLegs="<<legs;
        }
};

class Dog : public Animal{
    bool tail = true;
    public:
        void display2(){
            cout<<"\nTail="<<tail;
        }
};
```

```
int main()
{
    Animal a1;
    Dog d1;
    d1.display1();
    d1.display2();
}
```

Output:  
Legs=4  
Tail=1

## 2. Multilevel Inheritance



- Any class is derived from a class which is derived from another class then it is called **multilevel inheritance**.
- Here, class **C** is derived from class **B** and class **B** is derived from class **A**, so it is called **multilevel inheritance**.

Example:

```
class Person
{
    //content of class person
};
class Student :public Person
{
    //content of Student class
};
```

```
class ITStudent :public
Student
{
    //content of ITStudent
class
};
```

```

class Person{
    public:
        void display1(){
            cout<<"\nPerson class";
        }
};
class Student:public Person{
    public:
        void display2(){
            cout<<"\nStudent class";
        }
};
class ITStudent:public Student{
    public:
        void display3(){
            cout<<"\nITStudent class";
        }
};

```

```

int main()
{
    Person p;
    Student s;
    ITStudent i;
    p.display1();
    s.display2();
    s.display1();
    i.display3();
    i.display2();
    i.display1();
}

```

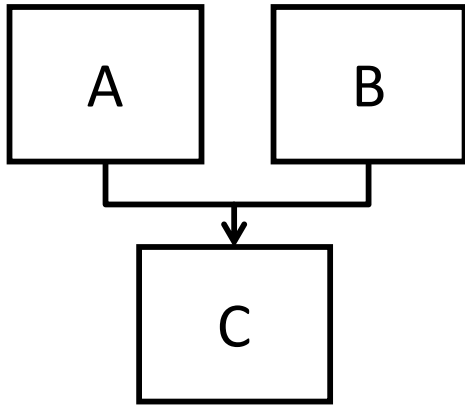
### Output:

```

Person class
Student class
Person class
ITStudent class
Student class
Person class

```

# 3. Multiple Inheritance



- If a class is derived from more than one class then it is called **multiple inheritance**.
- Here, class **C** is derived from two classes, class **A** and class **B**.

Example:

```
class Liquid
{
    //content of Liquid class
};
class Fuel
{
    //content of Fuel class
};
```

```
class Petrol: public
Liquid, public Fuel
{
    //content of
    Petrol class
};
```

Arrows indicate the inheritance relationship: one arrow points from the 'Liquid' class definition to the 'Liquid' base class in the 'Petrol' definition, and another arrow points from the 'Fuel' class definition to the 'Fuel' base class in the 'Petrol' definition.

```

class Liquid{
    public:
        void display1(){
            cout<<"\nLiquid class";
        }
};
class Fuel{
    public:
        void display2(){
            cout<<"\nFuel class";
        }
};
class Petrol:public Liquid,public Fuel{
    public:
        void display3(){
            cout<<"\nPetrol class";
        }
};

int main()
{
    Liquid l;
    Fuel f;
    Petrol p;
    l.display1();
    f.display2();
    p.display3();
    p.display2();
    p.display1();
}

```

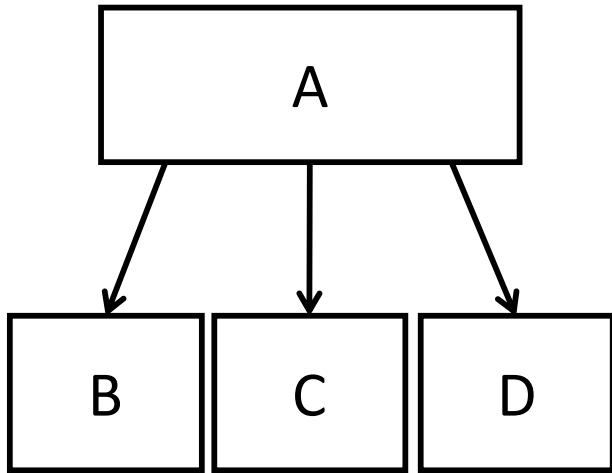
### Output:

```

Liquid class
Fuel class
Petrol class
Fuel class
Liquid class

```

# 4. Hierarchical Inheritance



- If one or more classes are derived from one class then it is called **hierarchical inheritance**.
- Here, class **B**, class **C** and class **D** are derived from class **A**.

Example:

```
class Animal
{
//content of class Animal
};
class Elephant :public Animal
{
//content of class Elephant
};
```

```
class Horse :public Animal
{
//content of class Horse
};
class Cow :public Animal
{
//content of class Cow
};
```



```
class Animal{
    public:
    void display1(){
        cout<<"\nAnimal Class";
    }
};
class Elephant:public Animal{
    public:
    void display2(){
        cout<<"\nElephant class";
    }
};
```

```
class Horse:public Animal{
    public:
    void display3(){
        cout<<"\nHorse class";
    }
};
class Cow:public Animal{
    public:
    void display4(){
        cout<<"\nCow class";
    }
};
```

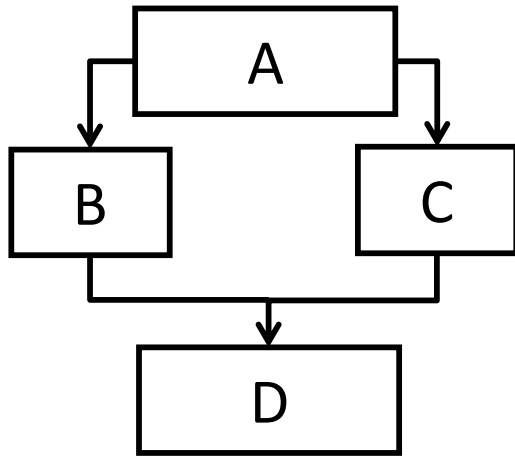
```
int main(){
    Animal a; Elephant e; Horse h; Cow c;
    a.display1();
    e.display2(); e.display1();
    h.display3(); h.display1();
    c.display4(); c.display1();
}
```

### Output:

```
Animal Class
Elephant class
Animal Class
Horse class
Animal Class
Cow class
Animal Class
```

# 5. Hybrid Inheritance

---



- It is a combination of any other inheritance types. That is either multiple or multilevel or hierarchical or any other combination.
- Here, class **B** and class **C** are derived from class **A** and class **D** is derived from class **B** and class **C**.
- class **A**, class **B** and class **C** is example of **Hierarchical Inheritance** and class **B**, class **C** and class **D** is example of Multiple Inheritance so this hybrid inheritance is combination of Hierarchical and Multiple Inheritance.

# Hybrid Inheritance (Cont...)

```
class Car
{
    //content of class Car
};
class FuelCar:public Car
{
    //content of class FuelCar
};
Class ElectricCar:public Car
{
    //content of class ElectricCar
};
Class HybridCar:public FuelCar, public ElectricCar
{
    //content of classHybridCar
};
```

The diagram illustrates hybrid inheritance with four classes: Car, FuelCar, ElectricCar, and HybridCar. Blue arrows indicate the inheritance relationships: one arrow points from Car to FuelCar, another from Car to ElectricCar, and a third from FuelCar to HybridCar. This shows that HybridCar inherits from both FuelCar and ElectricCar, while FuelCar and ElectricCar both inherit from Car.

```
class Car{
    public:
    void display1(){
        cout<<"\nCar class";
    }
};

class FuelCar:public Car{
    public:
    void display2(){
        cout<<"\nFuelCar class";
    }
};
```

```
class ElecCar:public Car{
    public:
    void display3(){
        cout<<"\nElecCar class";
    }
};

class HybridCar:public
FuelCar, public ElecCar{
    public:
    void display4(){
        cout<<"\nHybridCar class";
    }
};
```

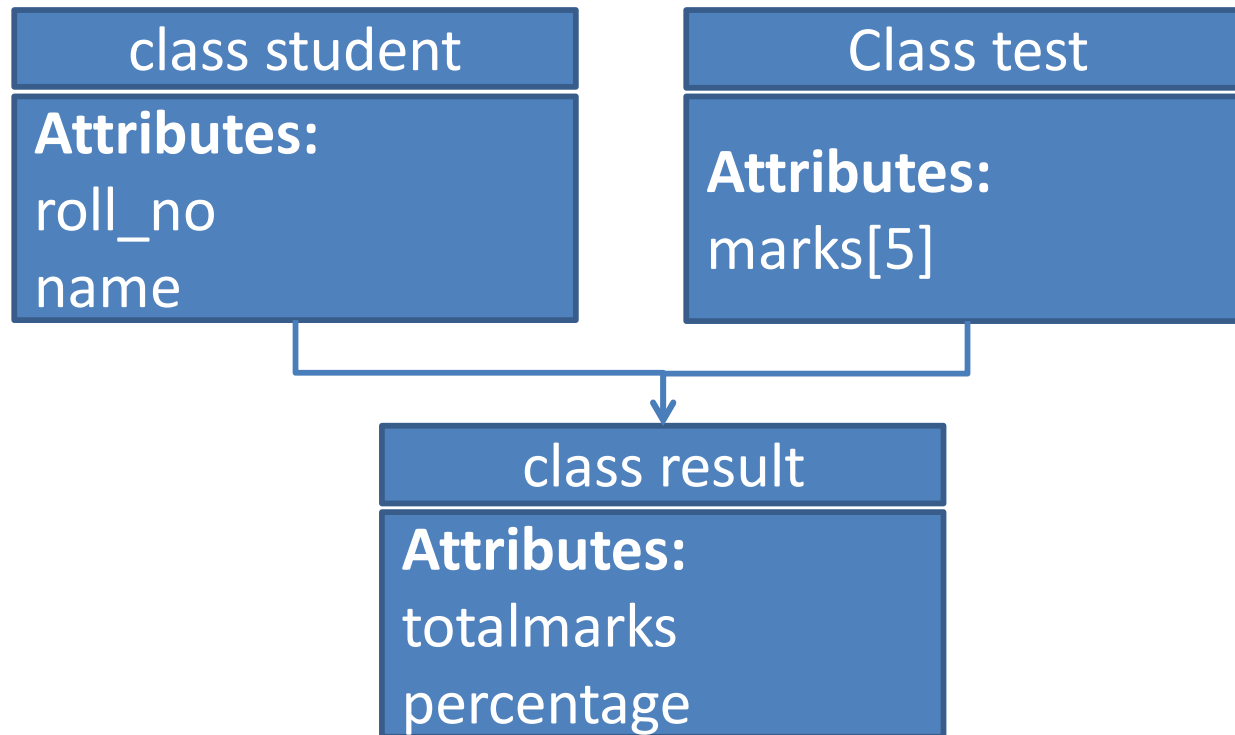
```
int main(){
    Car c; FuelCar f; ElecCar e;
    HybridCar h;
    h.display4();
    h.display3();
    h.display2();
}
```

**Output:**

HybridCar class  
ElecCar class  
FuelCar class

# GTU Program

- Create a class student that stores roll\_no, name. Create a class test that stores marks obtained in five subjects. Class result derived from student and test contains the total marks and percentage obtained in test. Input and display information of a student.



# Protected access modifier

- **Protected** access modifier plays a key role in inheritance.
- **Protected** members of the class can be accessed within the class and from derived class but cannot be accessed from any other class or program.
- It works like public for derived class and private for other class.

```
class ABC {
public:
    void setProtMemb(int i){
        m_protMemb = i; }
    void Display(){
        cout<<m_protMemb<<endl;}
protected:
    int m_protMemb;
    void Protfunc(){
        cout<<"\nAccess allowed\n";}
};
```

```
class XYZ : public ABC{
public:
    void useProtfunc(){
        Protfunc(); }
};
```

```
int main() {
    ABC a; XYZ x;
    a.m_protMemb;           //error, m_protMemb is protected
    a.setProtMemb(0);       //OK,uses public access function
    a.Display();
    a.Protfunc();           //error, Protfunc() is protected
    x.setProtMemb(5);       //OK,uses public access function
    x.Display();
    x.useProtfunc();} // OK, uses public access function
```

# Class access modifiers

---

- **public** – Public members are visible to all classes.
- **private** – Private members are visible only to the class to which they belong.
- **protected** – Protected members are visible only to the class to which they belong, and derived classes.



# Access modifiers

Base class  
members

How inherited base class  
members  
appear in derived class

```
private: x  
protected: y  
public: z
```

public  
base class

```
x is inaccessible  
protected: y  
public: z
```

```
private: x  
protected: y  
public: z
```

private  
base class

```
x is inaccessible  
private: y  
private: z
```

```
private: x  
protected: y  
public: z
```

protected  
base class

```
x is inaccessible  
protected: y  
protected: z
```

```
class base{
    private:
        int z;
    public:
        int x;
    protected:
        int y;
};
```

```
class privateDerived: private base
{
    // x is private
    // y is private
    // z is not accessible from
    privateDerived
};
```

```
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
};
```

# Inheritance using Public Access

**class Grade**

**private members:**

```
char letter;  
float score;  
void calcGrade();
```

**public members:**

```
void setScore(float);  
float getScore();  
char getLetter();
```

**class Test : public Grade**

**private members:**

```
int numQuestions;  
float pointsEach;  
int numMissed;
```

**public members:**

```
Test(int, int);
```

**private members:**

```
int numQuestions;  
float pointsEach;  
int numMissed;
```

**public members:**

```
Test(int, int);  
void setScore(float);  
float getScore();  
char getLetter();
```

When Test class inherits  
from Grade class using  
public class access, it  
looks like this: →

# Inheritance using Private Access

**class Grade**

**private members:**

```
char letter;  
float score;  
void calcGrade();
```

**public members:**

```
void setScore(float);  
float getScore();  
char getLetter();
```

**class Test : private Grade**

**private members:**

```
int numQuestions;  
float pointsEach;  
int numMissed;
```

**public members:**


```
Test(int, int);
```

**private members:**

```
int numQuestions;  
float pointsEach;  
int numMissed;  
void setScore(float);  
float getScore();  
float getLetter();
```

**public members:**

```
Test(int, int);
```

When Test class inherits  
from Grade class using  private class access, it  
looks like this:

# Inheritance using Protected Access

**class Grade**

**private members:**

```
char letter;  
float score;  
void calcGrade();
```

**public members:**

```
void setScore(float);  
float getScore();  
char getLetter();
```

**class Test : protected Grade**

**private members:**

```
int numQuestions;  
float pointsEach;  
int numMissed;
```

**public members:**

```
Test(int, int);
```

**private members:**


```
int numQuestions;  
float pointsEach;  
int numMissed;
```

**public members:**

```
Test(int, int);
```

**protected members:**

```
void setScore(float);  
float getScore();  
float getLetter();
```

When Test class inherits  
from Grade class using   
protected class access, it  
looks like this:

# Visibility of inherited members

Base class visibility	Derived class visibility		
	Public derivation	Private derivation	Protected derivation
Private			
Protected			
Public			

# Function Overriding / Method Overriding

---

- If base class and derived class have member functions with same name and arguments then method is said to be **overridden** and it is called **function overriding** or **method overriding** in C++.

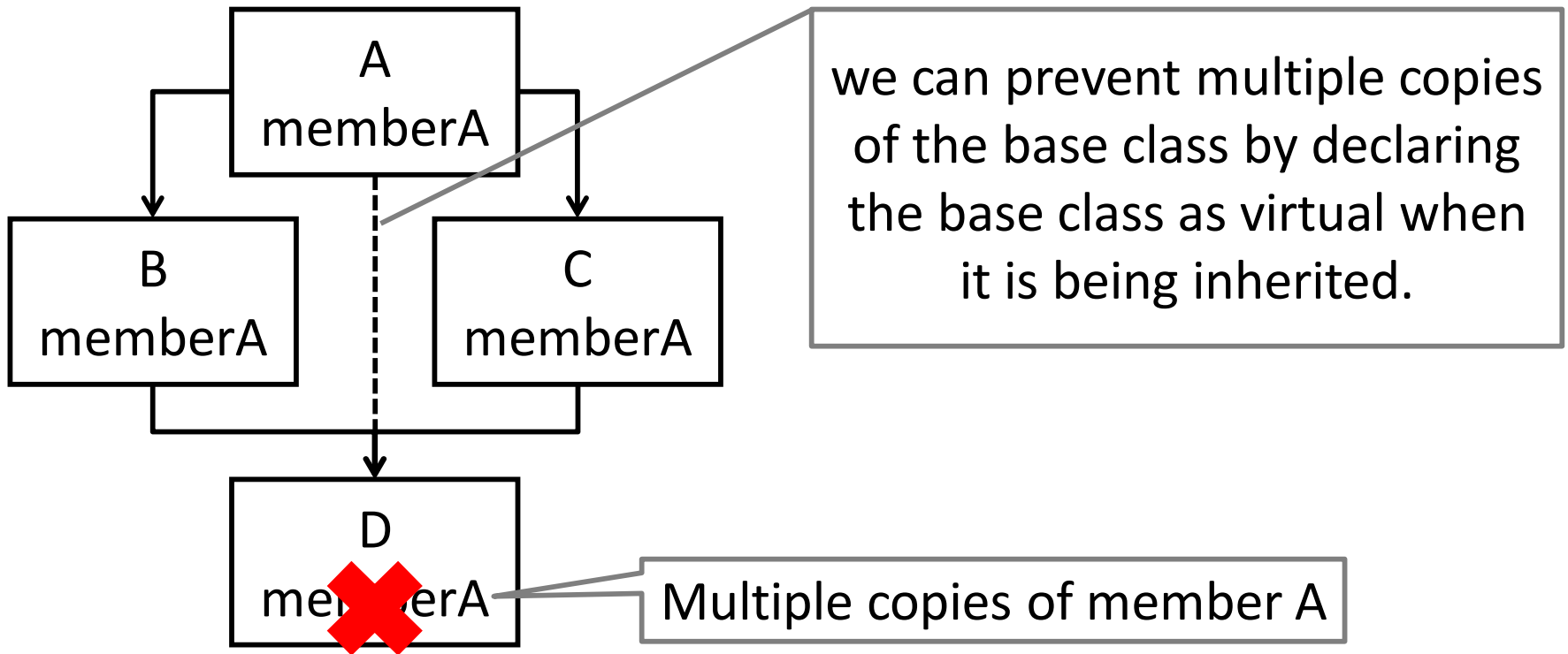
```
class ABC
{
    public:
    void display(){
        cout<<"This is parent class";
    }
};

class XYZ:public ABC{
    public:
    void display(){//overrides the display()method of class ABC
        cout<<"\nThis is child class";
    }
};

int main(){
    XYZ x;
    x.display();//method of class XYZ invokes, instead of class ABC
    x.ABC::display();
}
```



# Virtual Base Class



# Virtual base class (Cont...)

---

- **Virtual base class** is used to prevent the duplication/ambiguity.
- In hybrid inheritance child class has two direct parents which themselves have a common base class.
- So, the child class inherits the grandparent via two separate paths. it is also called as indirect parent class.
- All the public and protected member of grandparent are inherited twice into child.
- We can stop this duplication by making base class **virtual**.

```
class A
{
    public:
    int i;
};
class B: virtual public A
{
    public:
    int j;
};
class C: public virtual A
{
    public:
    int k;
};
```

```
class D: public B, public C
{
    public:
    int sum;
};
int main()
{
    D ob1;
    ob1.i=10;
    ob1.j=20;
    ob1.k=30;
    ob1.sum=ob1.i+ob1.j+ob1.k;
    cout<<ob1.sum;
}
```

# Derived class constructor

---

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

class Derived : public Base
{
    int y;
    public:
        Derived() { cout<<"Derived default constructor"; }
        Derived(int i) { cout<<"Derived parameterized constructor"; }
};

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

# Derived class constructor (Cont...)

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

class Derived : public Base {
  int y;
  public:
  Derived(int i,int j) : Base(j)
  { y = i; cout << "y="<<y;
  }
};

int main()
{
  Derived d(10,20) ;
}
```

# Execution of base class constructor

Method of inheritance	Order of execution
<pre>class Derived: public Base { };</pre>	<pre>Base(); Derived();</pre>
<pre>class C: public A, public B { };</pre>	<pre>A();//base(first) B();//base(Second) C();derived</pre>
<pre>class C:public A, virtual public B { };</pre>	<pre>B();//virtual base A();//base C();derived</pre>

Thank You