Introduction : Interface

- ✓ Similar to a class.
- ✓ Consists of only abstract methods and final variables.
- ✓ Any number of classes can implement an interface.
- ✓ One class can implement any number of interfaces.
- ✓ To implement an interface a class must define each of the method declared in the interface. Each class can also add new features.
- ✓ Interface disconnect the definition of a method or set of methods from the inheritance hierarchy.

Defining an Interface

✓ General form of an interface:

access interface *name* {

ret-type *method1*(parameter list);

ret-type *method2*(parameter list);

⋮

type final *var1* = value;

type final static *val2* = value;

}

Example:

```
interface callback{
    void callback (int param);
}
```

Application: Defining an Interface

- ✓ *Access* is either **public** or **default**.
- ✓ Variables declared inside an interface are implicitly **final** and **static.**
- ✓ Variables must be initialized with a constant value.
- ✓ All methods and variables are implicitly **public** if the interface, itself, is declared as **public**.

Implementing Interfaces

- ✓ The General Form:

  access class *classname* [extends *superclass*][implements *interface*[,*interface*]] {

  }
- ✓ The methods that implement an interface must be declared *public*.
- ✓ The type signature of the implementing method must match exactly the type signature specified in the interface definition.

Accessing Implementations through Interface Reference

- ✓ Interface reference is required to access the implementation.
- ✓ Any instance of the class that implements the interface can be referred to by such a variable.
- ✓ When a method is called through one of the reference, the correct version will be called based on the actual instance of the interface being referred to.
- ✓ The method to be executed is looked up dynamically at run time.

Example-1

```
interface call
{
    void callback(int param);
}

class client implements call
{
    public void callback(int p)
    {
        System.out.println("callback called with "+p);
    }
}

public class testIface
{
    public static void main(String args[])
    {
        call c = new client();
        c.callback(423);
    }
}
```

Example-2

```
interface call
{
        void callback(int param);
}

class client implements call
{
        public void callback(int p)
        {
System.out.println("callback is called with "+p);
        }
}
class anotherclient implements call
{
        public void callback(int p)
        {
          System.out.println("p squred is "+(p*p));
        }
}
```

```
public class testIface
{
public static void main(String args[])
        {
                call c = new client();
                c.callback(42);
                c=new anotherclient();
                c.callback(10);
        }
}
```

Partial Implementation

- ✓ If a class includes an interface but does not fully implement the methods defined by that interface, then that class must be declared as **abstract**.
- ✓ Example:

```
abstract class temp implements call{
int a, b;
void show()
{
	//body of the method
}
}
```

- ✓ Any class that inherits *temp* must implement callback() or declared abstract itself.

Extending Interfaces

- ✓ One interface can inherit another by using the keyword **extends**.
- ✓ The new sub interface will inherit all the member of the super interface.
- ✓ Any class that will implement the interface that inherits another interface, it must provide implementations of all methods defined within the interface inheritance chain.
- ✓ General form:

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

- ✓ Example:

```
interface ItemConstant
{
        int code =1001;
        String name =“Pen”;
}
```

```
interface Item extends ItemConstant
{
        void display();
}
```

✓An interface cannot extends a class.

Multiple Inheritance Using Interface

- ✓ Java supports multiple inheritance through the use of interface.
- ✓ Care should be taken to avoid some conflicts.

Example-3

```
interface test1
{
    int val=10;
    void display();
}
interface test2
{
    int val=20;
    void display();
}
```

```
class test3 implements test1, test2
{
    public void display()
    {
    System.out.println(“In test3”);
    System.out.println(test1.val);
    System.out.println(test2.val);
    }
}
```

Example-4

```
interface test1
{
    int val=10;
    void display();
}
interface test2
{
    int val=20;
    void display();
}
interface test3 extends test1, test2
{
    int val=50;
    void display();
}
```

```
class test4 implements test3
{
    int val=57;
    public void display()
    {
    System.out.println(test1.val);
    System.out.println(test2.val);
    System.out.println(test3.val);
    System.out.println(val);

    }
}
public class Iface_test
{
public static void main(String args[])
{
    test4 ob = new test4();
    ob.display();
}
}
```

Example-5

```
interface test1
{
    int val=33;
    void display();
}
class test2 implements test1
{
    static int val=34;
    void display()
    {
    System.out.println(test1.val);
    System.out.println(val);
    }
}
```

```
class test3 extends test2
{
    int val=35;
    void show()
    {
    System.out.println(test1.val);
    System.out.println(test2.val);
    System.out.println(val);
    }
}
class test4
{
    public static void main(String args[])
    {
    test3 ob = new test3();
    ob.show();
    }
}
```