Interface :- An interface contains one (or) more abstract methods i.e method declarations.

An interface is defined by using the keyword "interface".

General form to define interfaces :-

```
interface        nameOfTheInterface
{
    access  type   methodname1 ( parameter list);
    access  type   methodname2 ( parameter list);
                        ⋮
```

```
        access type methodnameN(parameters list);
        type finalvariable1 = value1;
        type finalvariable2 = value2;
                  ⋮
        type finalvariableN = valueN;
    }
```

* The methods in the interface are by default public & abstract methods
* Here, access specifies "access specifier," the method in the interface can have "public" or "default" access specifier
* The variables of interface are by default public, static & final variables.

Implementing interfaces

* A class can include an interface by using the keyword "implements"
* The general form :-

```
class classname extends superclass implements interface1, interface2, ...
                                                              interfacen
  {
        // body of the class
        // definition of interface methods

  }
```

* When a class implements interface, the implementing class must provide definition (body of the method) for all the methods declared inside the interface, if not, the implementing class must be declared as abstract.

The implementing class must declare the interface methods as "public" while giving the definition.

Interface cannot be instantiated (i.e objects cannot be created) but, we can declare variables of interface.

Write a java program to create interface Figure containing two methods area & display. One implementation class Triangle which includes Figure interface

```
interface figure
{
  double area( double dim1 , double dim2);
  void display( );

}
class Triangle implements Figure

  public double area( double dim1 , double dim2)
  {   return (dim1 * dim2/2);
  }
  public void display ()
  {
      system.out.println ("this is display");
  }
  int a=10;
  int b =20;
  void showab()
  {
      system.out.println (a+ "  "+b);
  }

ss InterfaceDemo

 public static void main (string args[])
 {
   // Figure f = new Figure();          //error
   Triangle t= new Triangle ();
   double d= t.area(10,12);
   System.out.println ("area is" +d);
   t.display ();
 } t.showab();
```

NOTE :-

1) An implementation class can add its own members along with methods of interface

2) An interface can be implemented by any class.

3) Interface provides fully abstraction.

4) Interfaces don't have the state (objects) & not even constructors, abstract class don't have the state but it has constructors, a concrete class has a state and also constructors.

Assigning object to interface variable :-

It is possible to declare variables to interface. These variables can refer objects of implementing classes.

Through this variable, only members known to interface variables can be accessed.

Example program :-

```
interface figure
{
        double area (double dim1, ~~dim~~ double dim2) ;



}
class Rectangle implements figure
{
     public double area ( double dim1, double dim2);
     {
          return (dim1 * dim 2) ;
     }
     int i = 1;
     int j = 2;
     void showij ( )
     {
          System.out.println (i+" "+ j);
     }
}
```

```
class InterfaceDemo
{   public static void main(string args[])
    {
        Rectangle r = new Rectangle();
        Figure f;                        // Figure f = new Rectangle ();
        f = r;
        system.out.println("area is :"+ f.area (10,20));
        f.showij(); //error
    }
}
```

NOTE :-

* With interfaces, we can achieve runtime polymorphism.

Extending interfaces :-

An interface can inherit another interfaces. when a class implements an interface that inherits another interface, it must provide implementations (definitions for all methods required by the interface inheritance chain.

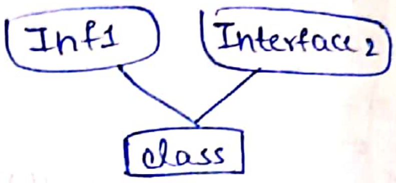
Eg :-

```
interface A
{
    void meth1();
    void meth2();

}
interface B extends A
{
    void meth3();

}
class callback implements B
{
    public void meth1 ()
    {
        s.o.p ("method 1");
    }
    public void meth2 ()
    {   s.o.p ("method 2");
    }
```

```
        public void meth3()
        {
            s.o.p("method 3");
        }
    }
    class ExtendsInterface
    {
        public static void main(String args[])
        {
            Callback ob = new Callback();
            ob.meth1();
            ob.meth2();
            ob.meth3();
        }
    }
```



Nesting of interfaces

An interface can be defined as a member of class (or) other interface. Such an interface is called "member interface" or a "nested interface"

→ A nested interface can have access specifiers : public, private & protected (whereas high level interface can only be declared as public & default)

→ To access the nested interface outside of its enclosing scope, it must be qualified by its name(i.e its enclosing class or interface name. nested interface name)

Write a java program to demonstrate nested interface

```
class A
{
    interface MyIf
    {
        boolean isNegative(int x);
    }
}
```

```
class NestedIf implements A.MyIf
{
    public boolean isNegative (int x)
    {
        return x<0? true : false;
    }
}
class MemberInterfaceDemo
{
  public static void main (string args[])
  {
     NestedIf in = new NestedIf ();

     boolean b= in. isNegative (-10);
     A.MyIf ob = in;
     System.out.println (ob.is Negative (12));
  }
}
```

Output :-

```
true
false
```

Variables in interface :-

We can declare & initialize variables inside interface.

The variables of an interface are by default public, static & final variables.