18. 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.

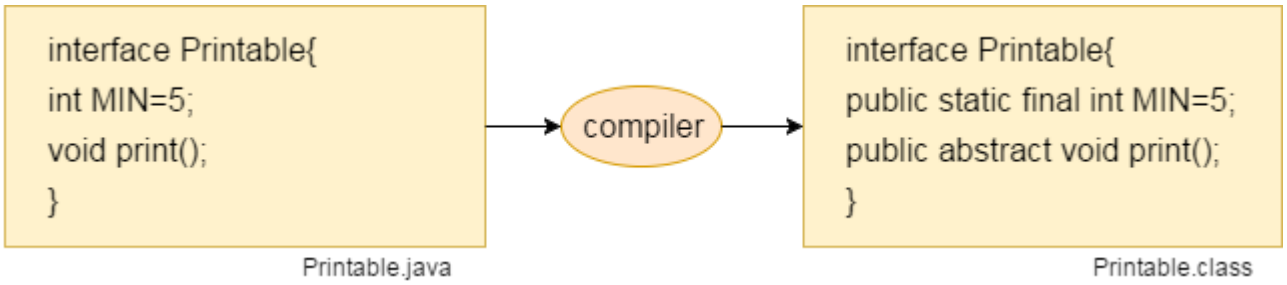
Java 8 Interface Improvement

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

- **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.



Access specifier of methods in interfaces

In Java, all methods in an interface are *public* even if we do not specify *public* with method names. Also, data fields are *public static final* even if we do not mention it with fields names. Therefore, data fields must be initialized.

Consider the following example, *x* is by default *public static final* and *foo()* is *public* even if there are no specifiers.

```
interface Test {
  int x = 10;  // x is public static final and must be initialized here
  void foo();  // foo() is public
}
```

Access specifiers for classes or interfaces in Java

In Java, methods and data members of a class/interface can have one of the following four access specifiers. The access specifiers are listed according to their restrictiveness order.
1) private
2) default (when no access specifier is specified)
3) protected
4) public
But, the classes and interfaces themselves can have only two access specifiers when declared outside any other class.
1) public
2) default (when no access specifier is specified)
We cannot declare class/interface with private or protected access specifiers. For example, following program fails in compilation.

```
//filename: Main.java
protected class Test {}

public class Main {
  public static void main(String args[]) {

  }
}
```

Note : Nested interfaces and classes can have all access specifiers.

Nested Interface in Java

We can declare interfaces as member of a class or another interface. Such an interface is called as member interface or nested interface.

Interface in a class

Interfaces (or classes) can have only public and default access specifiers when declared outside any other class .This interface declared in a class can either be default, public, protected not private. While implementing the interface, we mention the interface

as **c_name.i_name** where **c_name** is the name of the class in which it is nested and **i_name** is the name of the interface itself.
Let us have a look at the following code:-

```
// Java program to demonstrate working of

// interface inside a class.

import java.util.*;

class Test

{

   interface Yes

   {

      void show();

   }

}


class Testing implements Test.Yes

{

   public void show()

   {

      System.out.println("show method of interface");

   }

}


class A

{

   public static void main(String[] args)

   {

      Test.Yes obj;

      Testing t = new Testing();

      obj=t;

      obj.show();

   }
```

```
}
```

Output:

```
show method of interface
```

The access specifier in above example is default. We can assign public, protected or private also.

- Below is an example of protected. In this particular example, if we change access specifier to private, we get compiler error because a derived class tries to access it.

```
// Java program to demonstrate protected
// specifier for nested interface.
import java.util.*;
class Test
{
   protected interface Yes
   {
      void show();
   }
}

class Testing implements Test.Yes
{
   public void show()
   {
      System.out.println("show method of interface");
   }
}
 class A
{
   public static void main(String[] args)
   {
      Test.Yes obj;
```

```
        Testing t = new Testing();
        obj=t;
        obj.show();
    }
}
```

Output:

```
show method of interface
```

Interface in another Interface

An interface can be declared inside another interface also. We mention the interface as **i_name1.i_name2** where **i_name1** is the name of the interface in which it is nested and **i_name2** is the name of the interface to be implemented.

```
// Java program to demonstrate working of
// interface inside another interface.

import java.util.*;

interface Test
{
   interface Yes
   {
      void show();
   }
}
 class Testing implements Test.Yes
{
   public void show()
   {
      System.out.println("show method of interface");
   }
}
```

```
class A
{
  public static void main(String[] args)
  {
    Test.Yes obj;
    Testing t = new Testing();
    obj = t;
    obj.show();
  }
}
```

Output:

```
show method of interface
```

Note: In the above example, access specifier is public even if we have not written public. If we try to change access specifier of interface to anything other than public, we get compiler error. Remember, interface members can only be public..

```
// Java program to demonstrate an interface cannot
// have non-public member interface.
import java.util.*;
interface Test
{
   protected interface Yes
   {
      void show();
   }
}
 class Testing implements Test.Yes
{
   public void show()
   {
```

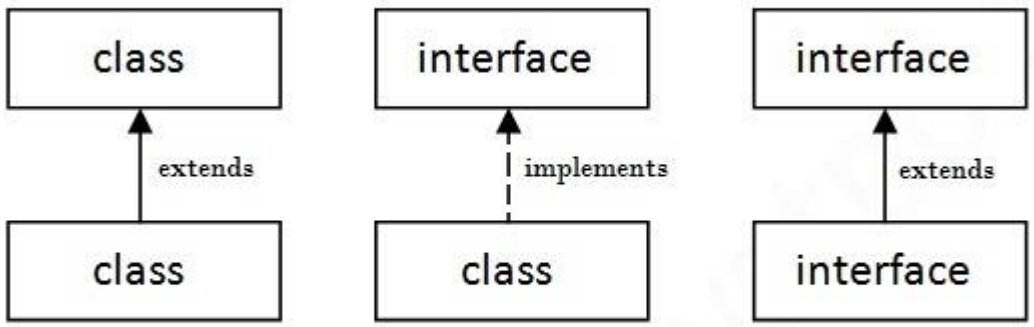
```
        System.out.println("show method of interface");
    }
}
 class A
{
    public static void main(String[] args)
    {
        Test.Yes obj;
        Testing t = new Testing();
        obj = t;
        obj.show();
    }
}
```

Output:

```
illegal combination of modifiers: public and protected
   protected interface Yes
```

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");}
6.
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

```
//Interface declaration: by first user
1. interface Drawable{
2. void draw();
3. }
4. //Implementation: by second user
5. class Rectangle implements Drawable{
6. public void draw(){System.out.println("drawing rectangle");}
7. }
8. class Circle implements Drawable{
9. public void draw(){System.out.println("drawing circle");}
10. }
```

```
11. //Using interface: by third user
12. class TestInterface1{
13. public static void main(String args[]){
14. Drawable d=new Circle();//In real scenario, object is provided by method e.g. getDrawable()
15. d.draw();
16. }}
```

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

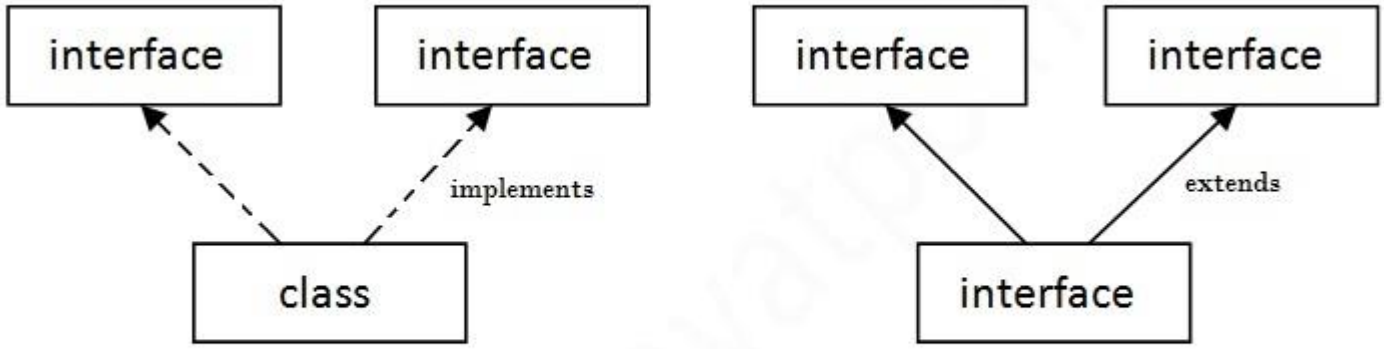
```
1. interface Bank{
2. float rateOfInterest();
3. }
4. class SBI implements Bank{
5. public float rateOfInterest(){return 9.15f;}
6. }
7. class PNB implements Bank{
8. public float rateOfInterest(){return 9.7f;}
9. }
10. class TestInterface2{
11. public static void main(String[] args){
12. Bank b=new 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.



Multiple Inheritance in Java

```
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");}
10.
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. }
7.
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. }
```

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");}
10.
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 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. }}
```

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. }
8.
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
```

Difference between abstract class and interface

Abstract class and interface both are used to achieve abstraction where we can declare the abstract methods. Abstract class and interface both can't be instantiated.

But there are many differences between abstract class and interface that are given below.

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) **Example:**
public abstract class Shape{
public abstract void draw();
}
	Example:
	public interface Drawable{
	void draw();
	}

Simply, abstract class achieves partial abstraction (0 to 100%) whereas interface achieves fully abstraction (100%).

Example of abstract class and interface in Java

Let's see a simple example where we are using interface and abstract class both.

```
//Creating interface that has 4 methods
interface A{
void a();//bydefault, public and abstract
void b();
void c();
void d();
}

//Creating abstract class that provides the implementation of one method of A interface
abstract class B implements A{
public void c(){System.out.println("I am C");}
}

//Creating subclass of abstract class, now we need to provide the implementation of rest of the me
thods
class M extends B{
public void a(){System.out.println("I am a");}
public void b(){System.out.println("I am b");}
public void d(){System.out.println("I am d");}
}

//Creating a test class that calls the methods of A interface
class Test5{
public static void main(String args[]){
A a=new M();
a.a();
a.b();
a.c();
a.d();
}}
```

Output:

```
I am a
I am b
I am c
I am d
```