|  |
| --- |
| **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.   **Example of abstract class**   1. 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**   1. 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](http://www.javatpoint.com/opr/test.jsp?filename=Honda4)  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() method  s.draw();  }  }  [Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=TestAbstraction1)  drawing circle  **Another example of Abstract class in java**  File: TestBank.java   1. abstract class Bank{ 2. abstract int getRateOfInterest(); 3. } 4. class SBI extends Bank{ 5. int getRateOfInterest(){return 7;} 6. } 7. class PNB extends Bank{ 8. int getRateOfInterest(){return 8;} 9. } 11. class TestBank{ 12. public static void main(String args[]){ 13. Bank b; 14. b=new SBI(); 15. System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %"); 16. b=new PNB(); 17. System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %"); 18. }}   [Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=TestBank)  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   1. //Example of an abstract class that has abstract and non-abstract methods 2. abstract class Bike{ 3. Bike(){System.out.println("bike is created");} 4. abstract void run(); 5. void changeGear(){System.out.println("gear changed");} 6. } 7. //Creating a Child class which inherits Abstract class 8. class Honda extends Bike{ 9. void run(){System.out.println("running safely..");} 10. } 11. //Creating a Test class which calls abstract and non-abstract methods 12. class TestAbstraction2{ 13. public static void main(String args[]){ 14. Bike obj = new Honda(); 15. obj.run(); 16. obj.changeGear(); 17. } 18. }   [Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=TestAbstraction2)  bike is created  running safely..  gear changed  **Rule: If there is an abstract method in a class, that class must be abstract.**   1. class Bike12{ 2. abstract void run(); 3. }   [Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=Bike12)  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.** |

**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:**

1. interface <interface\_name>{
3. // declare constant fields
4. // declare methods that abstract
5. // by default.
6. }

**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**

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](http://www.javatpoint.com/opr/test.jsp?filename=A6)

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

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. }}

[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=TestInterface1)

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

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](http://www.javatpoint.com/opr/test.jsp?filename=TestInterface2)

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.

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](http://www.javatpoint.com/opr/test.jsp?filename=A7)

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:

1. interface Printable{
2. void print();
3. }
4. interface Showable{
5. void print();
6. }
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. }

[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=TestInterface3)

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 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](http://www.javatpoint.com/opr/test.jsp?filename=TestInterface4)

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](http://www.javatpoint.com/opr/test.jsp?filename=TestInterfaceDefault)

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. }
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. }}

[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=TestInterfaceStatic)

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) 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(); } |

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.

1. //Creating interface that has 4 methods
2. interface A{
3. void a();//bydefault, public and abstract
4. void b();
5. void c();
6. void d();
7. }
9. //Creating abstract class that provides the implementation of one method of A interface
10. abstract class B implements A{
11. public void c(){System.out.println("I am C");}
12. }
14. //Creating subclass of abstract class, now we need to provide the implementation of rest of the methods
15. class M extends B{
16. public void a(){System.out.println("I am a");}
17. public void b(){System.out.println("I am b");}
18. public void d(){System.out.println("I am d");}
19. }
21. //Creating a test class that calls the methods of A interface
22. class Test5{
23. public static void main(String args[]){
24. A a=new M();
25. a.a();
26. a.b();
27. a.c();
28. a.d();
29. }}

[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=Test5)

Output:

I am a

I am b

I am c

I am d