

Abstract class in Java

A class which is declared with the abstract keyword is known as an abstract class in **Java**. It can have abstract and non-abstract methods (method with the body).

Before learning the Java abstract class, let's understand the abstraction in Java first.

Abstraction in Java

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.

Ways to achieve Abstraction

There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%)
2. Interface (100%)

Abstract class in Java

A class which is declared as abstract is known as an **abstract class**. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.

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.

Rules for Java Abstract class

**1**

An abstract class must be declared with an abstract keyword.

2

It can have abstract and non-abstract methods.

3

It cannot be instantiated.

4

It can have final methods

5

It can have constructors and static methods also.

Example of abstract class

```
abstract class A{
```

Abstract Method in Java

A method which is declared as abstract and does not have implementation is known as an abstract method.

Example of abstract method

```
abstract void printStatus();//no method body and abstract
```

Example of Abstract class that has an abstract method

In this example, Bike is an abstract class that contains only one abstract method run. Its implementation is provided by the Honda class.

```
abstract class Bike{
    abstract void run();
}
class Honda4 extends Bike{
    void run(){System.out.println("running safely");}
    public static void main(String args[]){
        Bike obj = new Honda4();
        obj.run();
    }
}
```

Test it Now

```
running safely
```

Understanding the real scenario of Abstract class

In this example, Shape is the abstract class, and its implementation is provided by the Rectangle and Circle classes.

Mostly, we don't know about the implementation class (which is hidden to the end user), and an object of the implementation class is provided by the **factory method**.

A **factory method** is a method that returns the instance of the class. We will learn about the factory method later.

In this example, if you create the instance of Rectangle class, draw() method of Rectangle class will be invoked.

File: TestAbstraction1.java

```
abstract class Shape{
    abstract void draw();
}
//In real scenario, implementation is provided by others i.e. unknown by end user
class Rectangle extends Shape{
    void draw(){System.out.println("drawing rectangle");}
}
class Circle1 extends Shape{
```

```

void draw(){System.out.println("drawing circle");}

}

//In real scenario, method is called by programmer or user

class TestAbstraction1{
public static void main(String args[]){
Shape s=new Circle1();//In a real scenario, object is provided through method, e.g., getShape() met
s.draw();
}
}

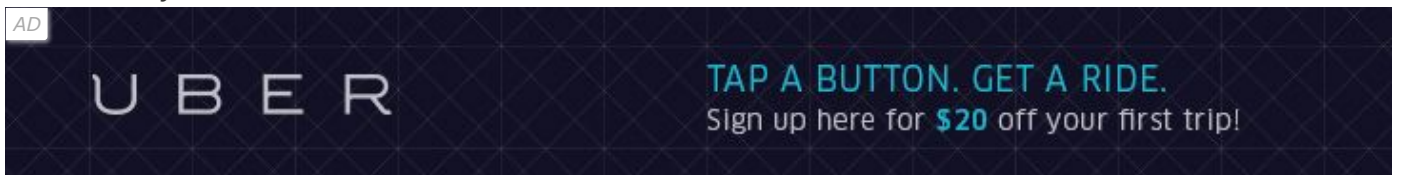
```

Test it Now

drawing circle

Another example of Abstract class in java

File: TestBank.java



```

abstract class Bank{
abstract int getRateOfInterest();
}

class SBI extends Bank{
int getRateOfInterest(){return 7;}
}

class PNB extends Bank{
int getRateOfInterest(){return 8;}
}

class TestBank{
public static void main(String args[]){
Bank b;
b=new SBI();
System.out.println("Rate of Interest is: " +b.getRateOfInterest()+" %");
b=new PNB();
}
}

```

```
System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");  
}}
```

Test it Now

```
Rate of Interest is: 7 %  
Rate of Interest is: 8 %
```

Abstract class having constructor, data member and methods

An abstract class can have a data member, abstract method, method body (non-abstract method), constructor, and even main() method.

File: TestAbstraction2.java

```
//Example of an abstract class that has abstract and non-abstract methods  
abstract class Bike{  
    Bike(){System.out.println("bike is created");}  
    abstract void run();  
    void changeGear(){System.out.println("gear changed");}  
}  
  
//Creating a Child class which inherits Abstract class  
class Honda extends Bike{  
    void run(){System.out.println("running safely..");}  
}  
  
//Creating a Test class which calls abstract and non-abstract methods  
class TestAbstraction2{  
    public static void main(String args[]){  
        Bike obj = new Honda();  
        obj.run();  
        obj.changeGear();  
    }  
}
```

Test it Now

```
bike is created  
running safely..
```

```
gear changed
```

Rule: If there is an abstract method in a class, that class must be abstract.

```
class Bike12{  
    abstract void run();  
}
```

Test it Now

```
compile time error
```

Rule: If you are extending an abstract class that has an abstract method, you must either provide the implementation of the method or make this class abstract.

Another real scenario of abstract class

The abstract class can also be used to provide some implementation of the **interface**. In such case, the end user may not be forced to override all the methods of the interface.

Note: If you are beginner to java, learn interface first and skip this example.

```
interface A{  
    void a();  
    void b();  
    void c();  
    void d();  
}  
  
abstract class B implements A{  
    public void c(){System.out.println("I am c");}  
}
```

```
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");}  
}
```

```
class Test5{  
    public static void main(String args[]){  
        A a=new M();  
        a.a();  
        a.b();  
        a.c();  
        a.d();  
    }  
}
```

Test it Now

```
Output:I am a  
       I am b  
       I am c  
       I am d
```

[< Prev](#)[Next >](#)

AD

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?

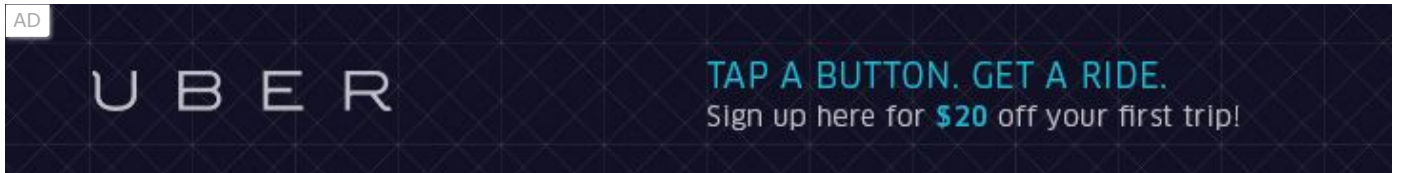
An interface is declared by using the interface keyword. It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default. A class that implements an interface must implement all the methods declared in the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

Syntax:

```
interface <interface_name>{  
  
    // declare constant fields  
    // declare methods that abstract  
    // by default.  
}
```

Java 8 Interface Improvement

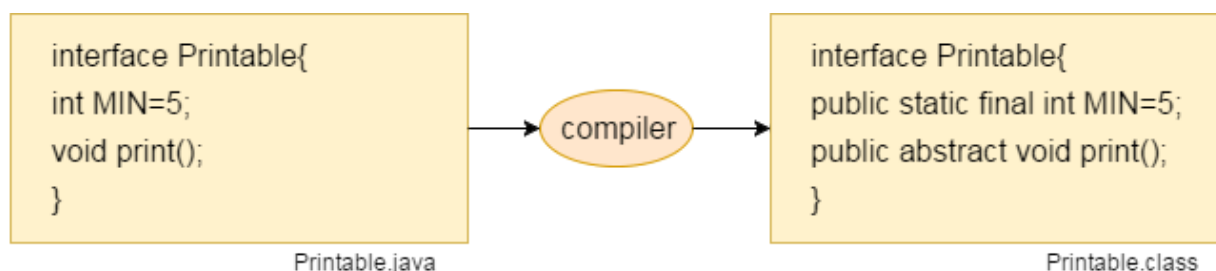
Since Java 8, interface can have default and static methods which is discussed later.



Internal addition by the compiler

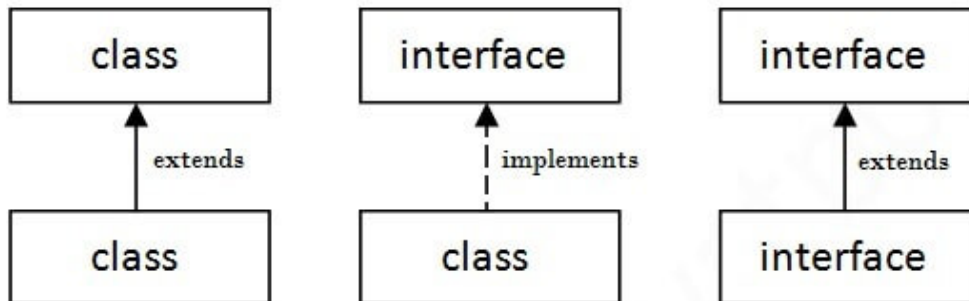
The Java compiler adds public and abstract keywords before the interface method. Moreover, it adds public, static and final keywords before data members.

In other words, 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.

```
interface printable{  
    void print();  
}  
  
class A6 implements printable{  
    public void print(){System.out.println("Hello");}  
  
    public static void main(String args[]){  
        A6 obj = new A6();  
        obj.print();  
    }  
}
```

Test it Now

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

```
//Interface declaration: by first user
interface Drawable{
    void draw();
}

//Implementation: by second user
class Rectangle implements Drawable{
    public void draw(){System.out.println("drawing rectangle");}
}
class Circle implements Drawable{
    public void draw(){System.out.println("drawing circle");}
}

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

Test it Now

Output:

AD



VIDEO STREAMING FOR THE AWAKENED MIND

GET STARTED NOW

```
drawing circle
```

Java Interface Example: Bank

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

File: TestInterface2.java

```
interface Bank{  
    float rateOfInterest();  
}  
  
class SBI implements Bank{  
    public float rateOfInterest(){return 9.15f;}  
}  
  
class PNB implements Bank{  
    public float rateOfInterest(){return 9.7f;}  
}  
  
class TestInterface2{  
    public static void main(String[] args){  
        Bank b=new SBI();  
        System.out.println("ROI: "+b.rateOfInterest());  
    }  
}
```

Test it Now

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.



Multiple Inheritance in Java

```
interface Printable{  
    void print();  
}  
  
interface Showable{  
    void show();  
}  
  
class A7 implements Printable,Showable{  
    public void print(){System.out.println("Hello");}  
    public void show(){System.out.println("Welcome");}  
  
    public static void main(String args[]){  
        A7 obj = new A7();  
        obj.print();  
        obj.show();  
    }  
}
```

Test it Now

```
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:



```
interface Printable{  
    void print();  
}  
  
interface Showable{  
    void print();  
}
```

```
}

class TestInterface3 implements Printable, Showable{
    public void print(){System.out.println("Hello");}
    public static void main(String args[]){
        TestInterface3 obj = new TestInterface3();
        obj.print();
    }
}
```

Test it Now

Output:

```
Hello
```

As you can see in the above example, Printable and Showable interface have same methods but its implementation is provided by class TestInterface1, so there is no ambiguity.

Interface inheritance

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

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

interface Showable extends Printable{
    void show();
}

class TestInterface4 implements Showable{
    public void print(){System.out.println("Hello");}
    public void show(){System.out.println("Welcome");}

    public static void main(String args[]){
        TestInterface4 obj = new TestInterface4();
        obj.print();
        obj.show();
    }
}
```

```
}
```

Test it Now

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

```
interface Drawable{  
    void draw();  
    default void msg(){System.out.println("default method");}  
}  
class Rectangle implements Drawable{  
    public void draw(){System.out.println("drawing rectangle");}  
}  
class TestInterfaceDefault{  
    public static void main(String args[]){  
        Drawable d=new Rectangle();  
        d.draw();  
        d.msg();  
    }  
}
```

Test it Now

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*

```
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");}  
}  
  
class TestInterfaceStatic{  
    public static void main(String args[]){  
        Drawable d=new Rectangle();  
        d.draw();  
        System.out.println(Drawable.cube(3));  
    }  
}
```

Test it Now

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.

```
//How Serializable interface is written?  
public interface Serializable{  
}
```


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:

```
interface printable{  
    void print();  
    interface MessagePrintable{  
        void msg();  
    }  
}
```

[More about Nested Interface](#)

[« Prev](#)[Next »](#)

AD

 [For Videos Join Our Youtube Channel: Join Now](#)

Feedback

- Send your Feedback to feedback@javatpoint.com

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) 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 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: <pre>public abstract class Shape{ public abstract void draw(); }</pre>	Example: <pre>public interface Drawable{ void draw(); }</pre>

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
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();
}}
```

Test it Now

Output:

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

[< Prev](#)[Next >](#)

AD

 [For Videos Join Our Youtube Channel: Join Now](#)


Feedback


- Send your Feedback to feedback@javatpoint.com

Help Others, Please Share




Learn Latest Tutorials

 [Splunk tutorial](#)
Splunk

 [SPSS tutorial](#)
SPSS

 [Swagger tutorial](#)
Swagger

 [T-SQL tutorial](#)
Transact-SQL