Java means DURGA SOFT..

CORE JAVA

Material

Java™

India's No.1 Software Training Institute

DURGASOFT

www.durgasoft.com Ph: 9246212143 ,8096969696

Interfaces

1. Interface is also one of the type of class it contains only abstract methods. And Interfaces not alternative for abstract class it is extension for abstract classes.
2. The abstract class contains atleast one abstract method but the interface contains **only abstract methods.**
3. For the interfaces the compiler will generates .class files.
4. Interfaces giving the information about the functionalities and these are not giving the information about internal implementation.
5. Inside the source file it is possible to declare any number of interfaces. And we are declaring the interfaces by using **interface** keyword.

Syntax:-Interface interface-name

interface it1 { }

andbydefault above three methods are public

the interface contains constants and these constants by default **public static final**

Note-1 :- if u dont no the anything about implementation just we have the requirment specification them we should go for inteface

Note-2:- If u know the implementation but not completly then we shold go for abstract class

Note-3 :-if you know the implementation completly then we should go for concreate class

Both examples are same

```
Interface it1
{
        Void m1();
        Void m2();
        Void m3();
}
```

```
abstract interface it1
{
public abstract void m1();
public abstract void m2();
public abstract void m3();
}
```

Note: - If we are declaring or not each and every interface method by default public abstract. And the interfaces are by default abstract hence for the interfaces object creation is not possible.

Example-1 :-

- Interface constrains abstract methods and by default these methods are "public abstract ".
- Interface contains abstract method for these abstract methods provide the implementation in the implementation classes.
- Implementation class is nothing but the class that implements particular interface.
- While providing implementation of interface methods that implementation methods must be public methods otherwise compiler generate error message "***attempting to assign weaker access privileges".***

```
interface it1      // interface declaration
{       Void m1();  //abstract method by default [public abstract]
        Void m2();//abstract method by default [public abstract]
        Void m3();//abstract method by default [public abstract]
}
Class Test  implements it1              //Test is implementation class of It1 interface
{       Public void m1()                //implementation method must be public
        {       System.out.println("m1-method implementation ");      }
        Public void m2()
        {       System.out.println("m2-method implementation");      }
        Public void m3()
        {       System.out.println("m3 –method implementation");     }
        Public static void main(String[] args)
        {       Test t=new Test();
                t.m1();         t.m2();         t.m3();
        }
}

interface It1      //abstract
{       void m1(); //public abstract
        void m2();
        void m3();
}
class Test implements It1
{       public void m1(){System.out.println("m1 method");}
        public void m2(){System.out.println("m2 method");}
        public void m3(){System.out.println("m3 method");}

        public static void main(String[] args)
        {       Test t = new Test();
                t.m1();         t.m2();         t.m3();

                It1 i = new Test();
                i.m1();  //compile : It1  runtime : Test
                i.m2();  //compile : It1  runtime : Test
```

```
                i.m3();  //compile : It1  runtime : Test
        }
};
```

Example-2:-

- Interface contains abstract method for these abstract methods provide the implementation in the implementation class.
- If the implementation class is unable to provide the implementation of all abstract methods then declare implementation class with abstract modifier, take child class of implementation class then complete the implementation of remaining abstract methods.
- In java it is possible to take any number of child classes but at final complete the implementation of all abstract methods.

```
interface it1       // interface declaration
{       Void m1();  //abstract method by default [public abstract]
        Void m2();//abstract method by default [public abstract]
        Void m3();//abstract method by default [public abstract]
}
//Test1 is abstract class contains 2 abstract methods m2() & m3()hence object creation not possible
abstract class Test1 implements it
{       public void m1()
        {       System.out.println("m1 method");       }
};


//Test2is abstract class contains 1 abstract methodm3()hence object creation not possible
abstract class Test2 extends Test1
{       public void m2()
        {       System.out.println("m2 method");       }
};
//Test3 is normal class because it contains only normal methods hence object creation possible
class Test3 extends Test2
{       public void m3()
```

```
        {       System.out.println("m3 method");       }
        public static void  main(String[] args)
        {       Test3 t = new Test3();
                t.m1();         t.m2();         t.m3();
        }
};
interface It1   //abstract
{       void m1(); //public abstract
        void m2();
        void m3();
}
abstract class Test implements It1
{       public void m1(){System.out.println("m1 method");}
};
abstract class Test1 extends Test
{       public void m2(){System.out.println("m2 method");}
};
class Test2 extends Test1
{       public void m3(){System.out.println("m3 method");}
        public static void main(String[] args)
        {       Test2 t = new Test2();
                t.m1();
                t.m2();
                t.m3();
        }
};
```


Example 3:-

The interface reference variables is able to hold child class objects.

```
interface It1           // interface declaration
{       void m1();      //abstract method by default [public abstract]
        void m2();      //abstract method by default [public abstract]
```

```
        void m3();      //abstract method by default [public abstract]
}
//Test1 is abstract class contains 2 abstract methods m2() m3()hence object creation not possible
abstract class Test1 implements It1
{       public void m1()
{       System.out.println("m1 method");    }
};
//Test2  is abstract class contains 1 abstract method m3()  hence object creation not possible
abstract class Test2 extends Test1
{       public void m2()
{       System.out.println("m2 method");    }
};
//Test3 is normal class because it contains only normal methods hence object creation possible
class Test3 extends Test2
{       public void m3()
        {       System.out.println("m3 method");    }
        public static void  main(String[] args)
        {       It1 t = new Test3();            t.m1();         t.m2();         t.m3();
                Test1 t1 = new Test3();         t1.m1();        t1.m2();        t1.m3();
                Test2 t2 = new Test3();         t2.m1();        t2.m2();        t2.m3();
        }
};


interface It1
{       void m1();
}
interface It2
{       void m2();
}
interface It3 extends It1,It2
{       void m3();
}
class Test implements It1
{       1 method
};
class Test implements It1,It2
{       2 methods
};
class Test implements It1,It2,It3
{       3 methods
};
```

Difference between abstract classes & interfaces:-

Abstract class

1) The purpose of abstract class is to specify default functionality of an object and lest its sub classes explicitly implement that functionality. It stands it is providing abstraction layer that must be extended and implemented by the corresponding sub classes.

2) An abstract class is a class that declared with **abstract** modifier.

Ex: ***abstract class A***
{ abstract void m1(); }

3) The abstract allows declaring both abstract & concrete methods.

4) Abstract class methods must declare with abstract modifier.

5) If the abstract class contains abstract methods then write the implementations in child classes.

6) In child class the implementation methods need not be public it means while overriding it is possible to declare any valid modifier.

7) The abstract class is able to provide implementations of interface methods.
8) One java class is able to extends only one abstract class at a time.
9) Inside abstract class it is possible to declare main method &constructors.
10) It is not possible to instantiate abstract class.
11) For the abstract classes compiler will generate **.class** files.
12) The variables of abstract class need not be **public static final.**

Interface

1. It is providing complete abstraction layer and it contains only declarations of the project then write the implementations in implementation classes.

2. Declare the interface by using **interface** keyword.
Ex:-***interface It1***
{ void m1();}

3. The interface allows declaring only abstract methods.

4. Interface methods are by default **public abstract.**

5. The interface contains abstract methods writ the implementations in implementation classes.

6. In implementation class the implementation methods must be public.

7. The interface is unable to provide implementation of abstract class methods.
8. One java class is able to implements multiple interfaces at a time.
9. Inside interface it is not possible to declare methods and constructors.
10. It is not possible to instantiate interfaces.
11. For the interfaces compiler will generate .class files.
12. The variables declared in interface by default **public static final.**

Example-1:-

- Inside the interfaces it is possible to declare variables and methods.
- By default interface methods are **public abstract** and by default interface variables are **public static final.**
- The final variables are replaced with their values at compilation time only.

Application before compilations:-(.java file)

```
interface It1           //interface declaration
{       int a=10;       //interface variables
        void m1();      //interface method
}
class Test implements It1
{       public void m1()
        {       System.out.println("m1 method");
                System.out.println(a);
        }
        public static void main(String[] args)
        {       Test t = new Test();
```

```
                t.m1();
        }
};
```

Application after compilation:-(.class file)

```
interface It1
{       public abstract void m1();// compiler generate public abstract
        public static final int a = 10;//public static final generated by compiler
}
class Test implements It1
{       public void m1()
        {       System.out.println("m1 method");
                System.out.println(10);//a is final variable hence it replaced at compilation time only
        }
        public static void main(String[] args)
        {       Test t = new Test();
                t.m1();
        }
};
```

Message.java:-

```
packagecom.sravya.declarations;
public interface Message
{       void morn();
        void even();
        voidgn();
}
```

Helper.java-

```
packagecom.sravya.helper;
importcom.sravya.declarations.Message;
public abstract class Helper implements Message
{       public void gn(){System.out.println("good night from helper class");}
}
```

TestClient1.java:-

```
packagecom.sravya.client;
importcom.sravya.declarations.Message;
class TestClient1 implements Message
{       public void morn(){System.out.println("good morning");}
        public void even(){System.out.println("good evening");}
        public void gn(){System.out.println("good 9t");}
        public static void main(String[] args)
        {       TestClient1 t = new TestClient1();
                t.morn();
                t.even();
                t.gn();
        }
}
```

TestClient2.java:-

```
packagecom.sravya.client;
```

```
importcom.sravya.helper.Helper;
class TestClient2 extends Helper
{       public void morn(){System.out.println("good morning");}
        public void even(){System.out.println("good evening");}
        public static void main(String[] args)
        {       TestClient2 t = new TestClient2();
                t.morn();
                t.even();
                t.gn();
        }
}
D:\>javac -d . Message.java
D:\>javac -d . Helper.java
D:\>javac -d . TestClient1.java
D:\>javac -d . TestClient2.java

D:\>java com.sravya.client.TestClient1
good morning
good evening
good 9t

D:\>java com.sravya.client.TestClient2
good morning
good evening
good night from helper class
```

Real time usage of packages:-

Message.java:-

```
packagecom.dss.declarations;
public interface Message
{       void msg1();
        void msg2();
}
```

BusinessLogic.java:-

```
packagecom.dss.businesslogics;
importcom.dss.declarations.Message;
public class  BusinessLogic implements Message
{       public void msg1(){System.out.println("i like you");}
        public void msg2(){System.out.println("i miss you");}
}
```

TestClient.java:-

```
packagecom.dss.client;
importcom.dss.businesslogics.BusinessLogic;
```

```
class  TestClient
{       public static void main(String[] args)
        {       BusinessLogic b = new BusinessLogic();
                b.msg1();
                b.msg2();
                Message b1 = new BusinessLogic();
                b1.msg1();
                b1.msg2();
        }
}
```


Interfaces vs. inheritance :-

Example :-

```
interface it1       //it contains 2 methodsm1() & m2()
{       public abstract void m1();
        public abstract void m2();
}
interface it2 extends it1          // it contains 4 methods m1() & m2() & m3() & m4()
{       public abstract void m3();
        public abstract void m4();
}
interface it3 extends it2          // it contains 6 methods m1() & m2() & m3() & m4() & m5() & m6
{       public abstract void m5();
        public abstract void m6();
}
interface it4 extends it3 // it contains 7  methods m1() & m2() & m3() & m4() & m5() & m6 & m7()
{       public abstract void m7();
}
```

Case 1:

```
class Test implements it1
{       provide the implementation of 2 methods          m1() & m2()
```

```
};
```
Case 2:
```
class Test implements it2
{       provide the implementation of 4 methods          m1() & m2() & m3() & m4()
};
```
Case 3:-
```
class Test implements it3
{       provide the implementation of 6 methods          m1() & m2() & m3() & m4() & m5() & m6()
};
```
Case 4:-
```
class Test implements it4
{       provide the implementation of 7 methodsm1() & m2() & m3() & m4() & m5() & m6() & m7()
};
```
Case 6:-
```
class Test implements it1,it2 //one class  is able to implements multiple interfaces
{       provide the implementation of 4 methods          m1() & m2() & m3() & m4()
};
```
Case 7:-
```
class Test implements it1,it3
{       provide the implementation of 6 methods          m1() & m2() & m3() & m4() & m5() & m6
};
```
Case 8:-
```
class Test  implements it2,it3
{       provide the implementation of 6 methods          m1() & m2() & m3() & m4() & m5() & m6
};
```
Case 9:-
```
class Test   implements it2,it4
{       provide the implementation of 7 methodsm1() & m2() & m3() & m4() & m5() & m6 & m7()
};
```
Case 10:-
```
class Test   implements it1,it2,it3
{       provide the implementation of 6 methods          m1() & m2() & m3() & m4() & m5() & m6
};
```
Case 11:-
```
class Test   implements it1,it2,it3,it4
{       provide the implementation of 7 methodsm1() & m2() & m3() & m4() & m5() & m6 & m7()
};
```
Nested interfaces:-

Example :- declaring interface inside the class is called nested interface.
```
class A
{       interface it1//nested interface declared in A class
        {       void add();     }
};
class Test implements  A.it1
{       public void add()
        {       System.out.println("add method");
        }
```

```
        public static void main(String[] args)
        {       Test   t=new Test();       t.add();
        }
};
```

Example :- it is possible to declare interfaces inside abstract class also.

```
abstract class A
{       abstract void m1();
        interface it1   //nested interface declared in A class
        {       void m2();      }
};
class Test  implements  A.it1
{       public void m1()
        {       System.out.println("m1 method");
        }
        public void m2()
        {       System.out.println("m2 method");
        }
        public static void main(String[] args)
        {       Test t=new Test();
                t.m1();   t.m2();
        }
};
```

Ex:- declaring interface inside the another interface is called nested interface.

```
interface it2
{       void m1();
        interface it1
        {       void m2();      }
};
class Test2 implements it2.it1
{       public void m1()
        {       System.out.println("m1 method");       }
        public void m2()
        {       System.out.println("m2 method");       }
        public static void main(String[] args)
        {       Test2 t=new Test2();
                t.m1();  t.m2();
        }
};
```

Marker interface :-

- An interface that has no members (methods and variables) is known as marker interface or tagged interface or ability interface.
- In java whenever our class is implementing marker interface our class is getting some capabilities that are power of marker interface. We will discuss marker interfaces in detail in later classes.

Ex:-serializable , Cloneable , RandomAccess…etc

Note: - user defined empty interfaces are not a marker interfaces only, predefined empty interfaces are marker interfaces.

Possibility of extends & implements keywords:-

class extends class
class implements interface
interface extends interface

class A extends B =====>valid
class A extends B,C ====>invalid
class A implements it1 ====>valid
class A implements it1,it2,it3 ====>valid

interface it1 extends it2 ----->valid
interface it1 extends it2,it3 ----->valid
interface it1 extends A ---->invalid
interface it1 implements A ---->invalid

class A extends B implements it1,i2,it3 =====>valid (extends must be first)
class A implements it1 extends B =====>invalid

class extends class
interface extends interface
class implements interface

class A extends B ---->valid
class A extends B,C ---->Invalid
class A implements It ---->valid
class A implements It1,It2 ---->valid
class A extends It ---->Invalid
class A extends A -----Invalid

interface It1 extends It2 --->valid
interface It1 extends It2,It3 --->valid
interface It1 extends A --->invalid
interface It1 implements It2 --->invalid
interface It1 extends It1 ---->invalid

class A extends B implements It1,It2 --->valid [extends first]
class A implements It1,It2 extends B--->Invalid

Adaptor class:-

It is aintermediate class between the interface and user defined class. And it contains empty implementation of interface methods.

Example:-

```
interface It                    // interface
{       void m1():
        void m2();
        ;;;;;;;;
        void m100();
}
class X implements It    //adaptor class
{       public void m1(){}
        public void m2(){}
        ;;;;;;;;
        public void m100{}
};
//user defined class implementing interface
class Test implements It
{       must provide 100 methods impl
};
//user defined class extending Adaptor class(X)
class Test extends X
{ override required methods because already X contins empty implementions
};
```

Adaptor class realtime usage:-

```
Message.java:-                  interface
packagecom.dss.declarations;
public interface Message
{       void morn();     // by default public abstarct
        void eve();
```

```
        void night();
}
```

HelperAdaptor.java:- ***Adaptor class***

```
packagecom.dss.helper;
importcom.dss.declarations.Message;
public class  HelperAdaptor implements Message
{       public void night(){}
        public void eve(){}
        public void morn(){}
}
```

GoodStudent.java:-

```
packagecom.dss.bl;
importcom.dss.declarations.Message;
public class  GoodStudent implements Message
{       public void morn(){System.out.println("good morning ratan");}
        public void eve(){System.out.println("good evening ratan");}
        public void night(){System.out.println("good nightratan");}
}
```

TestClient.java:-

```
packagecom.dss.client;
importcom.dss.bl.GoodStudent;
importcom.dss.bl.Student;
classTestClient
{       public static void main(String[] args)
        {       GoodStudent s = new GoodStudent();
                s.eve();s.morn();s.night();

                Student s1 = new Student();
                s1.morn();
        }
}
```

```
com
 |-->dss
        |--->declarations
        |       |-->Message.class
        |-->helper
        |       |--->HelperAdaptor.class
        |-->bl
        |    |-->GoodStudent.class
        |  |-->Student.class
        |-->client
                |-->TestClient.class
```


Example :-

Demo.java

```
package a;
public interface Demo
{       public void sayHello(String msg);
}
```

ImplClass:-

```
package a;
class Test implements Demo
{       public void sayHello(String msg)//overriding method of Demo interface
        {       System.out.println("hi ratan--->"+msg);
        }
};
public class ImplClass
{       public Test objectcreation()//it returns Test class Object
        {       Test t = new Test();
                return t;
        }
}
```

Client.java

```
importa.ImplClass;
```

```
importa.Demo;
class Client
{       public static void main(String[] args)
        {       ImplClass i = new ImplClass();
                Demo d = i.objectcreation();
                 //it returns Object of class Test but we don't  know internally which object is created
                d.sayHello("hello");
        }
}
```


LEARN FROM EXPERTS ...
COMPLETE JAVA
CORE JAVA, ADV. JAVA, ORACLE, STRUTS, HIBERNATE, SPRING, WEB SERVICES,...
COMPLETE .NET
C#.NET, ASP.NET, SQL SERVER, MVC 5 & WCF
TESTING TOOLS
MANUAL + SELENIUM
ORACLE
D2K
MSBI
SHARE POINT
HADOOP
ANDROID
C, C++, DS, UNIX
CRT & APTITUDE TRAINING
AN ISO 9001:2008 CERTIFIED
DURGA
Software Solutions®
202, 2nd Floor, HUDA Maitrivanam,
Ameerpet, Hyd. Ph: 040-64512786,
9246212143, 8096969696
www.durgasoft.com