INTRODUCTION

Sometimes we want to inherit the features of two classes at the same time. But this is not possible in Java because Java does not support multiple inheritance. So in order to accomplish this task, Java provides us with two ways. They are :

1. Either we extend both classes one by one by using simple inheritance.

   Example : If we have 2 classes say I and J. M wants to inherit these classes. Then we can follow this process.

```
class I
    {
        — Body of class —
    }
class J extends I
    {
        — Body of class —
    }
class M extends J
    {
        — Body of class —
    }
```

   So in this way class M can inherit features of class J as well as class I. class M can inherits features of class I because class J (base class of M) is inheriting features of class I.

2. Second way by which you can implement multiple inheritance is using Interfaces. Interfaces are just like class declaration. They specify what all methods they will contain but do not give method definition. They may contain any number of methods. One class can implement as many interfaces it wants, so idea of multiple inheritance is achieved.

   In this chapter you will study about Interfaces, how they are defined, how they are implemented, how they are extended and how interface variable are accessed.

In this chapter you will also study about a very important feature of Java i.e. packages, how they are used and how classes are added to packages.

INTERFACES

Interfaces are similar to classes. As class contains methods and variables, interfaces also contain. But the difference is that in class you can define any type of variable and body of method but in interfaces you only define methodname and its parameter i.e. you can only declare them but you can not define body of method. They do not have instance variables as class does have. Variables that are declared inside interfaces are implicitly final and static which means they are constants and they cannot be changed by the class implementing interface. If we consider syntax for declaring interface then it is same as that of class.

Interface is declared by the use of interface keyword. Once an interface is defined, any number of classes can implement it. A class can implement as many interfaces it requires. If a class wants to implement an interface, then it is essential for a class to define all methods declared in the interface. If more than one class is implementing the interface then each of the class have freedom to define methods of interface as per their choice. There is no such rule that all classes should have one definition for interface method.

DEFINING INTERFACE

General form for defining an interface is :

```
access interface name
    {
        type method1(parameter_list);
        type method2(parameter_list);
        type final var1 = value;
        type final var2 = value;
        :
        :
        type methodN(parameter_list);
        type final varN = value;
    }
```

Here access is specified as public, if public access is not specified then no other access is used. It means that interface is accessed according to the rule of default (friendly) access specifier. When an interface is specified as public then it means that it can be accessed by any code either inside or outside the package. If no access is specified, then interface can be implemented by members of its package only.

Interface is the keyword that tells the compiler that it is an interface and name is any valid variable name given to interface. Interface is referred by its name. Curly braces { } indicate that body of interface starts from { and ends with }.

Then comes declaration of methods, type is the return type of method, method1, method2... methodN are names given to method. You can specify any name as per your choice. Methods only contain parameter list, they do not contain any body. They are followed by semicolon. i.e. declaration of methods end with a semicolon. An interface can contain many methods. A class should implement all methods of an interface, if it is implementing an interface. So this gives concept of Abstract classes. A class extending abstract classes must define all methods of Abstract class. Similarly, a class implementing interface must implements all methods of interface. But there is difference between abstract class and inheritance, in inheritance every method, variable whatever you include is incomplete while abstract class is a class in which you can define methods, but you must include at least one method that should be abstract.

Var1, Var2, VarN are names of variable. Again you can give any variable name. All variables are implicitly of static type & final. It means their values cannot be changed by the class implementing interface. Variables are initialized with constant value in an interface.

All the methods and variables declared inside interface are implicitly public, if interface is declared public.

Example :

```
public interface Example
    {
        void example(int value);
    }
```

This example creates a simple interface named as Example. It includes only one method named as example() that takes single integer parameter.

IMPLEMENTING INTERFACE

If a class wants to implement an interface, then it can implement an interface by using implements keyword.

General form:

```
class classname implements interfacename
    {
        Body of class
    }
```

Once a class implements an interface, then it creates the method defined by the interface. Here classname is name of class. When you implement an interface's method then it should be declared as public.

A class can implement more than one interface. If a class implements more than one interface, then interfaces are separated with comma as :

```
class classname implements interface1, interface2,
```

Now class will creates all methods defined in interface1 and interface2. If interface1 and interface2 declare the same method, then the same method will be used by clients of either of two interfaces. You can only define methods, but you can't change return type of method. It should be strictly same as defined in interface.

Let's define a class that implements Example interface.

Program : Creating a class that implements example interface.

```
class Show implements Example
    {
        public void example (int value)
```

```
        {
            System.out.println("Using interface");
            System.out.println("Value=" +value);
        }
    }
```

ACCESSING INTERFACE METHOD

Here we have just implement an interface. We can access methods of interface by declaring variables that use an interface not a class. These variables refer to the instance of class that is implementing interface eg :

First we create variable of Example type as :

Example ex;

ex = ob [Here ob is object of class Show]

Now we can refer to example with the help of ex as :

ex.example (20);

This statement will call example method defined by class Show and assigns 20 to value.

This method will call method defined in show class because interface variable points to ob and ob is object of Show class. Let's create a program that implement an interface and access its methods also.

Program : Implementing an interface and accessing its method also.

```
class Show implements Example
    {
        public void example (int e)
            {
                System.out.println ("Implementing interface");
                System.out.println ("Value =" +e);
            }
    }
class Test
    {
        public static void main (String args[ ])
        {
            Show ob = new Show( );
            Example ex;
            ex = ob;
            ex.example (20);
        }
    }
```

OUTPUT

Implementing interface
value = 20

It's not like that if a class is implementing an interface, then it can define members of only interface. It is very common for the class to define additional members of its own. Let's see an example.

Program : Defining a class that defines interface members as well as additional members of its own.

```
public interface Display
    {
        void value (int a, int b);
    }
class Show implements Display
    {
        public void value(int a, int b)
        {
            int c = a + b;
            System.out.println ("Sum=" + c);
        }
        void val( )
        {
            System.out.println ("Class using interface");
        }
    }
class Test
    {
        public static void main (String args[ ])
        {
            Show ob = new Show( );
            Display d;
            d = ob;
            ob.val( );
            d.value(2, 5);
        }
    }
```

Output

Class using interface
sum = 7

One thing that you should always remember is, when a class define its own members, then they are accessed by its own object interface variables that contains object reference cannot access then. It is because an interface reference variable contains knowledge about the methods that are declared by its interface declaration.

As in the above program d can be used to access value( ) as it is a method declared by its interface. Although d was assigned an instance of class Show, it cannot be used to access val( ) method of Show class. It is because d only knows about value( ) method declared in the interface Display as d is a variable of Display type. So d will not have any information about val( ) method which is a member of Show class.

Program : Declaring two methods in an interface

```
public interface Example
  {
      void calc (int a, int b);
      void show( );
  }
class Display implements Example
  {
  public void calc (int a, int b)
    {
        int c = a*b;
        System.out.println ("Product = "+ c);
    }
  public void show( )
    {
        System.out.println ("You can declare more than method interface");
    }
}
class Test
  {
      public static void main (String args[])
      {
          Display ob = new Display( );
          Example e;
          e = ob;
          e.show( );
          e.calc(4, 5);
      }
  }
```

Output

You can declare more than one method in interface

Product = 20.

Program : Declaring two classes implementing same interface.

```
public interface Example
    {
        void show( );
    }
class Display implements Example
    {
        public void show( )
        {
            System.out.println("Under Display Class");
        }
    }
class Second implements Example
    {
        public void show( )
        {
            System.out.println ("Under second class");
        }
    }
class Test
    {
        public static void main (String args[ ])
        {
            Display
            ~~Example~~ e = new Display( );  // Points to display object.
            Second obj = new Second( );
            e.show( );
            e = obj;    //e now print to object of second class
            e.show( );
        }
    }
```

OUTPUT

Under Display class

Under Second class.

When two classes implement same interface then version of method (declared in interface) that will be called depends upon the type of object that interface variable refers to at run time. So if we consider above example, then firstly interface variable is referring to object of display class, so at that time it will call version of show( ) defined in Display. Then we change reference to object of Second class, now it will refer to version of show( ) defined in Second class. So version of method that will be called depends upon the type of object interface variable is refering at runtime.

Program : Declaring a class implementing two interfaces

```
interface Example1
  {
     void show( );
  }
interface Example2
  {
     void display( );
  }
class Using implements Example1, Example2
  {
  public void show( )
     {
        System.out.println ("Showing multiple inheritance");
     }
  public void display( )
     {
        System.out.println ("Implementing two interfaces");
     }
  }
  class Test
     {
        public static void main (String args[ ])
        {
           Using ob = new Using( );
           Example1 e;
           e = ob;
           e.show( );
           Example2 e1;
           e1 =  ob;
           e1.display( );
        }
     }
```

OUTPUT

Showing multiple inheritance
Implementing two interfaces.

ou can also call interface methods by use of object of class that is defining them.

Program : Calling interface methods by use of class object.

```
public interface Example
    {
        void sum (int a, int b);
    }
class Show implements Example
    {
public void sum (int a, int b)
        {
            int c = a+b;
        System.out.println ("Sum =" +c);
        }
    }
class Test
    {
        public static void main(String args[ ])
        {
            Show obj = new Show( );
            obj.sum(2,7);   //calling interface method by show object
        }
    }
```

OUTPUT

Sum = 9

It is possible for a class to implement an interface a extend a class at one time.

General form :

class classname extends classname1 implements interface1.

Program : Showing a class that implements an interface and extends a class at the same time.

```
interface Example
    {
        void show (int d);
    }
class First
    {
        int a, b, c;
        public void sum( )
        {
            a = 50;
            b = 79;
            c = a + b;
```

```
        System.out.println( "Sum =" +c);
        }

    }
class Second extends First implements Example
    {
        public void show(int d)
        {
            System.out.println("Value of d =" + d);
        }


    }
class Test
    {
        public static void main (String args[ ])
        {
            Second ob = new Second( );
            ob.show (9);
            ob.sum( );
        }
}
```

OUTPUT

Value of d = 9

Sum = 129

When a class extend another class and implements an interface, then it can use all methods defined in class and interface.

A class can extends another class and can implement more than one interface at one time.

General form

class classname extends anotherclass implements interface1, interface2

Program : Showing a class that can implement more than one interfaces and extend a class at the same time.

```
interface Example1
    {
        void cal(int a,int b);
    }
interface Example2
    {
        void show(int a);
    }
class A
```

```
    {
        void name()
        {
            System.out.println ("Name is Nandini");
        }
}
class Display extends A implements Example1,Example2
    {
        public void cal(int a,int b)
        {
        int c = a *b;
        System.out.println("Product =  "+c);
        }
        public void show(int a)
        {
        System.out.println ("value of a = "+a);
        }
}
class Test
    {
        public static void main (String args[ ])
        {
            Display ob = new Display( );
            ob.show(29);
            ob.cal(59,70);
            ob.name();
        }
    }
```

OUTPUT

Value of a = 29

Product = 4130

Name is Nandini

ACCESSING INTERFACE VARIABLES

Interfaces can only be used to declare constants. Values given to variables of interfaces cannot be changed by class implementing these interfaces, but class implementing the interface can use any of these values. These values can be used by any method. Methods can use these constants for variable declaration, or as case constants, or in any form.

Program : Accessing interface variables inside a class.

```
interface Example
{
    int VALUE = 20;
}
class Show implements Example
{
    int a = VALUE;
    void display( )
    {
        System.out.println ("Value of a =" +a);
    }
}
class Test
{
    public static void main (String args[ ])
    {
        Show ob = new Show( );
        ob.display( );
    }
}
```

Output

Value of a = 20

Program : Accessing interface variables as case constants.

```
interface Values
{
    int A = 1;
    int B = 2;
    int C = 3;
    int D = 4;
}
class Show implements Values
{
    int choice = 3;
    void select( )
    {
        switch(choice)
        {
        case A :
            System.out.println("A");
```

```
            break;
        case B :
            System.out.println("B");
            break;
        case C :
            System.out.println("C");
            break;
        case D :
            System.out.println("D");
            break;
        default :
            System.out.println("Default Case");
        }
    }
}
class Test
{
    public static void main (String args[ ])
    {
        Show ob = new Show( );
        ob.select ( );
    }
}
```

OUTPUT

C

EXTENDING INTERFACES

As one class can inherits property of another class, similarly one interface can inherit another interface by using extends keyword.

General form :

interface interface2 extends interface1

So if a class implements a interface that inherits another interface, then it must define all methods of all interfaces that come in inheritance chain.

Program : Showing concept of extending interfaces.

```
interface I
    {
        void show1( );
```

```
    void show2( );
  }
interface J extends I
  {
    void show3( );
  }
class Display implements J
  {
    public void show1( )
    {
      System.out.println("Inside show1");
    }
    public void show2( )
    {
      System.out.println("Inside show2");
    }
    public void show3( )
    {
      System.out.println("Inside show3");
    }
  }
class Test
  {
    public static void main (String args[ ])
    {
      Display obj = new Display( );
      obj.show1( );
      obj.show2( );
      obj.show3( );
    }
  }
```

OUTPUT

Inside Show1
Inside Show2
Inside Show3

As stated earlier, you need to define all methods of all interfaces that come inside inheritance chain. So here class Display will define all methods of interface I and J. If it would have not define any of these methods then it would give complie-time error.