Interface in Java

An **interface in java** is a blueprint of a class. It has static constants and abstract methods.

The interface in Java is *a mechanism to achieve abstraction*. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.

In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.

Java Interface also **represents the IS-A relationship**.

It cannot be instantiated just like the abstract class.

Since Java 8, we can have **default and static methods** in an interface.

Since Java 9, we can have **private methods** in an interface.

Why use Java interface?

There are mainly three reasons to use interface. They are given below.

* It is used to achieve abstraction.
* By interface, we can support the functionality of multiple inheritance.
* It can be used to achieve loose coupling.



How to declare an interface?

An interface is declared by using the interface keyword. It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default. A class that implements an interface must implement all the methods declared in the interface.

Syntax:

1. **interface** <interface\_name>{
3. // declare constant fields
4. // declare methods that abstract
5. // by default.
6. }

Java 8 Interface Improvement

Since Java 8, interface can have default and static methods which is discussed later.

Internal addition by the compiler

The Java compiler adds public and abstract keywords before the interface method. Moreover, it adds public, static and final keywords before data members.

In other words, Interface fields are public, static and final by default, and the methods are public and abstract.



The relationship between classes and interfaces

As shown in the figure given below, a class extends another class, an interface extends another interface, but a **class implements an interface**.



Java Interface Example

In this example, the Printable interface has only one method, and its implementation is provided in the A6 class.

1. **interface** printable{
2. **void** print();
3. }
4. **class** A6 **implements** printable{
5. **public** **void** print(){System.out.println("Hello");}
7. **public** **static** **void** main(String args[]){
8. A6 obj = **new** A6();
9. obj.print();
10. }
11. }

**[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=A6" \t "_blank)**

Output:

Hello

Java Interface Example: Drawable

In this example, the Drawable interface has only one method. Its implementation is provided by Rectangle and Circle classes. In a real scenario, an interface is defined by someone else, but its implementation is provided by different implementation providers. Moreover, it is used by someone else. The implementation part is hidden by the user who uses the interface.

*File: TestInterface1.java*

1. //Interface declaration: by first user
2. **interface** Drawable{
3. **void** draw();
4. }
5. //Implementation: by second user
6. **class** Rectangle **implements** Drawable{
7. **public** **void** draw(){System.out.println("drawing rectangle");}
8. }
9. **class** Circle **implements** Drawable{
10. **public** **void** draw(){System.out.println("drawing circle");}
11. }
12. //Using interface: by third user
13. **class** TestInterface1{
14. **public** **static** **void** main(String args[]){
15. Drawable d=**new** Circle();//In real scenario, object is provided by method e.g. getDrawable()
16. d.draw();
17. }}

**[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=TestInterface1" \t "_blank)**

Output:

drawing circle

Java Interface Example: Bank

Let's see another example of java interface which provides the implementation of Bank interface.

*File: TestInterface2.java*

1. **interface** Bank{
2. **float** rateOfInterest();
3. }
4. **class** SBI **implements** Bank{
5. **public** **float** rateOfInterest(){**return** 9.15f;}
6. }
7. **class** PNB **implements** Bank{
8. **public** **float** rateOfInterest(){**return** 9.7f;}
9. }
10. **class** TestInterface2{
11. **public** **static** **void** main(String[] args){
12. Bank b=**new** SBI();
13. System.out.println("ROI: "+b.rateOfInterest());
14. }}

**[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=TestInterface2" \t "_blank)**

Output:

ROI: 9.15

Multiple inheritance in Java by interface

If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as multiple inheritance.



1. **interface** Printable{
2. **void** print();
3. }
4. **interface** Showable{
5. **void** show();
6. }
7. **class** A7 **implements** Printable,Showable{
8. **public** **void** print(){System.out.println("Hello");}
9. **public** **void** show(){System.out.println("Welcome");}
11. **public** **static** **void** main(String args[]){
12. A7 obj = **new** A7();
13. obj.print();
14. obj.show();
15. }
16. }

**[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=A7" \t "_blank)**

Output:Hello

Welcome

Q) Multiple inheritance is not supported through class in java, but it is possible by an interface, why?

As we have explained in the inheritance chapter, multiple inheritance is not supported in the case of class because of ambiguity. However, it is supported in case of an interface because there is no ambiguity. It is because its implementation is provided by the implementation class. For example:

1. **interface** Printable{
2. **void** print();
3. }
4. **interface** Showable{
5. **void** print();
6. }
8. **class** TestInterface3 **implements** Printable, Showable{
9. **public** **void** print(){System.out.println("Hello");}
10. **public** **static** **void** main(String args[]){
11. TestInterface3 obj = **new** TestInterface3();
12. obj.print();
13. }
14. }

**[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=TestInterface3" \t "_blank)**

Output:

Hello

As you can see in the above example, Printable and Showable interface have same methods but its implementation is provided by class TestTnterface1, so there is no ambiguity.

Interface inheritance

A class implements an interface, but one interface extends another interface.

1. **interface** Printable{
2. **void** print();
3. }
4. **interface** Showable **extends** Printable{
5. **void** show();
6. }
7. **class** TestInterface4 **implements** Showable{
8. **public** **void** print(){System.out.println("Hello");}
9. **public** **void** show(){System.out.println("Welcome");}
11. **public** **static** **void** main(String args[]){
12. TestInterface4 obj = **new** TestInterface4();
13. obj.print();
14. obj.show();
15. }
16. }

**[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=TestInterface4" \t "_blank)**

Output:

Hello

Welcome

Java 8 Default Method in Interface

Since Java 8, we can have method body in interface. But we need to make it default method. Let's see an example:

*File: TestInterfaceDefault.java*

1. **interface** Drawable{
2. **void** draw();
3. **default** **void** msg(){System.out.println("default method");}
4. }
5. **class** Rectangle **implements** Drawable{
6. **public** **void** draw(){System.out.println("drawing rectangle");}
7. }
8. **class** TestInterfaceDefault{
9. **public** **static** **void** main(String args[]){
10. Drawable d=**new** Rectangle();
11. d.draw();
12. d.msg();
13. }}

**[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=TestInterfaceDefault" \t "_blank)**

Output:

drawing rectangle

default method

Java 8 Static Method in Interface

Since Java 8, we can have static method in interface. Let's see an example:

*File: TestInterfaceStatic.java*

1. **interface** Drawable{
2. **void** draw();
3. **static** **int** cube(**int** x){**return** x\*x\*x;}
4. }
5. **class** Rectangle **implements** Drawable{
6. **public** **void** draw(){System.out.println("drawing rectangle");}
7. }
9. **class** TestInterfaceStatic{
10. **public** **static** **void** main(String args[]){
11. Drawable d=**new** Rectangle();
12. d.draw();
13. System.out.println(Drawable.cube(3));
14. }}

**[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=TestInterfaceStatic" \t "_blank)**

Output:

drawing rectangle

27

Q) What is marker or tagged interface?

An interface which has no member is known as a marker or tagged interface, for example, Serializable, Cloneable, Remote, etc. They are used to provide some essential information to the JVM so that JVM may perform some useful operation.

1. //How Serializable interface is written?
2. **public** **interface** Serializable{
3. }

Nested Interface in Java

Note: An interface can have another interface which is known as a nested interface. We will learn it in detail in the nested classes chapter. For example:

1. **interface** printable{
2. **void** print();
3. **interface** MessagePrintable{
4. **void** msg();
5. }
6. }

Java Nested Interface

An interface i.e. declared within another interface or class is known as nested interface. The nested interfaces are used to group related interfaces so that they can be easy to maintain. The nested interface must be referred by the outer interface or class. It can't be accessed directly.

Points to remember for nested interfaces

There are given some points that should be remembered by the java programmer.

* Nested interface must be public if it is declared inside the interface but it can have any access modifier if declared within the class.
* Nested interfaces are declared static implicitely.

Syntax of nested interface which is declared within the interface

1. **interface** interface\_name{
2. ...
3. **interface** nested\_interface\_name{
4. ...
5. }
6. }

Syntax of nested interface which is declared within the class

1. **class** class\_name{
2. ...
3. **interface** nested\_interface\_name{
4. ...
5. }
6. }

Example of nested interface which is declared within the interface

|  |
| --- |
| In this example, we are going to learn how to declare the nested interface and how we can access it. |

1. **interface** Showable{
2. **void** show();
3. **interface** Message{
4. **void** msg();
5. }
6. }
7. **class** TestNestedInterface1 **implements** Showable.Message{
8. **public** **void** msg(){System.out.println("Hello nested interface");}
10. **public** **static** **void** main(String args[]){
11. Showable.Message message=**new** TestNestedInterface1();//upcasting here
12. message.msg();
13. }
14. }

**[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=TestNestedInterface1" \t "_blank)**

[download the example of nested interface](https://static.javatpoint.com/src/nested/nestedinterface.zip)

Output:hello nested interface

|  |
| --- |
| As you can see in the above example, we are acessing the Message interface by its outer interface Showable because it cannot be accessed directly. It is just like almirah inside the room, we cannot access the almirah directly because we must enter the room first. In collection frameword, sun microsystem has provided a nested interface Entry. Entry is the subinterface of Map i.e. accessed by Map.Entry. |

Internal code generated by the java compiler for nested interface Message

|  |
| --- |
| The java compiler internally creates public and static interface as displayed below:. |

1. **public** **static** **interface** Showable$Message
2. {
3. **public** **abstract** **void** msg();
4. }

Example of nested interface which is declared within the class

|  |
| --- |
| Let's see how can we define an interface inside the class and how can we access it. |

1. **class** A{
2. **interface** Message{
3. **void** msg();
4. }
5. }
7. **class** TestNestedInterface2 **implements** A.Message{
8. **public** **void** msg(){System.out.println("Hello nested interface");}
10. **public** **static** **void** main(String args[]){
11. A.Message message=**new** TestNestedInterface2();//upcasting here
12. message.msg();
13. }
14. }

**[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=TestNestedInterface2" \t "_blank)**

Output:hello nested interface

Can we define a class inside the interface?

Yes, If we define a class inside the interface, java compiler creates a static nested class. Let's see how can we define a class within the interface:

1. **interface** M{
2. **class** A{}
3. }