# **Interface**

Abstract class is used for partial abstraction. Interface is used for full abstraction.

Abstraction is a process where you show only relevant data and hide unnecessary details of an object from the user.

Interface looks like a class but it is not a class. An interface can have methods and variables just like the class but the methods declared in interface are by default abstract (only method signature, no body). Also, the variables declared in an interface are public, static & final by default.

Since methods in interfaces do not have body, they have to be implemented by the class before you can access them. The class that implements interface must implement all the methods of that interface. Also, java programming language does not allow you to extend more than one class, However, you can implement more than one interfaces in your class.

Syntax:

interface DemoInterface{

public void testmethod1();

public void testmethod2();

}

Example:

TestInterface Interface:

**package** FPPackage;

**public** **interface** TestInterface {

**public** **void** test1();

**public** **void** test2();

}

ClassForInterface Class:

**package** FPPackage;

**public** **class** ClassForInterface **implements** TestInterface{

**public** **void** test1() {

System.***out***.println("Implementing test1");

}

**public** **void** test2() {

System.***out***.println("Implementing test2");

}

**public** **static** **void** main(String[] args) {

TestInterface cl = **new** ClassForInterface();

cl.test1();

cl.test2();

}

}

An interface can not implement another interface. It has to extend another interface.

Example:

Interface TestInterface:

**package** FPPackage;

**public** **interface** TestInterface {

**public** **void** test1();

**public** **void** test2();

}

Interface TestInterface1:

**package** FPPackage;

**public** **interface** TestInterface1 **extends** TestInterface{

**public** **void** test3();

**public** **void** test4();

}

Class ClassForInterface:

**package** FPPackage;

**public** **class** ClassForInterface **implements** TestInterface1{

**public** **void** test1() {

System.***out***.println("Implementing test1");

}

**public** **void** test2() {

System.***out***.println("Implementing test2");

}

**public** **void** test3() {

System.***out***.println("Implementing test3");

}

**public** **void** test4() {

System.***out***.println("Implementing test4");

}

**public** **static** **void** main(String[] args) {

TestInterface cl = **new** ClassForInterface();

cl.test1();

cl.test2();

cl.test3();

cl.test4();

}

}

Even though the class is implementing only TestInterface1 it has to implement all the methods of the TestInterface also as TestInterface1 is extending TestInterface.

**Tag or Maker Interfaces:**

An empty interface is known as tag or Maker interface.

Ex: Serializable, EventListener, Remote

These interfaces do not have any field and methods in it. You must be thinking if they are empty why class implements them? What’s the use of it? Class implements them to claim the membership in a particular set. For example: If a class implements Serializable interface, it is claiming to be the member of Serializable classes, so if JVM (Java Virtual Machine) sees that a class is Serializable, it does some trick or special operation that helps in the serialization/de-serialization process.

Basically, Tag interfaces are meaningful to the JVM (Java virtual machine). You can also create your own tag interfaces to segregate and categorize your code. It would improve the readability of your code.

The following is how a tag interface looks like.

package java.util

public interface EventListener{

}

**Nested Interfaces:**

An interface which is declared inside another interface or class is called [nested](https://beginnersbook.com/2016/03/nested-or-inner-interfaces-in-java/) interface. They are also known as inner interface. For example, Entry interface in collections framework is declared inside Map interface, that’s why we don’ use it directly, rather we use it like this: Map.Entry

Key points:

1) We can’t instantiate an interface in java. That means we cannot create the object of an interface

2) Interface provides full abstraction as none of its methods have body. On the other hand abstract class provides partial abstraction as it can have abstract and concrete(methods with body) methods both.

3) implements keyword is used by classes to implement an interface.

4) While providing implementation in class of any method of an interface, it needs to be mentioned as public.

5) Class that implements any interface must implement all the methods of that interface, else the class should be declared abstract.

6) Interface cannot be declared as private, protected or transient.

7) All the interface methods are by default abstract and public.

8) Variables declared in interface are public, static and final by default.

interface Try

{

int a=10;

public int a=10;

public static final int a=10;

final int a=10;

static int a=0;

}

All of the above statements are identical.

9) Interface variables must be initialized at the time of declaration otherwise compiler will throw an error.

interface Try

{

int x;//Compile-time error

}

Above code will throw a compile time error as the value of the variable x is not initialized at the time of declaration.

10) Inside any implementation class, you cannot change the variables declared in interface because by default, they are public, static and final. Here we are implementing the interface “Try” which has a variable x. When we tried to set the value for variable x we got compilation error as the variable x is public static final by default and final variables can not be re-initialized.

class Sample implements Try

{

public static void main(String args[])

{

x=20; //compile time error

}

}

11) An interface can extend any interface but cannot implement it. Class implements interface and interface extends interface.

12) A class can implement any number of interfaces.

13) If there are two or more same methods in two interfaces and a class implements both interfaces, implementation of the method once is enough.

interface A

{

public void aaa();

}

interface B

{

public void aaa();

}

class Central implements A,B

{

public void aaa()

{

//Any Code here

}

public static void main(String args[])

{

//Statements

}

}

14) A class cannot implement two interfaces that have methods with same name but different return type.

interface A

{

public void aaa();

}

interface B

{

public int aaa();

}

class Central implements A,B

{

public void aaa() // error

{

}

public int aaa() // error

{

}

public static void main(String args[])

{

}

}

15) Variable names conflicts can be resolved by interface name.

interface A

{

int x=10;

}

interface B

{

int x=100;

}

class Hello implements A,B

{

public static void Main(String args[])

{

/\* reference to x is ambiguous both variables are x

\* so we are using interface name to resolve the

\* variable

\*/

System.out.println(x);

System.out.println(A.x);

System.out.println(B.x);

}

}