Abstraction

# Abstract VS Interface

**Abstraction** is a process of hiding the implementation details and showing only functionality to the user.

Another way, it shows only essential things to the user and hides the internal details, for example, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery.

Abstraction lets you focus on what the object does instead of how it does it.

#### **Points to Remember**

* An abstract class must be declared with an abstract keyword.
* It can have abstract and non-abstract methods.
* It cannot be instantiated.
* It can have constructors and static methods also.
* It can have final methods which will force the subclass not to change the body of the method.

1. **abstract** **class** Shape{
2. **abstract** **void** draw();
3. **void display(){ //if this method is not abstract that mean class have to define body for display function().**
4. }
5. }
6. //In real scenario, implementation is provided by others i.e. unknown by end user
7. **class** Rectangle **extends** Shape{
8. **void** draw(){System.out.println("drawing rectangle");}
9. }
10. **class** Circle1 **extends** Shape{
11. **void** draw(){System.out.println("drawing circle");}
12. }
13. //In real scenario, method is called by programmer or user
14. **class** TestAbstraction1{
15. **public** **static** **void** main(String args[]){
16. Shape s=**new** Circle1();//In a real scenario, object is provided through method, e.g., getShape() method
17. s.draw();
18. }  }
19. **interface** A{
20. **void** a();
21. **void** b();
22. **void** c();
23. **void** d();
24. }
26. **abstract** **class** B **implements** A{
27. **public** **void** c(){System.out.println("I am c");}
28. }
30. **class** M **extends** B{
31. **public** **void** a(){System.out.println("I am a");}
32. **public** **void** b(){System.out.println("I am b");}
33. **public** **void** d(){System.out.println("I am d");}
34. }
36. **class** Test5{
37. **public** **static** **void** main(String args[]){
38. A a=**new** M();
39. a.a();
40. a.b();
41. a.c();
42. a.d();
43. }}

Abstract class which implements interface now abstract class have to give body to all declare in interface.

Becoz by default all method declare in interface is abstract.

# **Interface in Java**

An **interface in java** is a blueprint of a class. It has **static constants** and **abstract methods.** An interface is a description of a set of methods that conforming implementing classes must have. An interface may never contain method definitions.

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**.

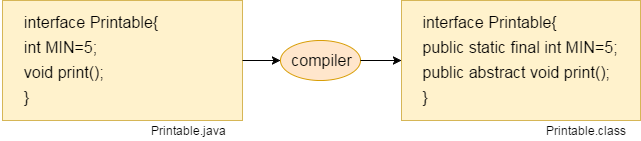
**It cannot be instantiated just like the abstract class.**

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

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. }

## **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. }

## **Java 8 Default aND Static Method**

## **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. **static** **int** cube(**int** x){**return** x\*x\*x;}
4. **default** **void** msg(){System.out.println("default method");}
5. }
6. **class** Rectangle **implements** Drawable{
7. **public** **void** draw(){System.out.println("drawing rectangle");}
8. }
9. **class** TestInterfaceDefault{
10. **public** **static** **void** main(String args[]){
11. Drawable d=**new** Rectangle();
12. d.draw();
13. d.cube();
14. d.msg();
15. }}

If a class which implements interface does not want to give body to fun which is declare in interface that function can be made default.

## **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. }

# **Java Nested Interface**

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.

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. }

### **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. }

### **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. }

|  |  |
| --- | --- |
| **Abstract class** | **Interface** |
| 1) Abstract class can **have abstract and non-abstract** methods. | Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also. |
| 2) Abstract class **doesn't support multiple inheritance**. | Interface **supports multiple inheritance**. |
| 3) Abstract class **can have final, non-final, static and non-static variables**. | Interface has **only static and final variables**. |
| 4) Abstract class **can provide the implementation of interface**. | Interface **can't provide the implementation of abstract class**. |
| 5) The **abstract keyword** is used to declare abstract class. | The **interface keyword** is used to declare interface. |
| 6) An **abstract class** can extend another Java class and implement multiple Java interfaces. | An **interface** can extend another Java interface only. |
| 7) An **abstract class** can be extended using keyword "extends". | An **interface class** can be implemented using keyword "implements". |
| 8) A Java **abstract class** can have class members like private, protected, etc. | Members of a Java interface are public by default. |
| 9)**Example:** public abstract class Shape{ public abstract void draw(); } | **Example:** public interface Drawable{ void draw(); } |