# CHAPTER 7

# MULTIPLE INHERITANCE - INTERFACES

- The interface in Java is a mechanism to achieve abstraction. There can be only abstract methods in the Java interface, not the method body.
- Java does not support multiple inheritance. That is, classes in Java cannot have more than one superclass.
- It is used to achieve abstraction and multiple inheritances in Java using Interface. In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.

**Uses of Interfaces in Java:**

- It is used to achieve total abstraction.
- Java does not support multiple inheritances in the case of class, by using an interface it can achieve multiple inheritances.
- Any class can extend only 1 class, but can any class implement an infinite number of interfaces.
- It is also used to achieve loose coupling.
- Interfaces are used to implement abstraction.

## Defining interfaces

- An interface is basically a kind of class. Like classes, interfaces contains methods and variables but with a major difference.
- The difference is that interfaces define only abstract methods and final fields. This means that interfaces do not specify any code to implement these methods and data fields contain only constant

**The general form of an interface definition**

```
interface InterfaceName
{
    variables declaration;
    methods declaration;
}
```

**Example:**

```
interface  Item
{
    static final int code = 1001;
    static final String name = "Fan";
    void display ( ) ;
}
```

**Variables are declared as follows**

static final type VariableName= Value;

- Methods declaration will contain only a list of methods without any body statements.

**Example:**

return-type methodNamel(parameter_list);

## Extending Interfaces :

- Like classes, interfaces can also be extended. That is, an interface can be sub interfaced from other interfaces.
- The new sub interface will inherit all the members of the super interface in the manner similar to subclasses.
- This is achieved using the keyword **extends**

```
interface name2 extends name1
{
    body of name2
}
```

For example, we can put all the constants in one interface and the methods in the other. This will enable us to use the constants in classes where the methods are not required.

```
interface  ItemConstants
{
      int code = 1001;
      string name = "Fan";
}
interface Item extends ItemConstants
{
      void display ( );
}
```

```
interface ItemConstants
{
    int code = 1001;
    String name = "Fan";
}
interface ItemMethods
{
    void display( );
}
interface Item extends ItemConstants, ItemMethods
{
    ..........
    ..........
}
```

While interfaces are allowed to extend to other interfaces, sub interfaces cannot define the methods declared in the super interfaces

## Implementing interfaces

Interfaces are used as "superclasses" whose properties are inherited by classes. It is therefore necessary to create a class that inherits the given interface. This is done as follows

```
class classname implements Interfacename
{
    body of classname
}
```

Here the class classname“ implements" the interface interfacename.

## A general form of implementation:

```
class classname  extends superclass
      implements interface1,interface2, ......
{
    body of classname
}
```

```
// InterfaceTest.java
interface Area                                    // Interface defined
{
  final static float pi = 3.14F;
  float compute (float x, float y);
}
class Rectangle implements Area                   // Interface implemented
{
  public float compute (float x, float y)
  {
    return (x*y);
  }
}
class Circle implements Area                      // Another implementation
{
  public float compute (float x, float y)
  {
    return (pi*x*x);
  }
}
class InterfaceTest
{
  public static void main(String args[ ])
  {
    Rectangle rect = new Rectangle( );
    Circle cir = new Circle( );

    Area area;                                    // Interface object

    area = rect;
    System.out.println("Area of Rectangle = "
                                  + area.compute(10,20));

    area = cir;
    System.out.println("Area of Circle = "
                                  + area.compute(10,0));
  }
}
```

# Accessing interface

- Variables Interfaces can be used to declare a set of constants that can be used in different classes.
- Interfaces do not contain methods, there is no need to worry about implementing any methods.
- The constant values will be available to any class that implements the interface.
- The values can be used in any method, as part of any variable declaration, or anywhere we can use a final value.

**Example:**

```
interface A
{
    int m = 10;
    int n = 50;
}

class B implements A
{
    int x = m ;
    void methodB(int size)
     {
        ..................
        ..................
        if (size < n)
     }
}
```