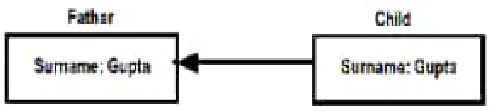
Inheritance:



- Inherit ance is one of the Very powerful features of the object oriented programming.
- We can Create a ***new brand thing*** by adapting (inheriting) the features of ***existing thing.***
- ***We can improve the blueprint/design for object.***
- ***To understand it let take an example.***

```
/*Write a class named Index that consist following properties and methods
properties: idx that is an intger varible stores index numbers i.e. start from 0 up n
 constructor: default constructor that set idx variable to 0
methods:
        next() that increase the value of idx variable by 1
        display() that show current value of idx variable
*/
#include<iostream>
using namespace std;
class Index
{
        protected:
                int idx;
        public:
        Index()
        {
                this->idx = 0;
        }
        void next()
        {
                this->idx++;
        }
        void display()
        {
                cout << endl << "current index is : " << this->idx;
        }
};
main()
{
        Index myindex ;
        myindex.next();
        myindex.display();
        myindex.next();
        myindex.display();
        myindex.next();
        myindex.display();

}
```

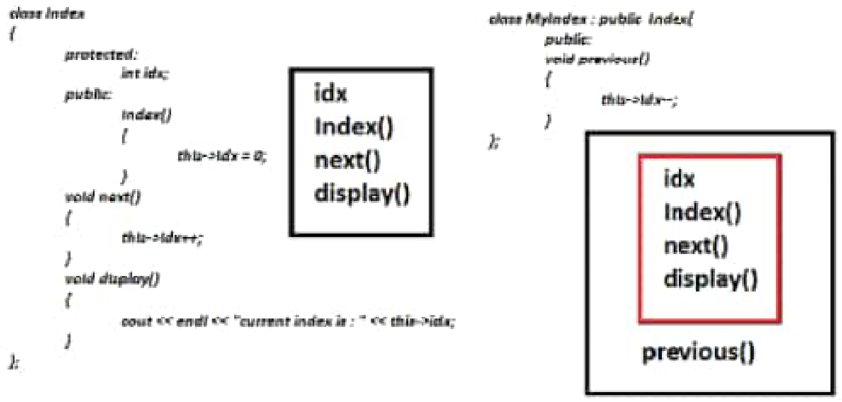
- The class Index offer to increase the index, but there is not feature to decrease index i.e. is there is no method of decreasing the index.
- One way is to add new method named previous ( ) method into Index class which is already compile or debugged, then do the testing process again. In-spite of that there always exists a possibility that at the end of the entire process the original class itself may not work satisfactorily.

  e.g.

```
#include<iostream>
using namespace std;
class Index
{
        protected:
                int idx;
        public:
                Index()
                {
                        this->idx = 0;
                }
                void next()
                {
                        this->idx++;
                }
                void display()
                {
                        cout << endl << "current index is : " << this->idx;
                }
};
class MyIndex : public  Index
{
        public:
        void previous()
        {
                this->idx--;
        }
};
main()
{
        MyIndex myindex;
        myindex.next();
        myindex.display();
        myindex.previous();
        myindex.display();

}
```

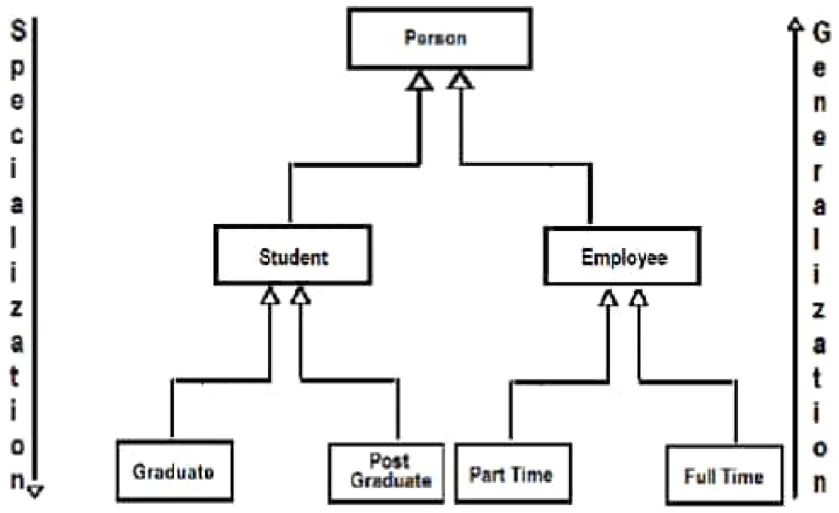
Fig.



```
class Index
{
        protected:
                int idx;
        public:
                Index()
                {
                        this->idx = 0;
                }
        void next()
        {
                this->idx++;
        }
        void display()
        {
                cout << endl << "current index is : " << this->idx;
        }
};
```

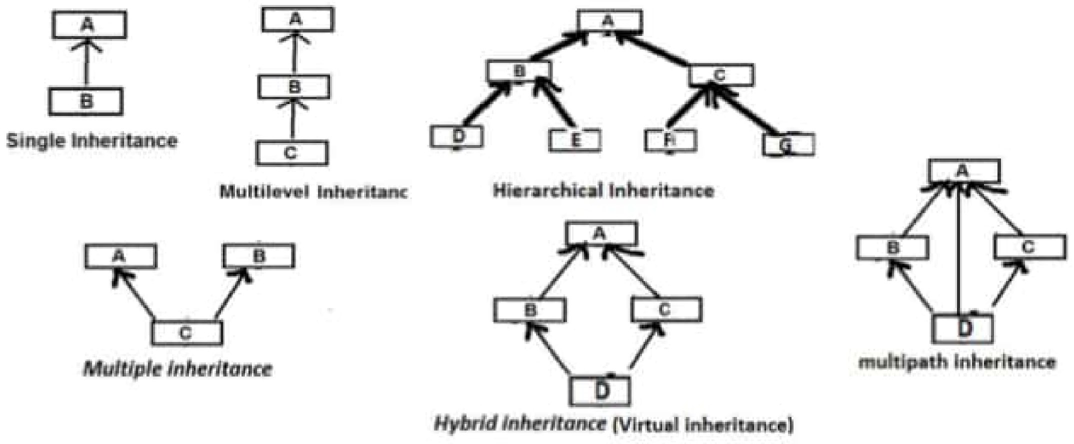
```
class MyIndex : public Index{
        public:
        void previous()
        {
                this->idx--;
        }
};
```

- It is called ***"is a"*** relationship and also called ***"a kind of "*** relationship.
- ***Benefits of Inheritance:***
    - **Code Reusability**
        - Once the super class is written and debugged.
        - It needs not to be touched again but we can use this super class to handle different – different situations by inheriting super class.
        - ***Reusing existing code not only saves time and "money" and increases the program "reliability" and "productivity".***
    - ***Error detection and correction become easy.***
    - ***Up gradation:*** Inheritance provides extendibility too, So developer can easily upgrade the existing System.
    - ***Specialization and Generalization*** concepts come through the inheritance.



- The concept of *generalization* in OOP means that an object encapsulates common state an behavior for a category of objects.
- The concept of *specialization* in OOP means that an object can inherit the common state and behavior of a generic object; however, each object needs to define its own special and particular state an behavior.
- ***Inheritance provide great help in case if distributing libraries***

- ***Type of inheritance:***



Derivation of member

Derivation of members

base class members / derivation type	public	protected	private
public (as it is)	public	protected	N/A
protected	protected	protected	N/A
private	private	private	N/A

Accessibility

	design time		*Execution*
access / *member*	*Same class*	*derive class*	*using object*
private	*yes*	*N/A private members of base class*	*NO*
protected	*yes*	*yes*	*NO*
public	*yes*	*yes*	*yes*

Example of Multiple Inheritance.

e.g.

```
#include<iostream>
using namespace std;
//developer1
class A
{
        public :
                void f1()
                {
                        cout <<"\nI am f1 of class A" ;
                }
};
//Developer2
```

```
class B
{
        public :
                void f2()
                {
                        cout <<"\nI am f2 of class B" ;
                }
};
//Developer3
class C
{
        public :
                void f3()
                {
                        cout <<"\nI am f3 of class C" ;
                }
} ;
//Developer4
class D: public A, public B, public C
{
        public :
                void f4()
                {
                        cout <<"\nI am in f4 of class D";
                }
};
//Programmer
main ()
{
                D d;
                d.f1();
                d.f2();
                d.f3();
                d.f4();
}
```

Function overriding

- ***A Developer who inherits the class that can create not only additional members in derived class as well as override member function.***
- If a parent class and a child class both are having same Signature of function then this concept is called **function overriding**.

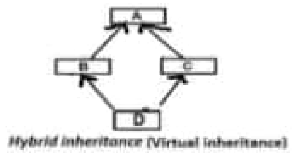
```
    e.g.
    #include<iostream>
    using namespace std;
    class Base
    {
            public :
                    void MyFunction()
                    {
                            cout <<"\nI am in Base class";

                    }
    };

    class derive : public Base
    {
            public :
            void MyFunction()
            {
                    cout << "\nI am in derive class";

            }
};
main()
{
        derive d;
        d.MyFunction();
}
```

Diamond Problem



Hybrid inheritance (Virtual inheritance)

```
e.g.
#include<iostream.h>
class A
{
        public:
                void display()
                {
                        cout << "\nI am display of A";
                }
};
```

```
class B : public A
{
        public:
                void f1()
                {
                        cout << "\nI am f1 of B";
                }
};

class C : public A
{
        public:
                void f2()
                {
                        cout << "\nI am f2 of C";
                }
};

class D : public B, public C
{
        public:

                void f3()
                {
                        cout << "\nI am display of D";
                }
};

main ()
{
        D d ;
        d.f1();
        d.f3();
        d.f2();
        //d.display();
}
```

- ***Such type of problem normally comes with Multiple Inheritance***
- The Members of class A comes twice inside the Class D.
- ***Virtual base class*** is used to avoid such ambiguity of the diamond problem

Virtual base Classes

- When we use virtual base class, then the technology *create virtual class on the basis of existing class* and ***insert the heavy code inside it*** for removing the duplicity but performance get degrade.

```
#include<iostream>
using namespace std;
class A
{
        public:
                void display()
                {
                        cout << "\nI am display of A";
                }
};
class B : virtual public A
{
        public:
                void f1()
                {
                        cout << "\nI am f1 of B";
                }
};

class C : public virtual  A
{
        public:
                void f2()
                {
                        cout << "\nI am f2 of C";
                }
};

class D : public B, public C
{
        public:

                void f3()
                {
                        cout << "\nI am display of D";
                }
};

main ()
{
        D d ;
        d.display();
}
```

- In this case class A used as a virtual base class for B and C, virtual keyword is apply at the time of declaration of derive class before or after access specifier.

Inheritance with Constructor

- e.g.

```
#include<iostream>
using namespace std;
class base
{
        private:
                int x , y ;
```

Scanned with CamScanner

```
          public :
                    base ()
                    {
                              x = 0 ;
                              y = 0 ;
                    }

                    void display ()
                    {
                              cout<<"\n" << x << y ;
                    }
} ;
class derive : public  base
{
          private:
                    int k ;
          public :
                    derive()
                    {
                              k = 0 ;
                    }
                    void display ()
                    {
                              base::display();
                              cout <<"\n" << k ;
                    }
} ;

main ()
{
          derive obj ;
          obj.display();
}
```

- If we create an object of derive class then immediately control goes to the header derive class constructor.
- But before execute the body of the derive class constructor control goes to its base class *default constructor and a*fter successful execution of the base class *default constructor* control return back to the derive class constructor and Finally execute the body of the derive class constructor

- ***Parametrized constructor with inheritance***

e.g.

```
#include<iostream>
using namespace std;
class base
{
          private:
                    int x , y ;
          public :
                    base ()
                    {
```

```
                x = 0 ;
                y = 0 ;
            }
            base (int i , int j )
            {
                x = i ;
                y = j ;
            }
            void display ()
            {
                cout<<"\n" << x << y ;
            }
};

class derive : public base
{
        private:
            int k ;
        public :
            derive()
            {
                k = 0 ;
            }
            derive(int i , int j , int l )
            {
                k = l;
            }
            void display ()
            {
                base::display();
                cout <<"\n" << k ;
            }
};

main ()
{
        derive a (10,20,30);
        a.display() ;

}
```

- When we create parameterized constructor in derive class we have some extra responsibilities
  - We have to define parameters for inherited member as well.
  - If we do not call base class constructor explicitly from derive class constructors then base class default constructor call first but If we want to execute the base class parameterized constructor, explicitly call base class parameterized constructor otherwise

```
            derive(int i , int j , int l )  : base(i,j)
            {                                 |__explicitly call
                k = l;
```

The Target Institute

(Sheet -1)

1. Define a class ***MyCalc*** having following methods:
   a. **float addition(float a, float b);**
   b. **float subtraction(float a, float b);**
   c. **float multiply(float a, float b);**
   d. **float division(float a, float b);**

2. Define a class ***MyCommonMethods*** that consist of following ***methods***
   a. ***simple_ interest*** that takes three ***parameters p,r,t*** of type float and ***return simple interest using these parameters.***
   b. ***gross_salary*** that takes a parameter named ***sal*** for salary of type float and ***return gross salary where hra 40 % and da 20 % of sal.***
   c. ***leap_year*** that takes a parameter named ***year*** of type int and **return true if year is leap year otherwise return false.**

3. Define a class ***Myfactorial*** having a method ***factorial*** that has a parameter (***n***) and ***return the factorial of n then return it.***

4. Define a class ***MyPower*** having a method power that has two parameter (***a, b***) and ***return a raised to the power of b then return the power. e.g.***

$\frac{a}{2}$ $\frac{b}{5}$ a^b 2^5 = 2*2*2*2*2

5. Create a ***class Book*** consist of following properties and methods
   a. ***Properties :*** it hold the information of book
      i. name
      ii. page
      iii. price
   b. ***getdata()*** method that take name, page and price and store into the properties of object
   c. ***display()*** method print the properties of object.

6. Create a ***class Student*** that consist of following properties and methods of object
   a. ***Properties*** : it hold the information of employee
      i. Rollno
      ii. Name
      iii. percentage
   b. ***getdata*** method that take rollno, name and percentage and store into the properties of object
   c. ***display*** method print the properties of object.

7. Create a ***class Employee*** that consist of following properties and methods of object
   a. **Properties** : it hold the information of employee
      i. Employee no
      ii. Name
      iii. salary
   b. ***setData*** method that set the properties of object using parameters
   c. ***getdata*** method that take empno, name and salary and store into the properties of object
   d. ***display*** method print the properties of object.

8. Create a class Acccount for a account balance information that consist of following properties and methods of object
   a. Properties : it hold the information about account
      i. Balance
   b. void setdata(float balance);
   c. void withdrawal float amt): to withdrawal the amount
   d. void deposit(float amt) :to deposit the amount
   e. void mybalance() : display current balance

9981315087

The Target Institute

Sheet - 1

1. Define a **class Vehicle** which have following characteristics:
   a. **Properties:** , Color, Price, Category.
   b. **Default constructor:**
      i. It initialize default values and Default values as follows
         1. Manufacturing Year is 2018
         2. Color is white
         3. Price is 40000
         4. Category is sedan
   c. **Method():**
      i. **getData():** It take values from user and assign to the respective properties.
      ii. **display():** It shows the values of properties.

   Also make a **self-executable class** and test object of **Vehicle** class.

2. Define a ***Class Employee which creates a payroll system having*** following characteristics:
   a. **Properties:** Employee Name, Employee Code, Designation, Basic Salary, HRA, DA
   b. **Define Default constructor to initialize properties by default values.**
   c. **Methods:**
      i. **getData():** which takes values from users and assign to respective properties
      ii. **grossSalary():** Which calculate gross salary according to HRA,DA.
      iii. **display():** which show all the properties of ***Employee*** and its Gross Salary in output.

   Also make a ***self-executable*** Class to test it.

3. Define a ***class BankAccount*** which includes the following members:
   a. **Data Members:**
      i. Name of the depositor
      ii. Account Number
      iii. Type of Account
      iv. Balance Amount in the account
   b. **Constructor**
      i. To initialize balance with 3000
   c. **Methods**
      ii. **getData() takes values from enduser of name, account number, type of acccount**
      iii. To deposit an amount
      iv. To withdraw an amount after checking the balance.
      v. To display name and balance.

   Also Define a **self-executable** class which test the performance of **BankAccount** .

4. Define a ***Class Library*** which consists of **several properties** such as **Accession Number, Name of The Author, Title of the Book** and **Constructor for initializing** default values to properties of Library.
   Define another ***class Book*** which acquire the properties of class Library and have ***additional data members*** such as **Year of Publication, Publisher's name, Cost of the book.** Class book have **two methods getData()** which takes values for all the properties of Book class and **display()** which shows the properties of book.
   Also make a **self-executable class** which test the performance of your Library Information System.