Object Oriented Programming in C++

Nepal College of Information Technology

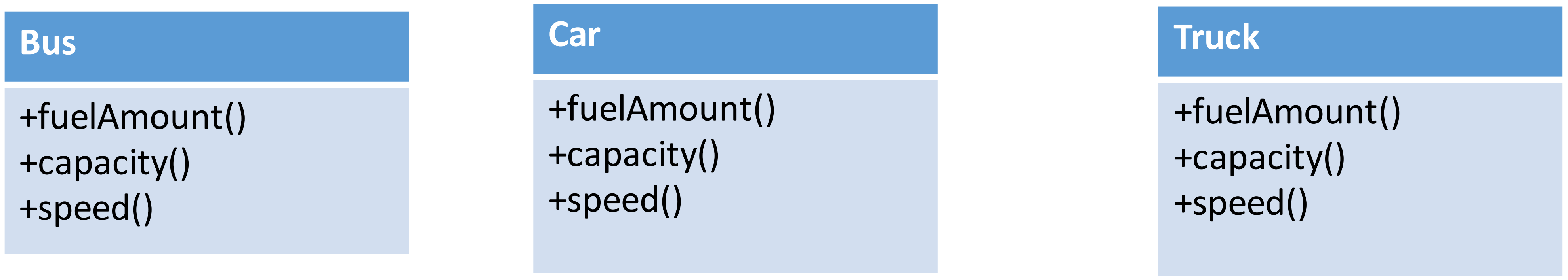
Course Instructor: Er. Rabina Chaudhary

Chapter 4: Object Inheritance and Reusability

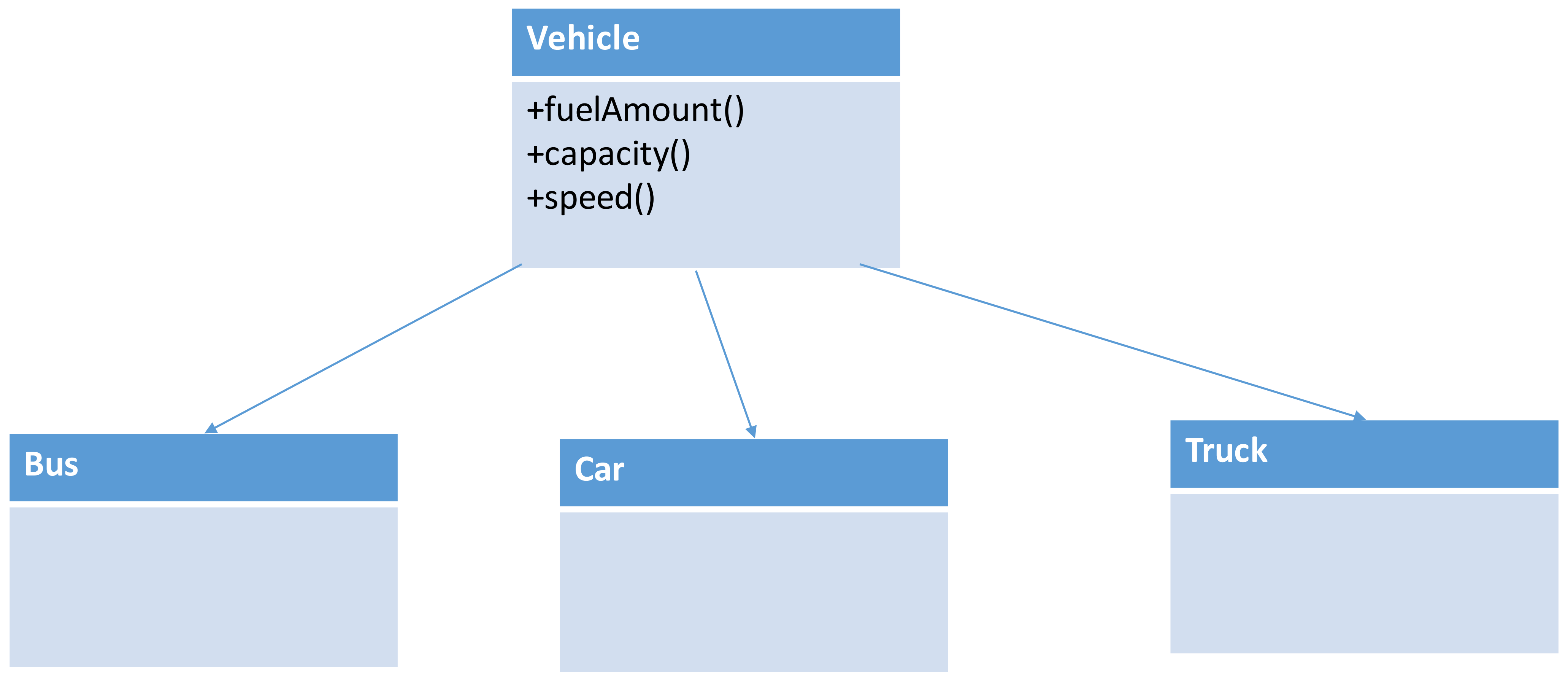
Inheritance:

- Inheritance allows us to create new class from already existing class
- Inheritance is a process where the child class acquires properties and functionality of its parent class
- The Class that inherits another class is referred as child class, derived class or subclass
- The already existing class through which new class is inherited is referred as parent class, base class or super class

Without using inheritance:



Using inheritance:



Inheritance:

Syntax for defining a derived class is :

```
class baseClassName
        {
        ...     //code for base class
        };
class derivedClassName : visibility_mode baseClassName
        {
        ...      //code specific for derived class
        };
```

Inheritance:

Example:

```
class ABC
{
        code for base class ABC
};
class XYZ : public ABC
{
        code specific for derived class XYZ
};
```

Visibility modes/ Inheritance mode:

Visibility modes specifies whether the features of the base class is derived in private, public or protected mode

Types:

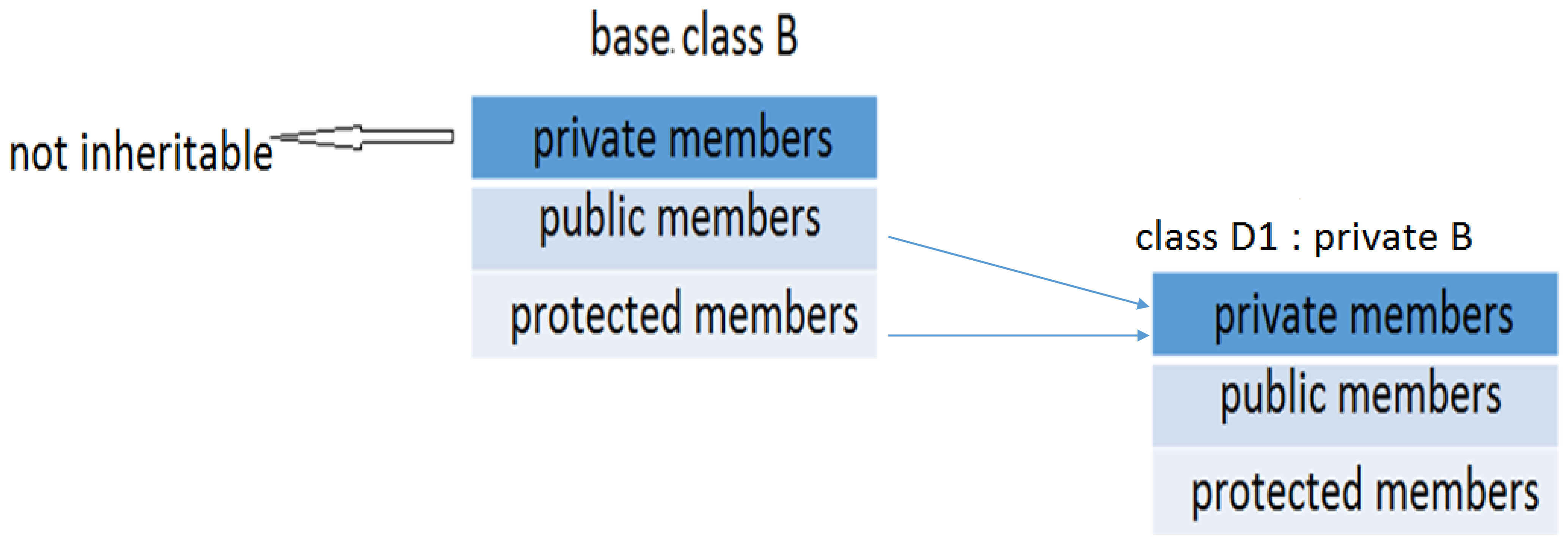
1. Private mode
2. Public mode
3. Protected mode

1. Private mode:

- When the base class is inherited by derived class in private mode, both the public member and protected members of base class will become private member in derived class.

Note: The private members of base class are not directly accessible in derived class however, they can be accessed using public or protected methods of base class in the derived class.

1. Private mode:

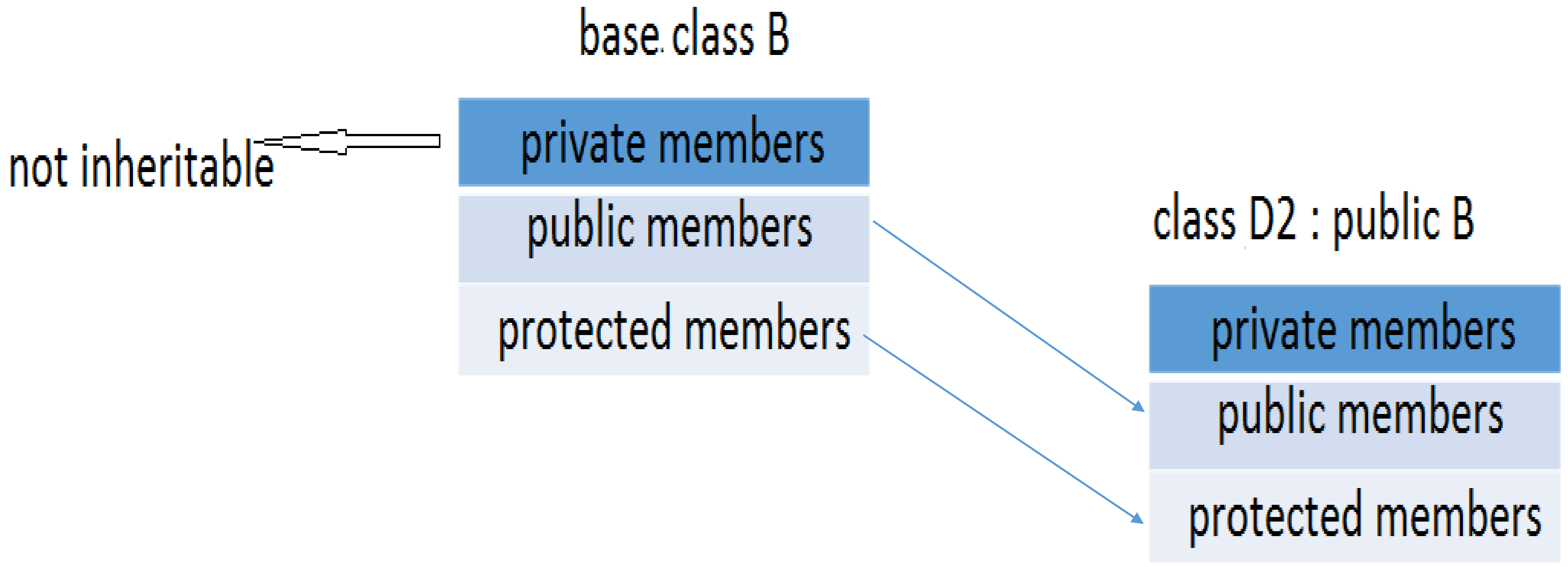


2. Public mode:

- When the base class is inherited by derived class in public mode, public members of base class will become public members and protected members will become protected members for derived class

Note: The private members of base class are not directly accessible in derived class however, they can be accessed using public or protected methods of base class in the derived class.

2. Public mode:



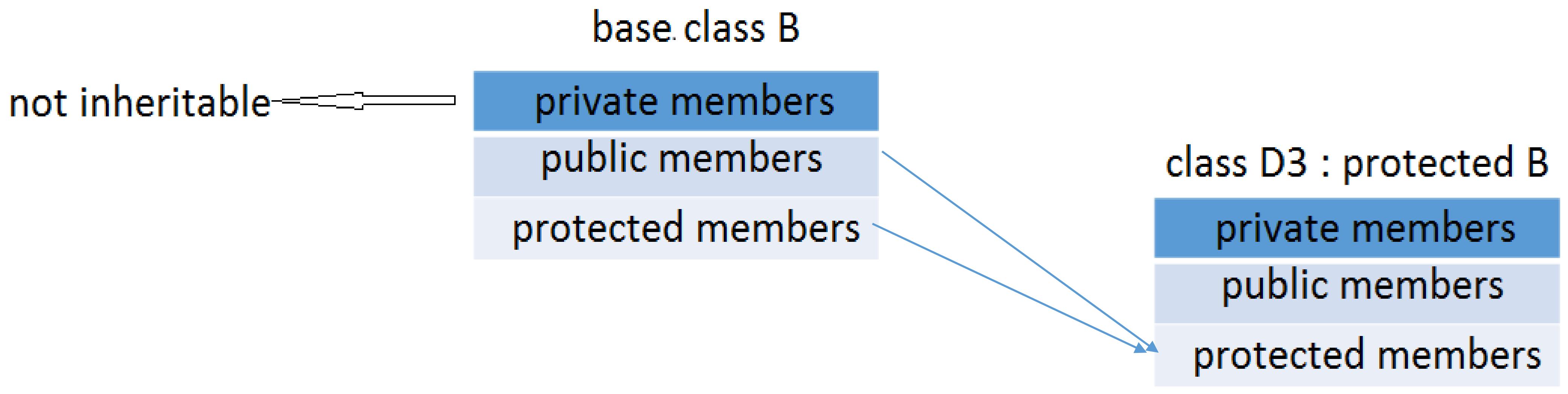
3. Protected mode:

- When the base class is inherited by derived class in protected mode, public members and protected members of base class will become protected members for derived class

Note: The private members of base class are not directly accessible in derived class however, they can be accessed using public or protected methods of base class in the derived class.

Note: Protected members can be accessed within same class and its derived classes only

3. Protected mode:



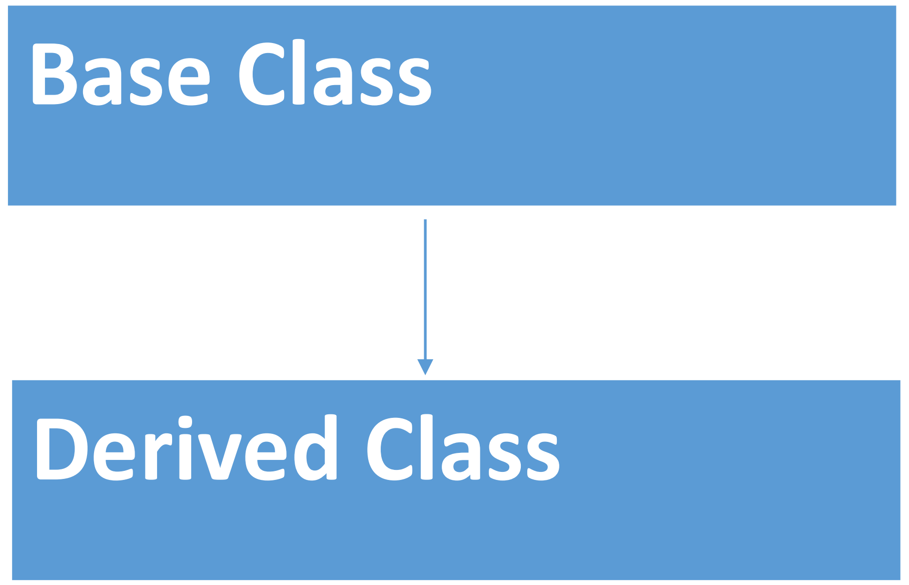
Types of Inheritance:

1. Single inheritance
2. Multilevel inheritance
3. Hierarchical inheritance
4. Multiple inheritance
5. Hybrid inheritance

1. Single Inheritance:

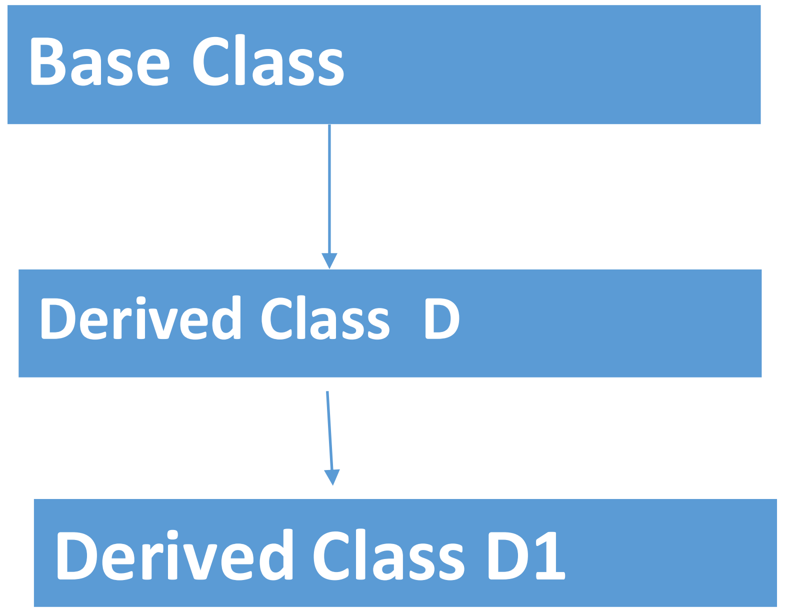
- When a single child class is being inherited by a single parent class, it is called single inheritance
- i.e. one derived class with only one base class

1. Single Inheritance:



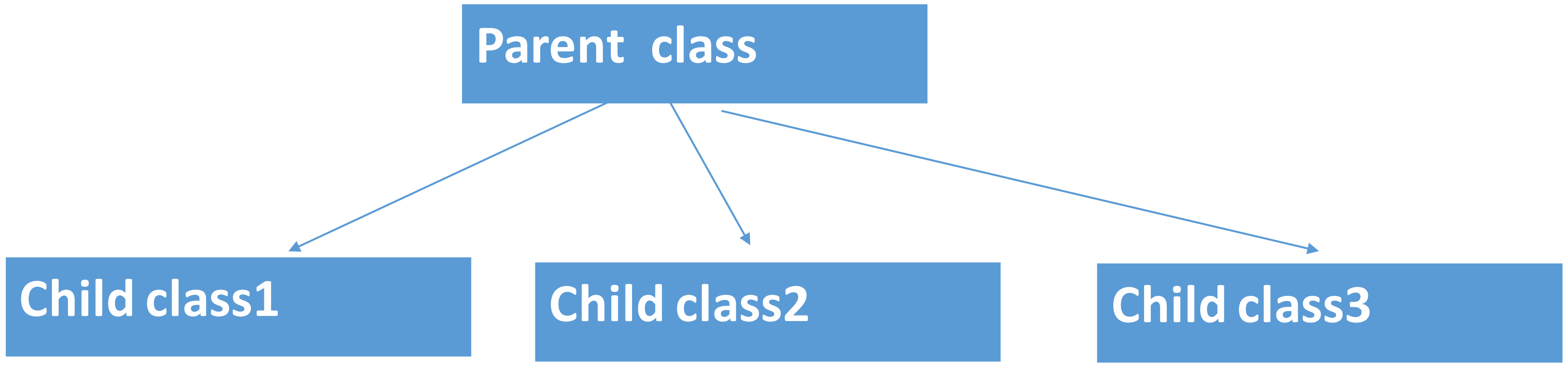
2. Multilevel Inheritance:

- In multilevel inheritance, a class is derived from another derived class.
- The base class of a derived class is derived class of another base class



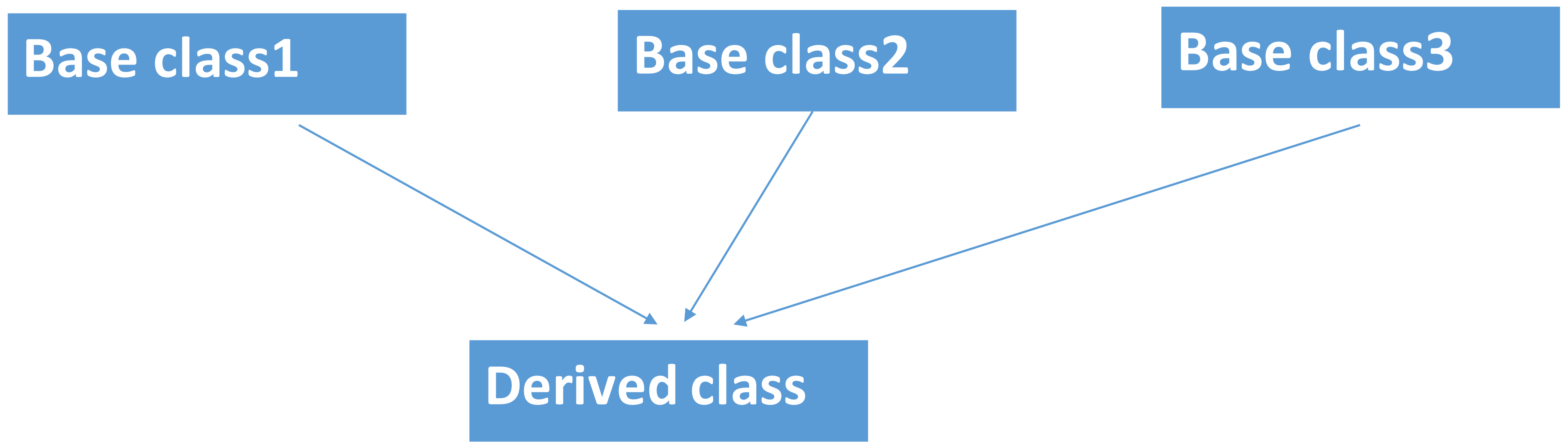
3. Hierarchical Inheritance:

- More than one child class is inherited from a base class in hierarchical inheritance



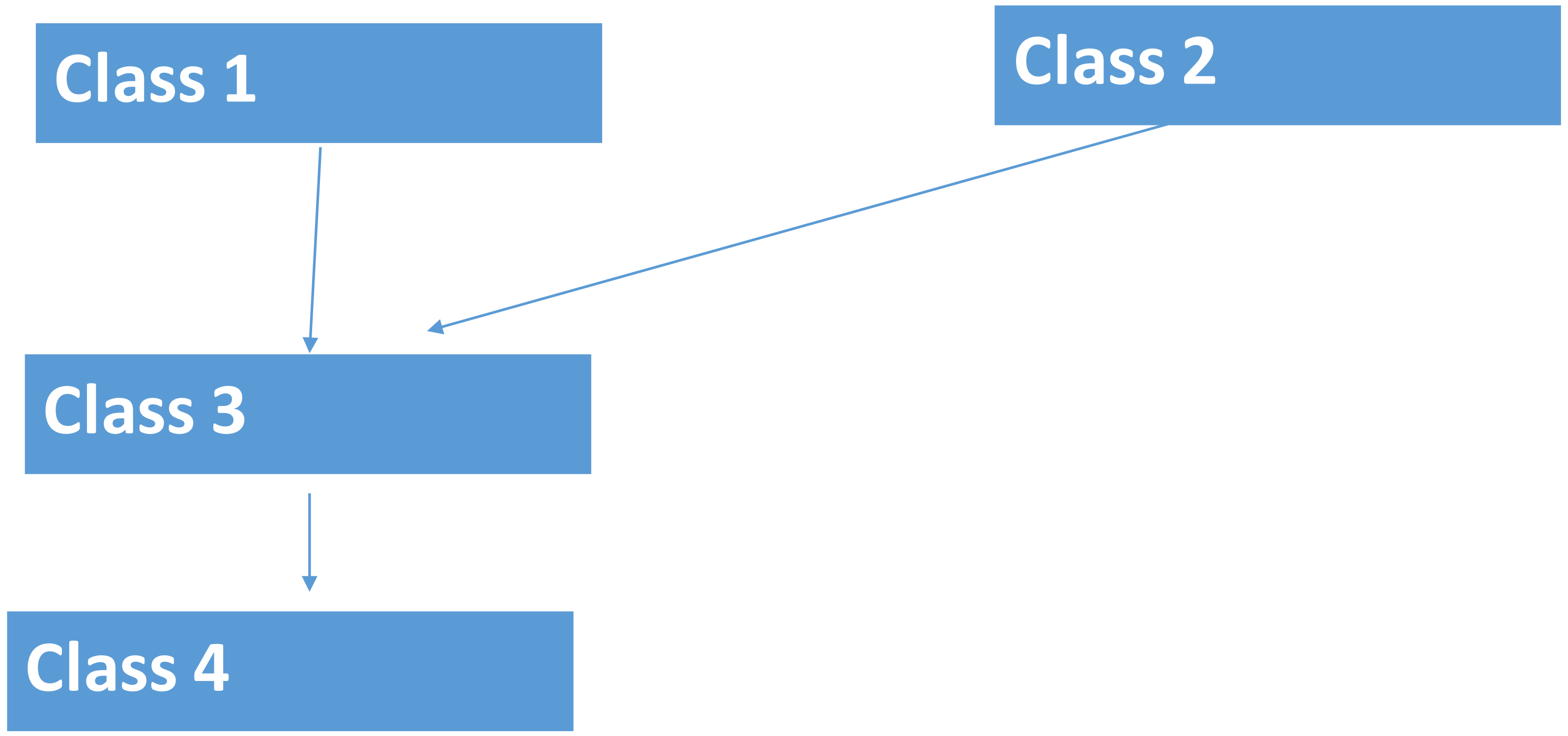
4. Multiple Inheritance:

- In this type of inheritance a child class is derived from more than one base class



5. Hybrid Inheritance:

- The inheritance which involves more than one form of inheritance is called hybrid inheritance



1. Single Inheritance:

- When a single child class is being inherited by a single parent class, it is called single inheritance
- i.e. one derived class with only one base class

Syntax to define a derived class using single inheritance:

```
class Derived_Class_Name : Inheritance_Mode Base_Class_Name
{
        class body
};
```

Example Program : using public inheritance mode

```
#include<iostream.h>
#include<conio.h>



class ABC
{
            private:
                        int a;
             protected:
                        int b;
            public:
                        int c;
                        void setValue();
                        void displayValue();
                        int getA()
                        {
                                    return a;
                        }
};
```

```
class XYZ: public ABC
{
        private:
                int x;
        public :
                void calculate();
                void displayResult();
};
```

```
void ABC :: setValue()
{
        cout << "Enter value of a, b and c:";
        cin >> a >> b >> c;
}
void ABC :: displayValue()
{
        cout << "Value of a : "<<a <<endl ;
        cout << "Value of b : "<<b<<endl;
        cout << "Value of c : "<<c <<endl;
}
```

```
void XYZ :: calculate()
{
        x = getA() + b + c;
}

void XYZ :: displayResult()
{
        cout<<endl<<"Result  after addition is   : "<< x <<endl;
}
```

```
void main()
{
            XYZ obj;
            obj.setValue();
            obj.displayValue();
            obj.calculate();
            obj.displayResult();

                        // obj.b =200;  //protected members are not accessible outside class.
            obj.c =100;    //public members are accessible outside class
            cout << endl <<"After updating the value of public member c :"<<endl;
            obj. displayValue();

getch();
}
```

Example Program : using private inheritance mode

```
#include<iostream.h>
#include<conio.h>



class ABC
{
                private:
                                int a;
                  protected:
                                int b;
                public:
                                int c;
                                void setValue();
                                void displayValue();
                                int getA()
                                {
                                                return a;
                                }
};
```

```
class XYZ: private ABC        // replace visibility mode with  protected
{
        private:
                int x;
        public :
                void calculate();
                void displayResult();
};
```

```
void ABC :: setValue()
{
        cout << "Enter value of a, b and c:";
        cin >> a >> b >> c;
}
void ABC :: displayValue()
{
        cout << "Value of a : "<<a <<endl ;
        cout << "Value of b : "<<b<<endl;
        cout << "Value of c : "<<c <<endl;
}
```

```
void XYZ :: calculate()
{
            setValue();             //accessing setValue() of Base class ABC
            x = getA() + b + c;
}

void XYZ :: displayResult()
{
            displayValue();      //accessing displayValue() of Base class ABC
            cout<<endl<<"Result after addition is  : "<< x <<endl;
}
```

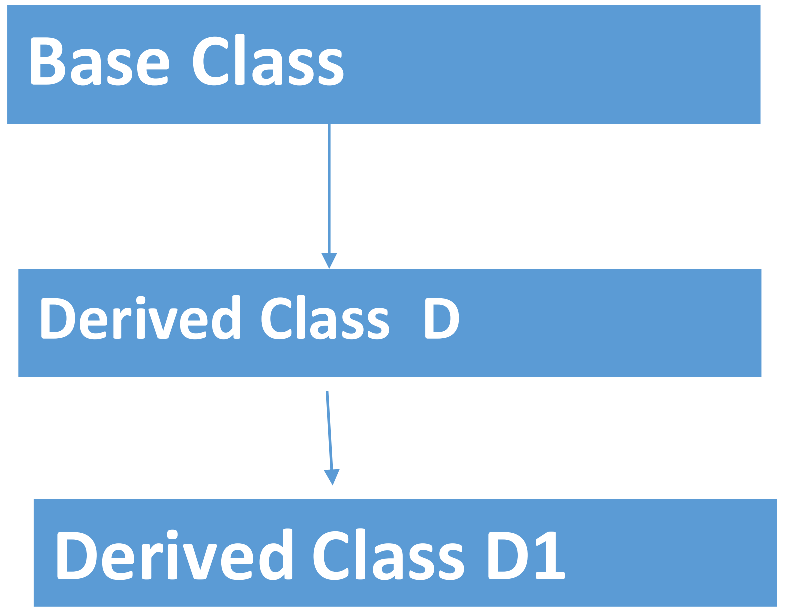
```
void main()
{
        XYZ obj;

        /* obj.setValue();
        obj.displayValue();             cannot be accessed outside class as XYZ has inherited ABC's properties
in private mode                 */


        obj.calculate();
        obj.displayResult();

                // obj.b =200;  //private members are not accessible outside class.
                 //obj.c =100;    //private members are accessible outside class

getch();
}
```

Practice:

Q1. Write a program to create a class named Person which has name and age as data members and member functions to read and display its data. Create another class Student derived from class Person to use the features of base class.

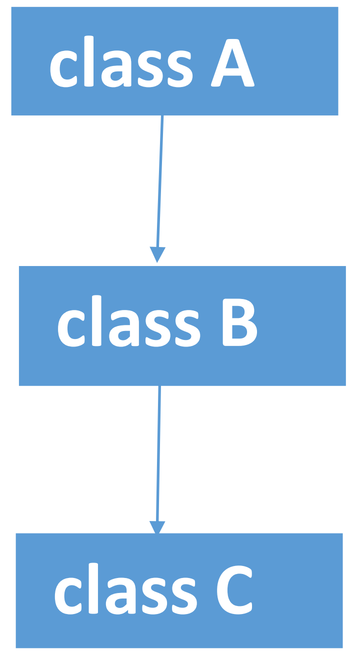
2. Multilevel Inheritance:

- In multilevel inheritance, a class is derived from another derived class.
- The base class of a derived class is derived class of another base class



2. Multilevel Inheritance:

```
class A
        {
                body of class A
        };
class B : visibility_mode A
        {
                body of class B
        };
class C : visibility_mode B
        {
                body of class C
        };
```



Example Program:

```
#include <iostream.h>
#include <conio.h>
class Base
{
        protected:
                int a;
        public:
                void setData ()
                {
                        cout << "Enter the value of a = ";
                        cin >> a;
                }
};
```

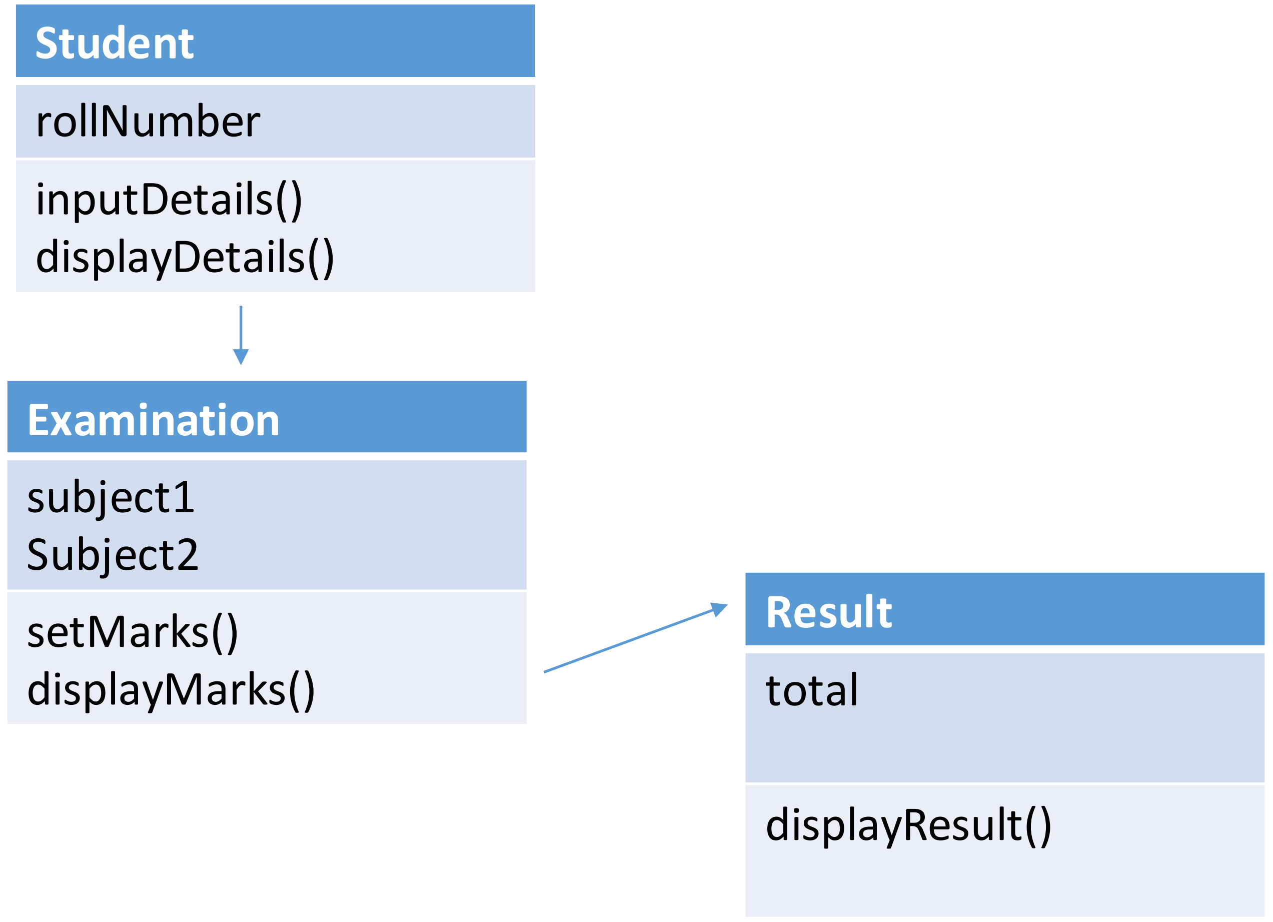
```
class DerivedOne : public Base
{
        protected:
                int b;
        public:
                void readData ()
                {
                        cout << "Enter the value of b = ";
                        cin >> b;
                }
};
```

```
class DerivedTwo : public DerivedOne
{
        private:
                int c;
        public:
                void input()
                {
                        cout << "Enter the value of c = ";
                         cin >> c;
                }
                void product()
                {
                        cout << "Product = " << a * b * c;      //accessing members a and b of base class
                }
};
```

```
int main ()
{
        DerivedTwo obj;
        obj.setData();        //accessing member of Base class
        obj.readData();    //accessing member of DerivedOne
        obj.input();
        obj.product();
        getch();
        return 0;
}
```

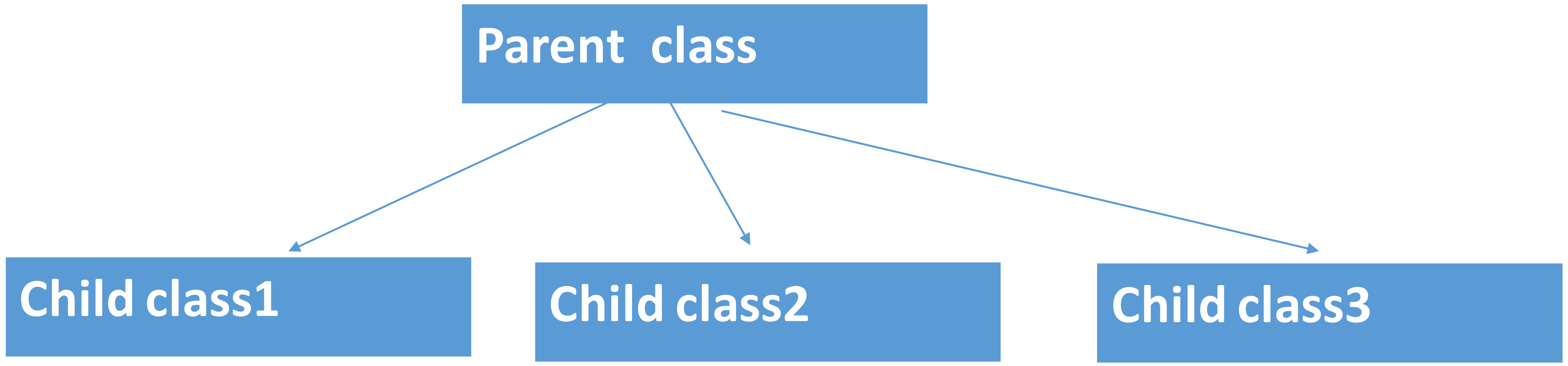
Practice:

Q1. Define a class Student which has roll number and member functions to input and display roll number. Derive a class Examination from class Student which has marks of two subjects and member functions to initialize and display marks. Again derive a class Result from class Examination and calculate total and display the result.

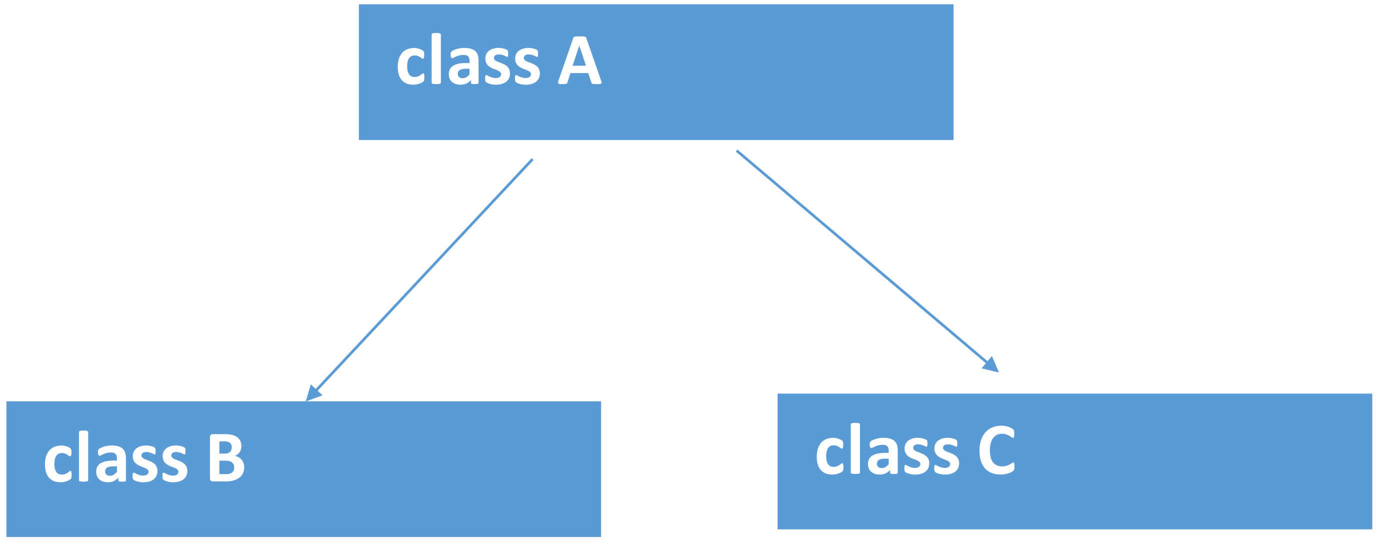


3. Hierarchical Inheritance:

- More than one child class is inherited from a base class in hierarchical inheritance



```
class A
        {
                ...
        };
class B : visibility_mode A
        {
                ...
        };
class C : visibility_mode A
        {
                ...
        }
```



Example Program:

```
#include <iostream.h>
#include <conio.h>
class Base          //single base class
{
  protected:
                         int a,b;
              public:
                         void setData ()
                         {
                         cout << "Enter the value of a : ";
                         cin >> a;
                         cout << "Enter the value of b : ";
                         cin >>b;
                         }
};
```

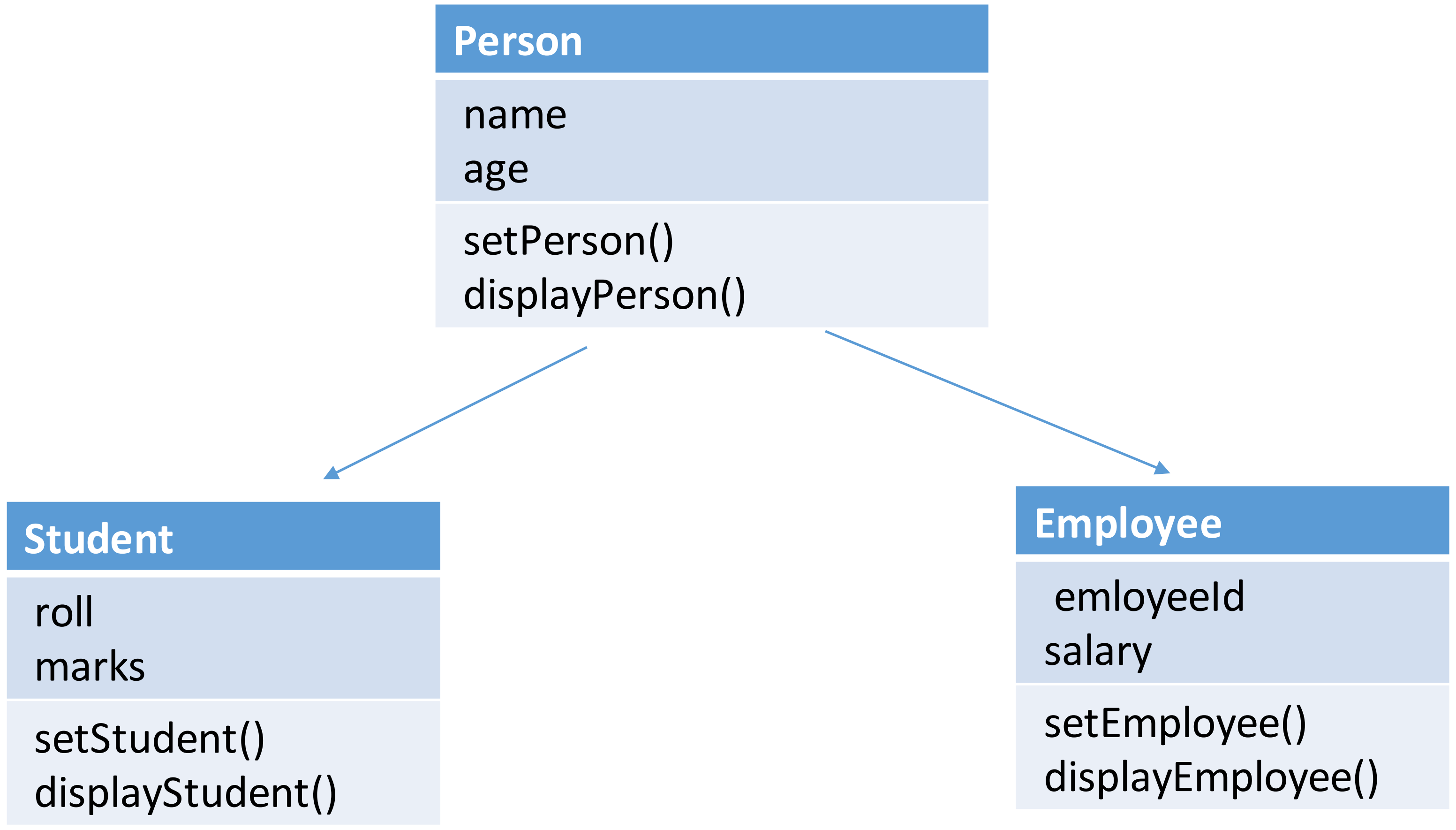
```
class DerivedOne : public Base          //DerivedOne is derived from class Base
{
            protected:
                        int c;
            public:

                        void add()
                        {
                                    c=a+b;                              //access the member of base class
                                    cout << a << "+" << b <<"="<<c<<endl;
                        }
};
```

```
class DerivedTwo : public Base              //DerivedTwo is also derived from class Base
{
        private:
                int c;
        public:

                void product()
                {
                        c=a*b;
                        cout << a < < "*" << b <<"="<<c<<endl;
                }
};
```

```
int main ()
{
        DerivedOne obj1;   //Object of DerivedOne class
        DerivedTwo obj2;    //Object of DerivedTwo class

        obj1.setData();        // call member function of Base
        obj1.add();

        obj2.setData();        // call member function of Base
        obj2.product();

        getch();
        return 0;
}
```

Practice:

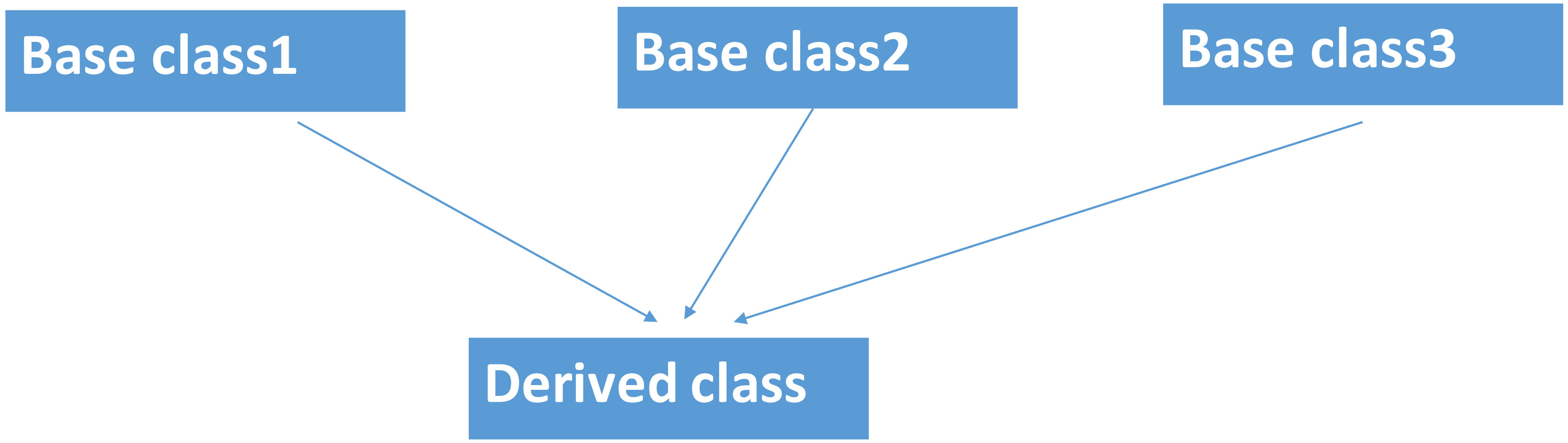
Q1. Write a program to use hierarchical inheritance.



4. Multiple Inheritance:

- In this type of inheritance a child class is derived from more than one base class

i.e. a child class inherits properties of two or more base classes.

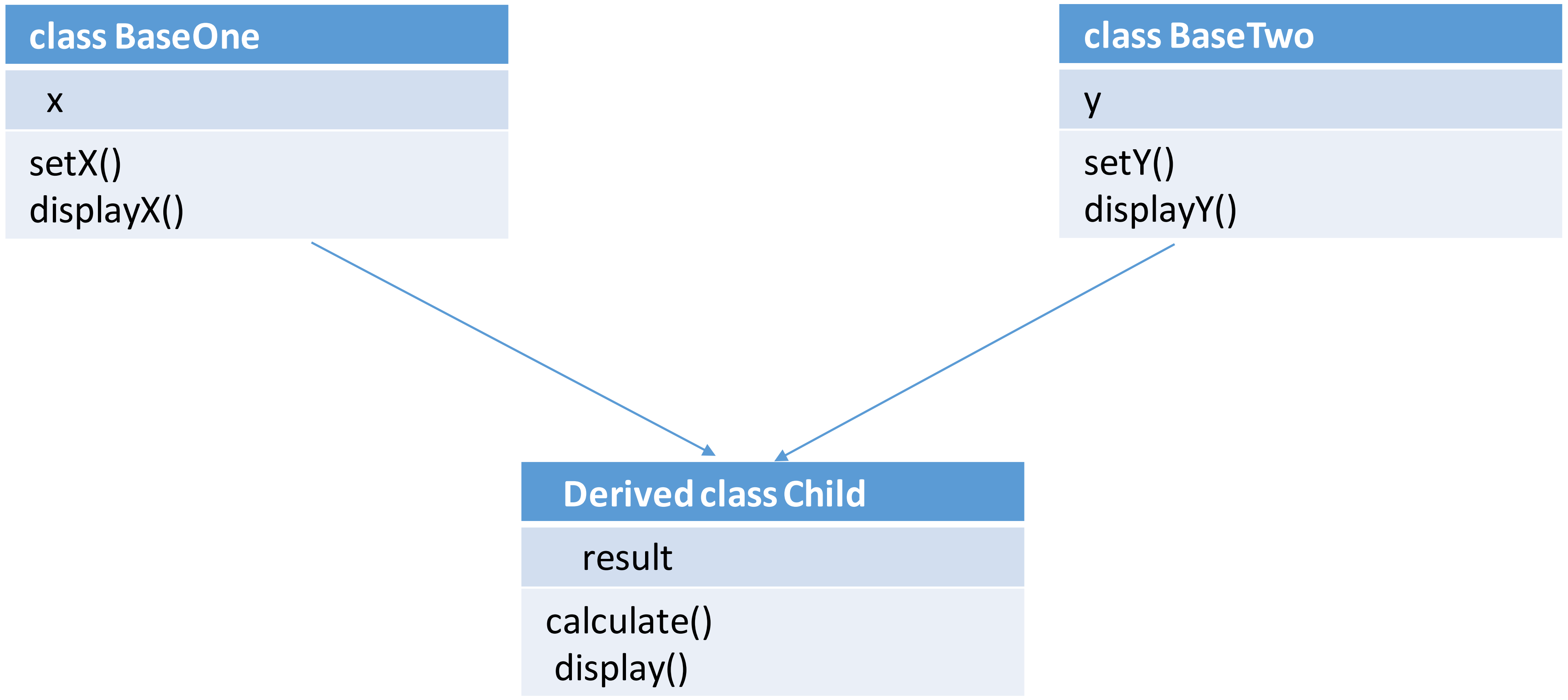


4. Multiple Inheritance:

Syntax to define a derived class using multiple inheritance:

```
class Base_Class_One
        {
        …
        };
class Base_Class_Two
        {
                …
        };
class Derived_Class : visibility_mode Base_Class_One , visibility_mode Base_Class_Two
        {
          …
        };
```

Example :


class BaseOne
x
setX()
displayX()
class BaseTwo
y
setY()
displayY()
Derived class Child
result
calculate()
display()

```
#include <iostream.h>
#include <conio.h>

class BaseOne
{
        protected  :
                int x;
        public  :
                void setX();
                void displayX();
};

class BaseTwo
{
        protected  :
                int y;
        public  :
                void setY();
                void displayY();
};
```

```
void BaseOne:: setX()
{
                cout << "Enter value of x :";
                 cin >> x ;
}

void BaseOne :: displayX()
{
                cout << endl <<"The value of x is : " <<x <<endl;
}
void BaseTwo:: setY()
{
                cout << "Enter value of y :";
                cin >> y ;
}

void BaseTwo :: displayY()
{
                cout << endl <<"The value of y is : " <<y <<endl;
}
```

```
class Child : public BaseTwo, public BaseOne
{
        private:
        int result;
  public:
        void calculate();
        void display();
};
```

```
void Child :: calculate()
{
        result = x + y;        //accessing x and y inherited from its base classes
}
void Child :: display()
{
        cout << endl <<"The result after addition is :"<<result <<endl;
}

void main()
{
        Child obj;
        obj.setX();
        obj.setY();
        obj.displayX();
        obj.displayY();
        obj.calculate();
        obj.display();

  getch();
}
```

Ambiguity in Multiple Inheritance:

- Ambiguity arises in multiple inheritance when more than one base class have same function name which is not overridden in derived class.
- If we try to call the function using the object of the derived class, compiler shows error because compiler doesn't know which function to call.

Ambiguity in Multiple Inheritance: Example program

```
#include <iostream.h>
#include <conio.h>

class BaseOne
{
            protected :
                        int x;
            public :
                        void set();
                        void display();
};

class BaseTwo
{
            protected :
                        int y;
            public :
                        void set();
                        void display();
};
```

```
void BaseOne:: set()
{
        cout << "Enter value of x :";
        cin >> x ;
}

void BaseOne :: display()
{
        cout << endl <<"The value of x is : " <<x <<endl;
}
void BaseTwo:: set()
{
        cout << "Enter value of y :";
        cin >> y ;
}
```

```
void BaseTwo :: display()
{
        cout << endl <<"The value of y is : " <<y <<endl;
}

void Child :: calculate()
{
        result = x + y;
}
void Child :: displayResult()
{
        cout << endl <<"The result after addition is :"<<result <<endl;
}
```

```
class Child : public BaseTwo, public BaseOne
{
        private:
        int result;
  public:
        void calculate();
        void displayResult();
};
```

```
void main()
{
        Child obj;
        obj.set();
                    /*compiler throws an error as there is ambiguity
                         about which base class's set() to call  */
        obj.set();
        obj.display();
                    / *compiler throws an error as there is ambiguity
                    about which base class's display() to call   */
        obj.display();
        obj.calculate();
        obj.displayResult();

  getch();
}
```

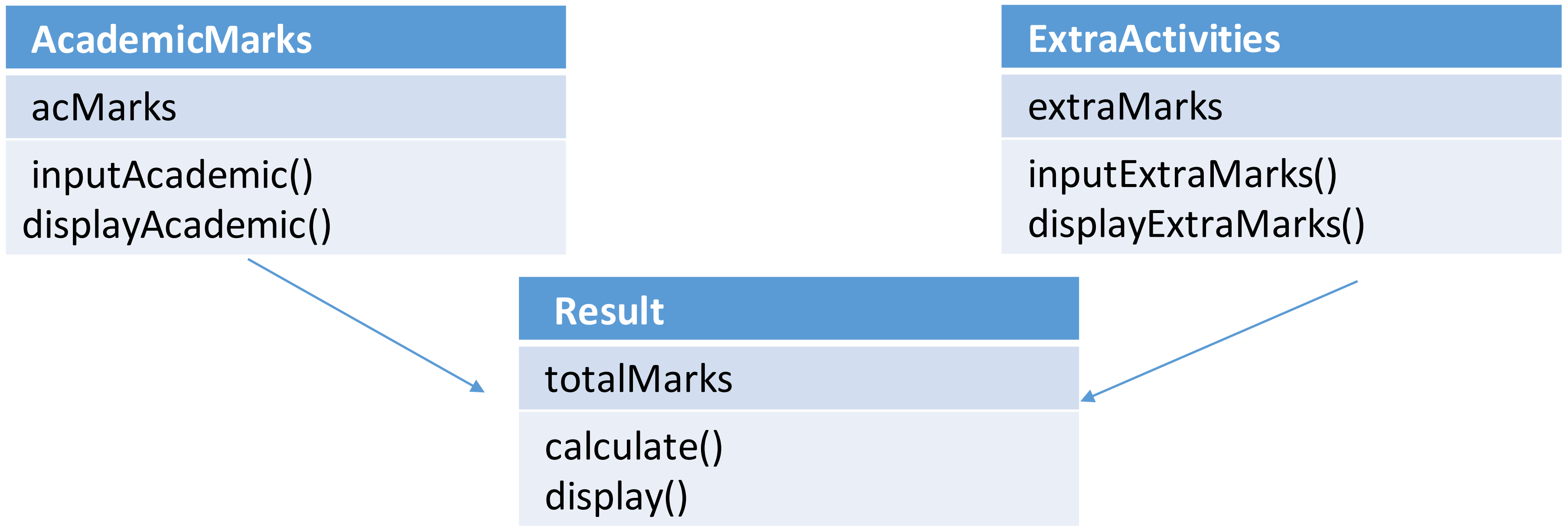
Solution of ambiguity in multiple inheritance:

The ambiguity can be resolved by using the class_name and scope resolution operator (::) to specify the class name of member function.

```
void main()
{
        Child obj;
        obj.BaseOne::set();                 //calls BaseOne's set()
        obj.BaseTwo::set();                 //calls BaseTwo's set()
        obj.BaseOne::display();              //calls BaseOne's display()
        obj.BaseTwo::display();              //calls BaseOne's display()
        obj.calculate();
        obj.displayResult();

  getch();
}
```

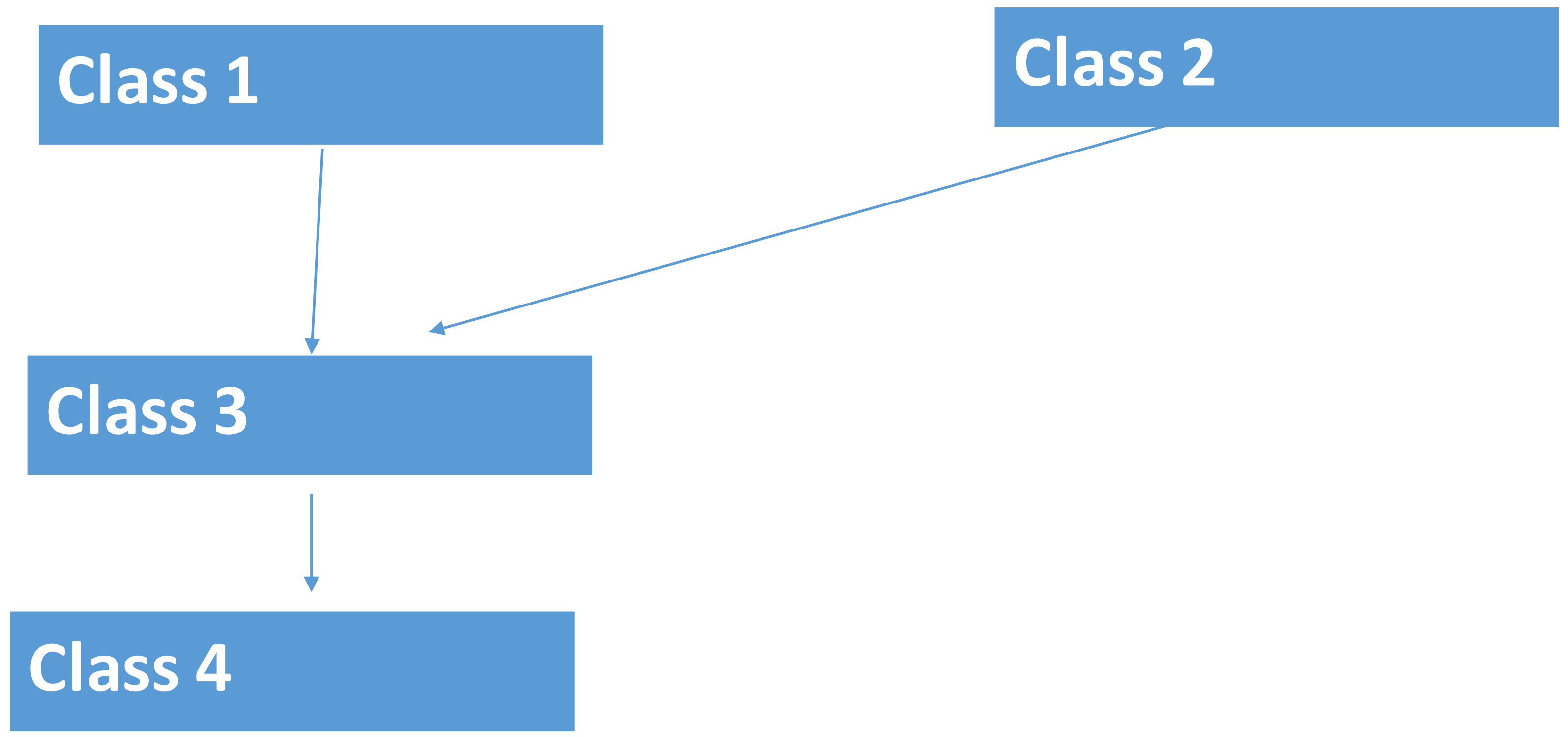
Practice:

Q1. Write a program to define class AcademicMarks and ExtraActivities which have data members for academic marks and extra activities marks respectively. Define member functions to initialize the marks. Derive a class Result from AcademicMarks and ExtraActivities and calculate the total marks obtained.



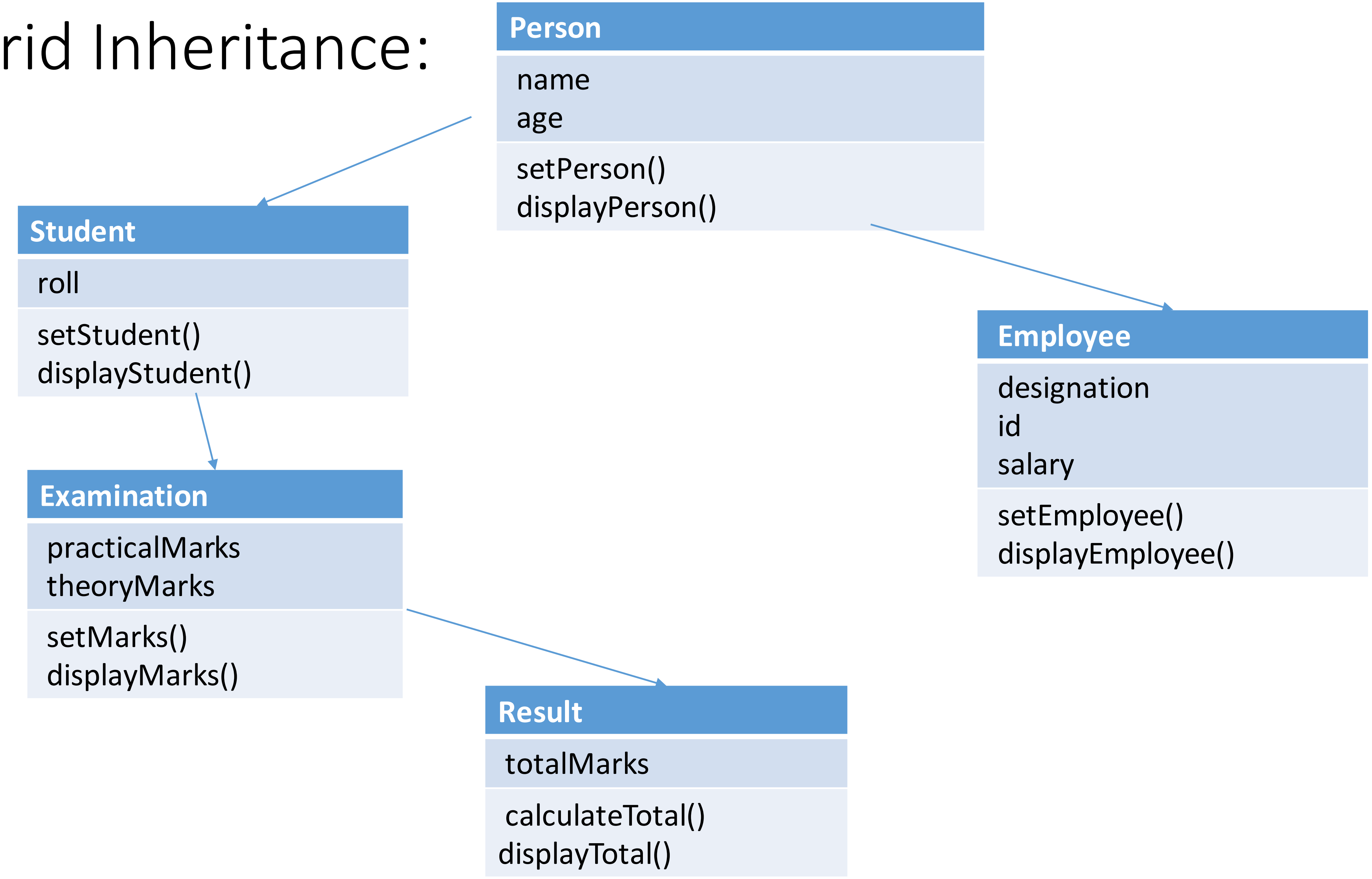
5. Hybrid Inheritance:

- The inheritance which involves more than one form of inheritance is called hybrid inheritance



Hybrid Inheritance:

Q1.



```
#include<iostream.h>
#include<conio.h>
class Person
{
   protected:
                                                  char name[20];
                                                    int age;
   public:
                         void setPerson()
                         {
                                                  cout << "Enter name : ";
                                                  cin >> name ;
                                                  cout << "Enter age : ";
                                                  cin >> age;
                         }
                         void displayPerson()
                         {
                                                  cout<< endl << "Name : "<< name <<endl << "Age : "<< age <<endl;
                         }
};
```

```
class Employee : public Person
{
          private :
                    char designation[20];
                     int id;
                    int salary;
          public  :
                    void setEmployee()
                    {
                              cout << "Enter designation : ";
                              cin >> designation;
                              cout << "Enter id : ";
                              cin >> id;
                               cout << "Enter salary : " ;
                              cin >> salary;
                     }
                     void displayEmployee()
                    {
                              cout << "Designation :"<<designation << endl << "Id : "<<id <<endl <<"Salary :" << salary <<endl;
                     }
};
```

```
class Student : public Person
{
      protected:
            int roll;
      public:
            void setStudent()
            {
                  cout<< endl <<"Enter roll number :";
                   cin >> roll;
             }
            void displayStudent()
            {
                  cout <<endl <<"Roll number :"<<roll<<endl;
            }
};
```

```
class Examination  : public Student
{
            protected:
                        float practicalMarks, theoryMarks;
            public:
                        void setMarks()
                        {
                                    cout << "Enter practical marks :";
                                    cin >> practicalMarks;
                                    cout << "Enter theory marks :";
                                    cin >> theoryMarks;
                        }
                        void displayMarks()
                        {
                                    cout << "Practical Marks :"<<practicalMarks << endl << "Theory Marks :" <<theoryMarks <<endl;
                        }
};
```

```
class Result : public Examination
{
            float totalMarks;
   public:
            void calculateTotal()
            {
                        totalMarks = practicalMarks + theoryMarks;
            }
            void displayTotal()
            {
                        cout << "Total Marks :"<<totalMarks<<endl;
            }
};
```

```
void main()
{
            Employee e;
             cout <<"Enter Information of Employee "<<endl;
            e.setPerson();
            e.setEmployee();
            cout << endl << endl << "Information of Employee : "<<endl;
            e.displayPerson();
            e.displayEmployee();

            Result s;
            cout << endl << endl << "Enter Information of Student "<<endl;
            s.setPerson();
            s.setStudent();
            s.setMarks();
            cout << endl << endl << "Information of Student "<<endl;
            s.displayPerson();
            s.displayStudent();
            s.displayMarks();
            s.calculateTotal();
            s.displayTotal();

  getch();
}
```

Practical:

Q1. Create a class called **Student** with two data member to represent name and age of the student. Use member function to read and print these data. From this class, derive a class called **Boarder** with data member to represent room number. Derive another class called **DayScholar** from the class Student with data member to represent address and bus number of the student. In both derived classes, use member function to read and print the respective data.

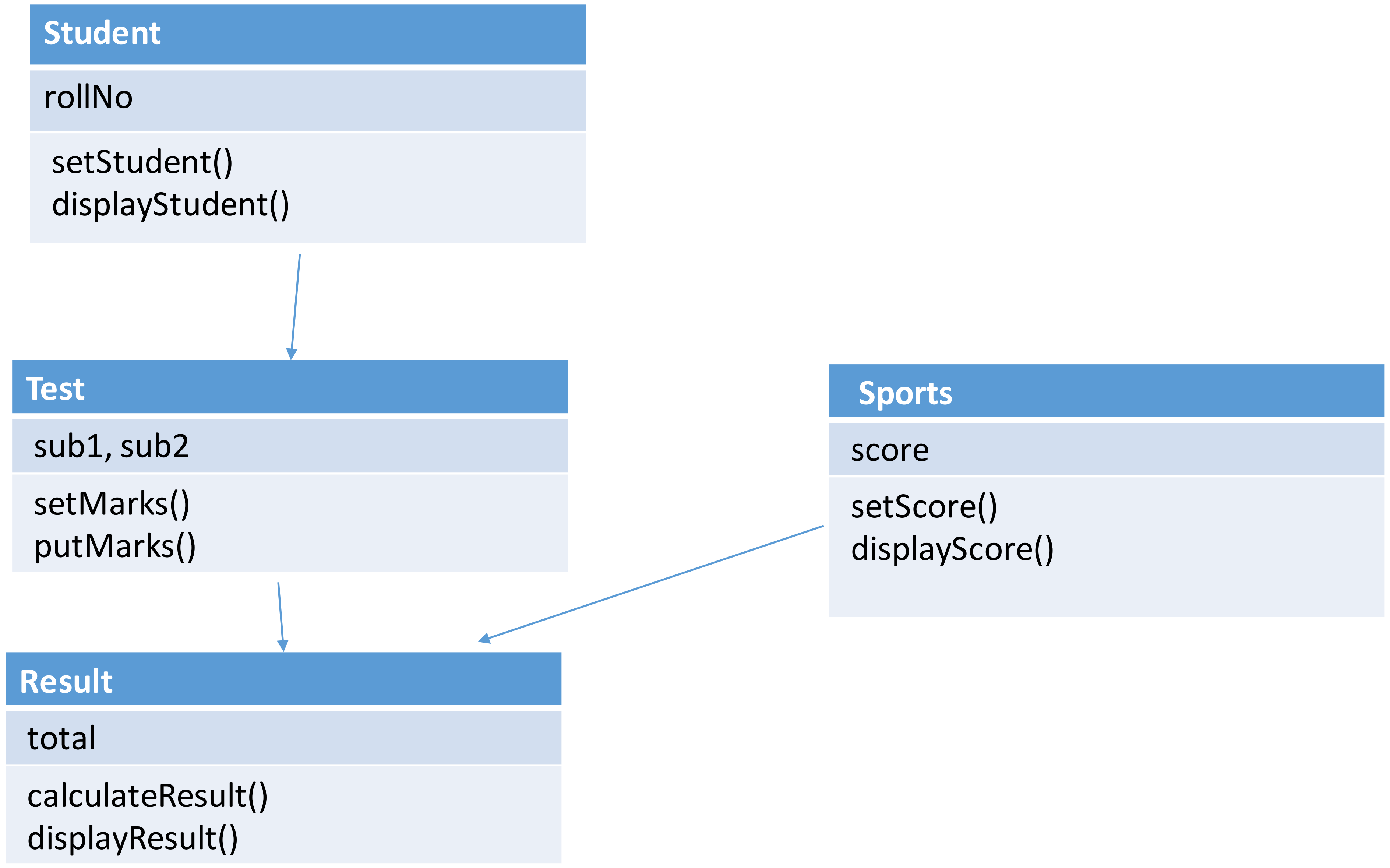
Practical:

Q2. Create a class called Rectangle with data member to represent length, breadth and area. Use appropriate member functions to read length and breadth from user to calculate area of the rectangle. Next create a derived class called Box from the class Rectangle. Use appropriate member function to read height and to calculate the volume of the box.

Practical:

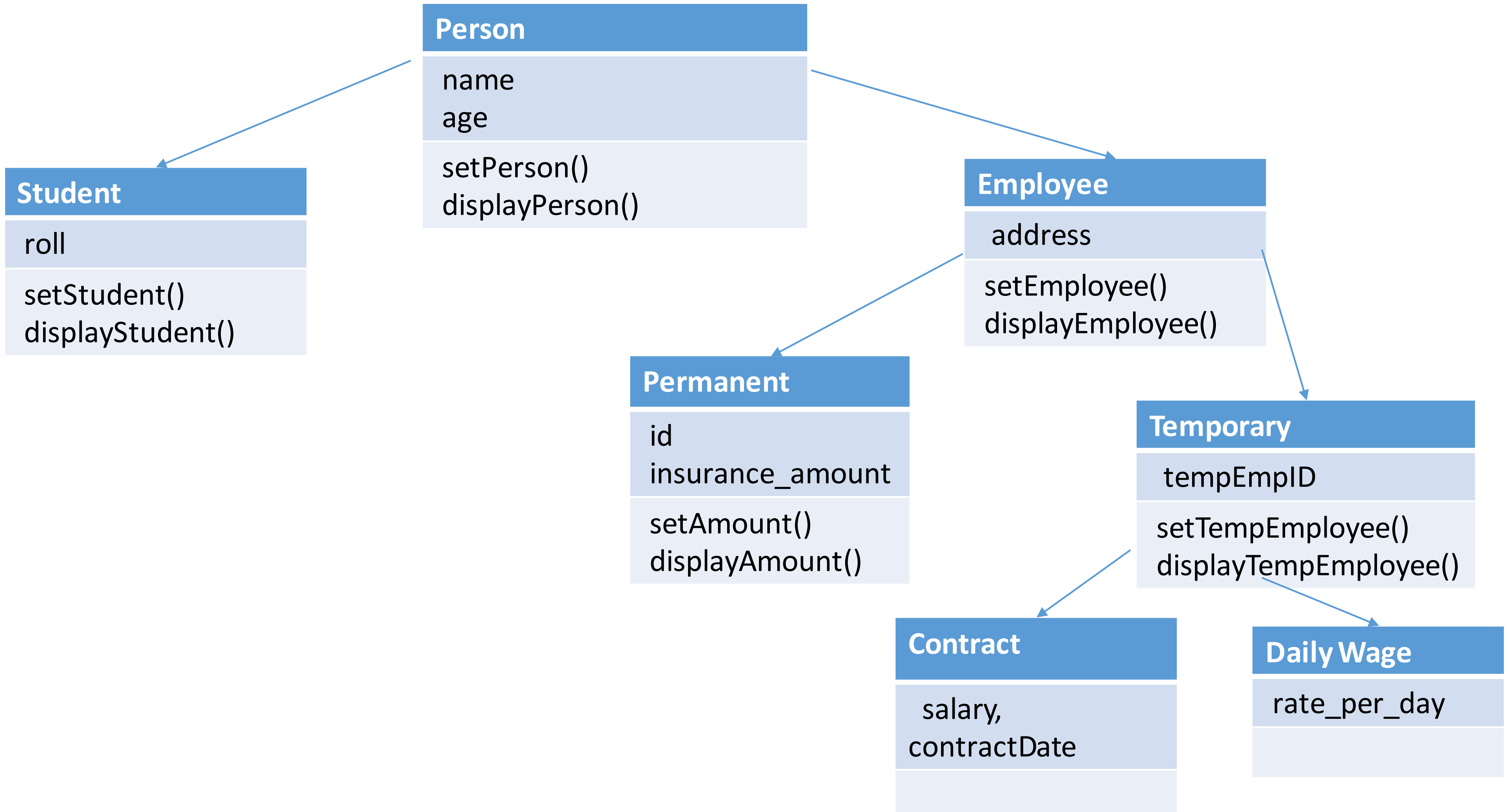
Q3. Write a base class that ask the user to enter a complex number and derived adds the complex number of its own with the base. Finally make third class that is friend of derived and calculate the difference of base complex number and its own complex number.

Q4. Write a program to represent following classes with following data members.



Q5. Create a class **Polygon** with data members to represent two dimensions and appropriate member functions to read its data member. Derive two classes **Rectangle** and **Triangle** from above class. Define member function area() in each derived class to calculate area of corresponding figure.

Q6. Write a program to represent following classes.


Person
name
age
setPerson()
displayPerson()
Student
roll
setStudent()
displayStudent()
Employee
address
setEmployee()
displayEmployee()
Permanent
id
insurance_amount
setAmount()
displayAmount()
Temporary
tempEmpID
setTempEmployee()
displayTempEmployee()
Contract
salary,
contractDate
Daily Wage
rate_per_day