Interface

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.

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.

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?

```
interface <interface_name>
{

   // declare constant fields
   // declare methods that abstract
   // by default.
}
```

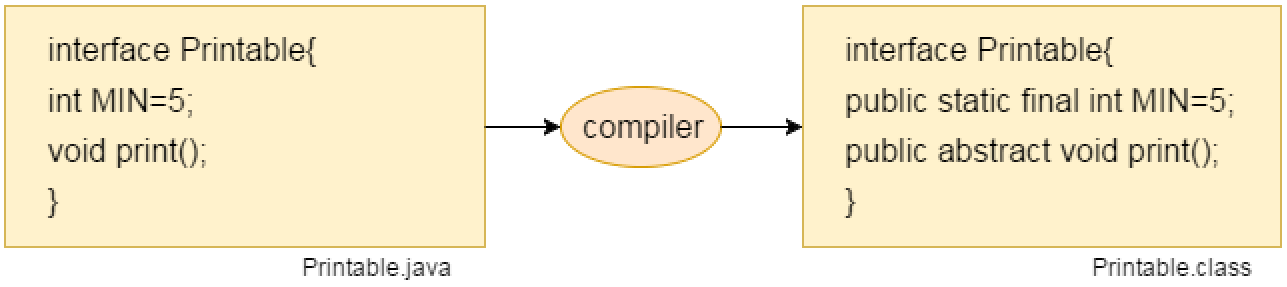
Java 8 Interface Improvement

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

Internal addition by the compiler

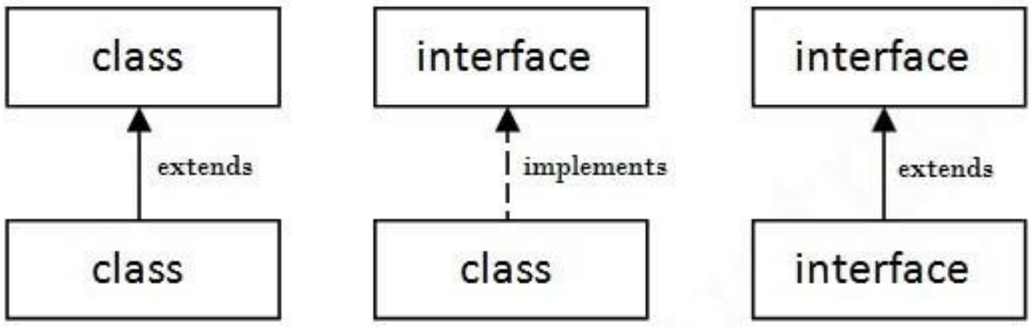
The Java compiler adds public and abstract keywords before the interface method. Moreover, it adds public, static and final keywords before data members.

In other words,



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**.



Interface Example

```
interface printable
{

void print();

}


class A implements printable
{

	public void print()
	{

		System.out.println("Hello");

	}


}


public class Idemo {

	public static void main(String[] args) {
		// TODO Auto-generated method stub

		A a1= new A();
		a1.print();

	}

}
```

Example 2

```
interface Printable
{
void print1();
}

interface Showable
{
void print2();
}

class TestInterface implements Printable, Showable
{
```

```
public void print1()
{
        System.out.println("Hello");
}
public void print2()
{
        System.out.println("Bye");
}

Class M
{

public static void main(String args[])
{

TestInterface obj = new TestInterface();
obj.print1();
obj.print2();


}

}
```

Example 3

```
interface Printable
{
void print();
}
interface Showable extends Printable
{
void show();
}
class TestInterface implements Showable
{
public void print()
{
System.out.println("Hello");

}
public void show()
{
        System.out.println("Welcome");
}

public static void main(String args[])
{
TestInterface obj = new TestInterface();
obj.print();
obj.show();

}
}
```

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:

```
interface Drawable
{
    void draw();
    default void msg()
    {
        System.out.println("default method");
    }
}
class Rectangle implements Drawable
{
    public void draw()
    {
        System.out.println("drawing rectangle");
    }
}


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

        Drawable d=new Rectangle();
        d.draw();
        d.msg();
    }
}
```

Java 8 Static Method in Interface

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

```
interface Drawable
{
    void draw();
    static int cube(int x)
    {
        return x*x*x;
    }
}

class Rectangle implements Drawable
{
    public void draw()
    {
        System.out.println("drawing rectangle");
    }
}
```

```java
public class Test {
    public static void main(String args[])
    {

        Drawable d=new Rectangle();
        d.draw();
        System.out.println(Drawable.cube(3));
    }
}
```