Abstract class in Java

A class that is declared with abstract keyword, is known as abstract class in java. It can have abstract and non-abstract methods (method with body).

Abstraction in Java

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

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

Ways to achieve Abstaction

There are two ways to achieve abstraction in java

1. Abstract class
2. Interface

Abstract class in Java

A class that is declared as abstract is known as **abstract class**. It needs to be extended and its method implemented. It cannot be instantiated.

Example abstract class

```
1. abstract class A{}
```

---

abstract method

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

Example abstract method

```
1. abstract void printStatus();//no body and abstract
```

Example of abstract class that has abstract method

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

```
1. abstract class Bike{
2.   abstract void run();
3. }
4. class Honda4 extends Bike{
5. void run(){System.out.println("running safely..");}
6. public static void main(String args[]){
7.  Bike obj = new Honda4();
8.  obj.run();
9. }
10. }
```

Output:

```
running safely..
```

Understanding the real scenario of abstract class

In this example, Shape is the abstract class, its implementation is provided by the Rectangle and Circle classes. Mostly, we don't know about the implementation class (i.e. hidden to the end user) and object of the implementation class is provided by the **factory method**.

A **factory method** is the method that returns the instance of the class.

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

File: TestAbstraction1.java

```
1. abstract class Shape{
2. abstract void draw();
3. }
4. //In real scenario, implementation is provided by others i.e. unknown by end user
5. class Rectangle extends Shape{
6. void draw(){System.out.println("drawing rectangle");}
7. }
8. class Circle1 extends Shape{
9. void draw(){System.out.println("drawing circle");}
```

```
10. }
11. //In real scenario, method is called by programmer or user
12. class TestAbstraction1{
13. public static void main(String args[]){
14. Shape s=new Circle1();//In real scenario, object is provided through method e.g. getShape
    () method
15. s.draw();
16. }
17. }
```

Output:

```
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.  }
10.
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. }}
```

Output

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

Abstract class having constructor, data member, methods etc.

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

File: TestAbstraction2.java

```
1. //example of abstract class that have method body
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.
8.  class Honda extends Bike{
9.  void run(){System.out.println("running safely..");}
10. }
11. class TestAbstraction2{
12. public static void main(String args[]){
13.  Bike obj = new Honda();
14.  obj.run();
15.  obj.changeGear();
16. }
17.}
```

Output

```
bike is created
running safely..
gear changed
```

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

```
1. class Bike12{
2. abstract void run();
3. }
```

Output

```
compile time error
```

> ***Rule: If you are extending any abstract class that have 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.

```
1. interface A{
2. void a();
3. void b();
4. void c();
5. void d();
6. }
7.
8. abstract class B implements A{
9. public void c(){System.out.println("I am C");}
10.}
11.
12.class M extends B{
13.public void a(){System.out.println("I am a");}
14.public void b(){System.out.println("I am b");}
15.public void d(){System.out.println("I am d");}
16.}
17.
18.class Test5{
19.public static void main(String args[]){
20.A a=new M();
21.a.a();
22.a.b();
23.a.c();
24.a.d();
25.}}
```

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

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?

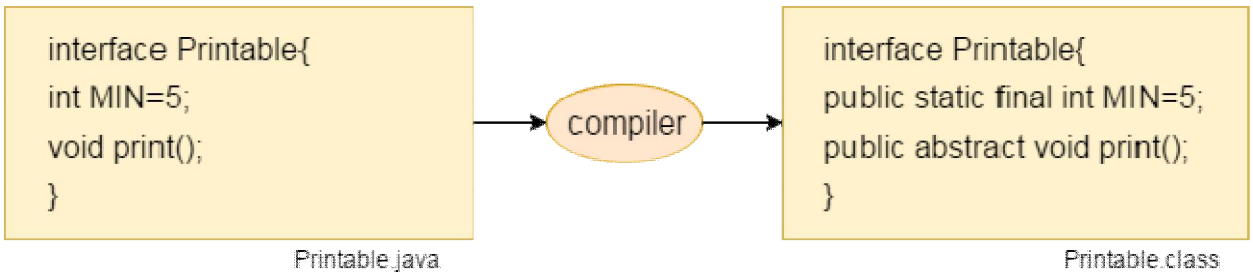
There are mainly three reasons to use interface. They are given below.

- It is used to achieve abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.

Internal addition by compiler

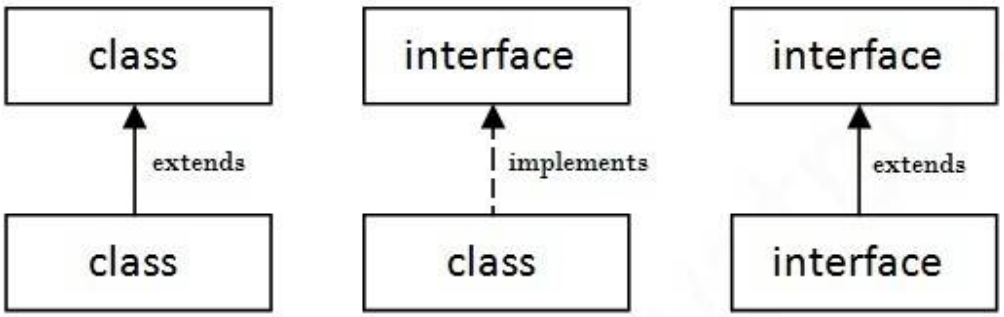
> ***The java compiler adds public and abstract keywords before the interface method. More, it adds public, static and final keywords before data members.***

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



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 A implements printable{
5. public void print(){System.out.println("Hello");}
6.
7. public static void main(String args[]){
8. A obj = new A();
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. 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

```
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. }}
```

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. }
```

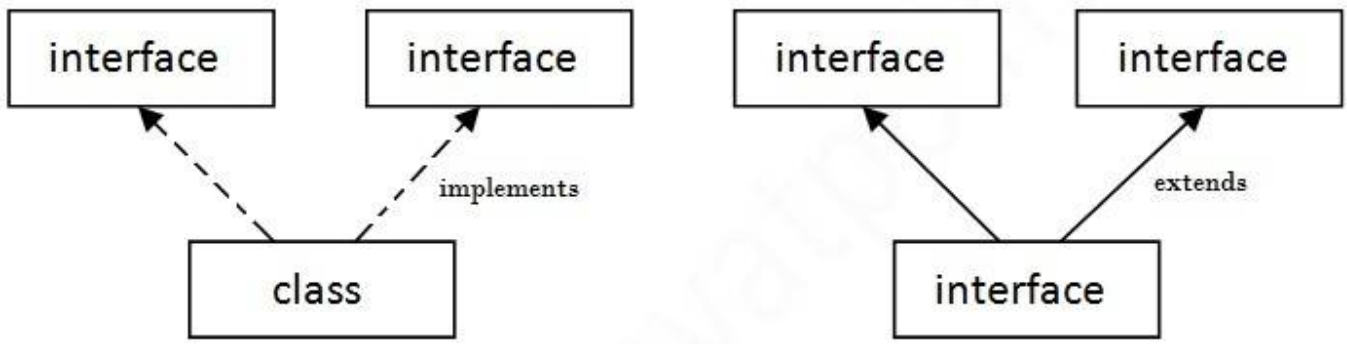
```
4. class SBI implements Bank{
5. public float rateOfInterest(){return 9.15f;}
6. }
7. class PNB implements Bank{
8. public float rateOfInterest(){return 9.7f;}
9. }
10. class TestInterface2{
11. public static void main(String[] args){
12. Bank b=new SBI();
13. System.out.println("ROI: "+b.rateOfInterest());
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 A 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. A obj = new A();
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 TestTnterface3 implements Printable, Showable{
9.  public void print(){System.out.println("Hello");}
10. public static void main(String args[]){
11. TestTnterface1 obj = new TestTnterface1();
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
```

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
```

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. }}
```

Test it Now

Output:

```
drawing rectangle
27
```

Q) What is marker or tagged interface?

An interface that have no member is known as 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. }
```

Nested Interface in Java

Note: An interface can have another interface i.e. known as nested interface. We will learn it in detail in the nested classes chapter. For example:

```
1. interface printable{
2.  void print();
3.  interface MessagePrintable{
4.    void msg();
5.  }
6. }
```

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.
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 have static methods, main method and constructor**.	Interface **can't have static methods, main method or constructor**.
5) Abstract class **can provide the implementation of interface**.	Interface **can't provide the implementation of abstract class**.
6) The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.

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

Final Keyword In Java

The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

1. variable
2. method
3. class

The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only. We will have detailed learning of these. Let's first learn the basics of final keyword. **Java final variable**

If you make any variable as final, you cannot change the value of final variable(It will be constant).

Example of final variable

There is a final variable speedlimit, we are going to change the value of this variable, but It can't be changed because final variable once assigned a value can never be changed.

```
1. class Bike9{
```

```
2.  final int speedlimit=90;//final variable
3.  void run(){
4.   speedlimit=400;
5.  }
6.  public static void main(String args[]){
7.  Bike9 obj=new  Bike9();
8.  obj.run();
9.  }
10. }//end of class
```

```
Output:Compile Time Error
```

Java final method

If you make any method as final, you cannot override it.

Example of final method

```
1.  class Bike{
2.    final void run(){System.out.println("running");}
3.  }
4.
5.  class Honda extends Bike{
6.     void run(){System.out.println("running safely with 100kmph");}
7.
8.     public static void main(String args[]){
9.     Honda honda= new Honda();
10.    honda.run();
11.    }
12. }
```

Test it Now

```
Output:Compile Time Error
```

---

3) Java final class

If you make any class as final, you cannot extend it.

Example of final class

```
final class Bike{}

class Honda1 extends Bike{
  void run(){System.out.println("running safely with 100kmph");}

  public static void main(String args[]){
  Honda1 honda= new Honda();
  honda.run();
  }
}
```

Test it Now

```
Output:Compile Time Error
```

Q) Is final method inherited?

Ans) Yes, final method is inherited but you cannot override it. For Example:

```
class Bike{
  final void run(){System.out.println("running...");}
}
class Honda2 extends Bike{
   public static void main(String args[]){
    new Honda2().run();
   }
}
```

Test it Now

```
Output:running...
```

Q) What is blank or uninitialized final variable?

A final variable that is not initialized at the time of declaration is known as blank final variable.

If you want to create a variable that is initialized at the time of creating object and once initialized may not be changed, it is useful. For example PAN CARD number of an employee.

It can be initialized only in constructor.

Example of blank final variable

```
class Student{
int id;
String name;
final String PAN_CARD_NUMBER;
...
}
```

Que) Can we initialize blank final variable?

Yes, but only in constructor. For example:

```
class Bike10{
  final int speedlimit;//blank final variable

  Bike10(){
  speedlimit=70;
  System.out.println(speedlimit);
  }

  public static void main(String args[]){
    new Bike10();
 }
}
```

Test it Now

```
Output:70
```

---

static blank final variable

A static final variable that is not initialized at the time of declaration is known as static blank final variable. It can be initialized only in static block.

Example of static blank final variable

```
class A{
  static final int data;//static blank final variable
  static{ data=50;}
  public static void main(String args[]){
    System.out.println(A.data);
  }
```

```
7. }
```

Q) What is final parameter?

If you declare any parameter as final, you cannot change the value of it.

```
1. class Bike11{
2.   int cube(final int n){
3.    n=n+2;//can't be changed as n is final
4.    n*n*n;
5.   }
6.   public static void main(String args[]){
7.     Bike11 b=new Bike11();
8.     b.cube(5);
9.  }
10.}
```

Test it Now

```
Output:Compile Time Error
```

Q) Can we declare a constructor final?

No, because constructor is never inherited.