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.

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

It cannot be instantiated just like abstract class.

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.

Internal addition by compiler

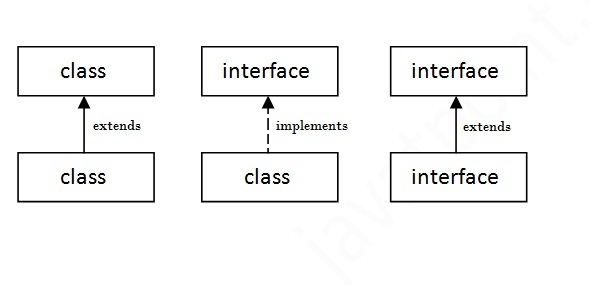
**The java compiler adds public and abstract keywords before the interface method. More, it adds public, static and final keywords before data members.**

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

interface in java

Understanding 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, Printable interface has only one method, its implementation is provided in the A 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. }

Output:

Hello

Java Interface Example: Drawable

In this example, Drawable interface has only one method. Its implementation is provided by Rectangle and Circle classes. In real scenario, interface is defined by someone but implementation is provided by different implementation providers. And, it is used by someone else. The implementation part is hidden by the user which 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. }}

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*

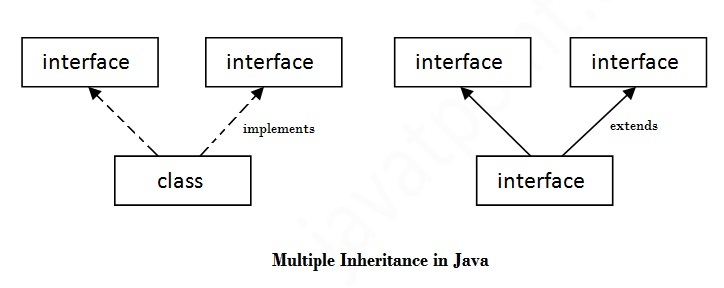
1. **interface** Bank{
2. **float** rateOfInterest();
3. }
4. **class** SBI **implements** Bank{
5. **public** **float** rateOfInterest(){**return** 9.15f;}
6. }
7. **class** PNB **implements** Bank{
8. **public** **float** rateOfInterest(){**return** 9.7f;}
9. }
10. **class** TestInterface2{
11. **public**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

Output:

ROI: 9.15

Multiple inheritance in Java by interface

If a class implements multiple interfaces, or an interface extends multiple interfaces i.e. 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. }

Output:Hello

Welcome

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

As we have explained in the inheritance chapter, multiple inheritance is not supported in case of class because of ambiguity. But it is supported in case of interface because there is no ambiguity as 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** TestTnterface3 **implements** Printable, Showable{
9. **public** **void** print()
10. {
11. System.out.println("Hello");
12. }
13. **public** **static** **void** main(String args[]){
14. TestTnterface1 obj = **new** TestTnterface1();
15. obj.print();
16. }
17. }

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 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. }

Output:

Hello

Welcome

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. }}

Output:

drawing rectangle

27