Interface in Java

An interface in java is a blueprint of a class. It has static constants and abstract methods only.

The interface in java is a mechanism to achieve fully abstraction. There can be only abstract methods in the java interface not method body. It is used to achieve fully 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?**

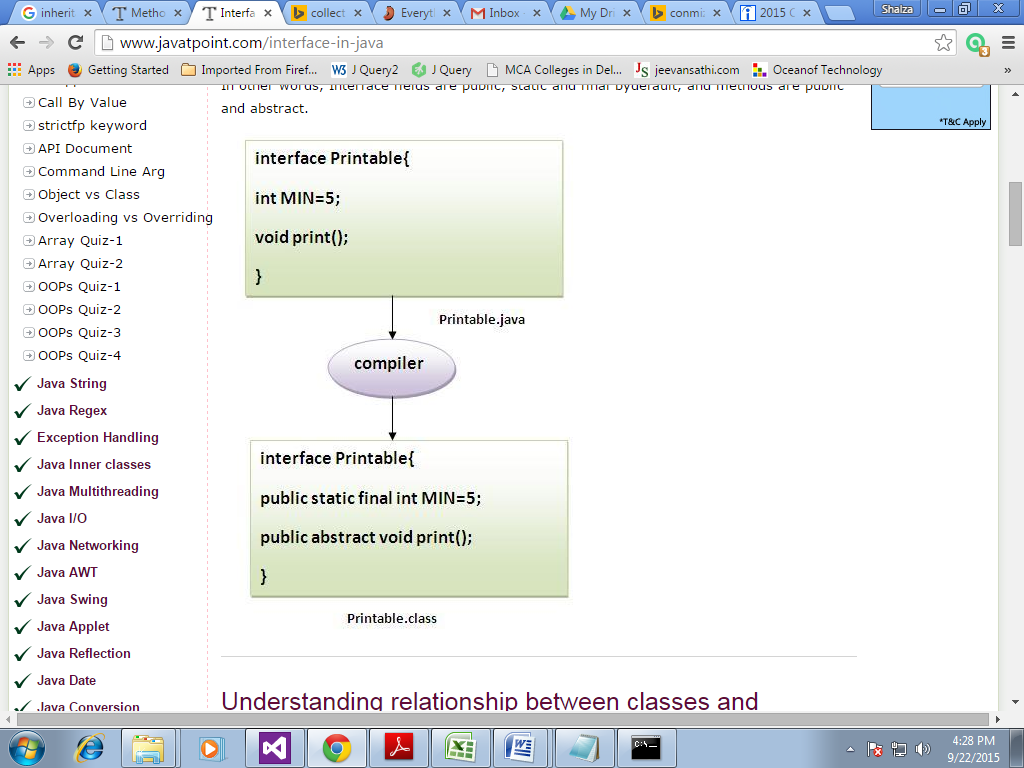
There are mainly three reasons to use interface. They are given below.

1) It is used to achieve fully abstraction.

2) By interface, we can support the functionality of multiple inheritance.

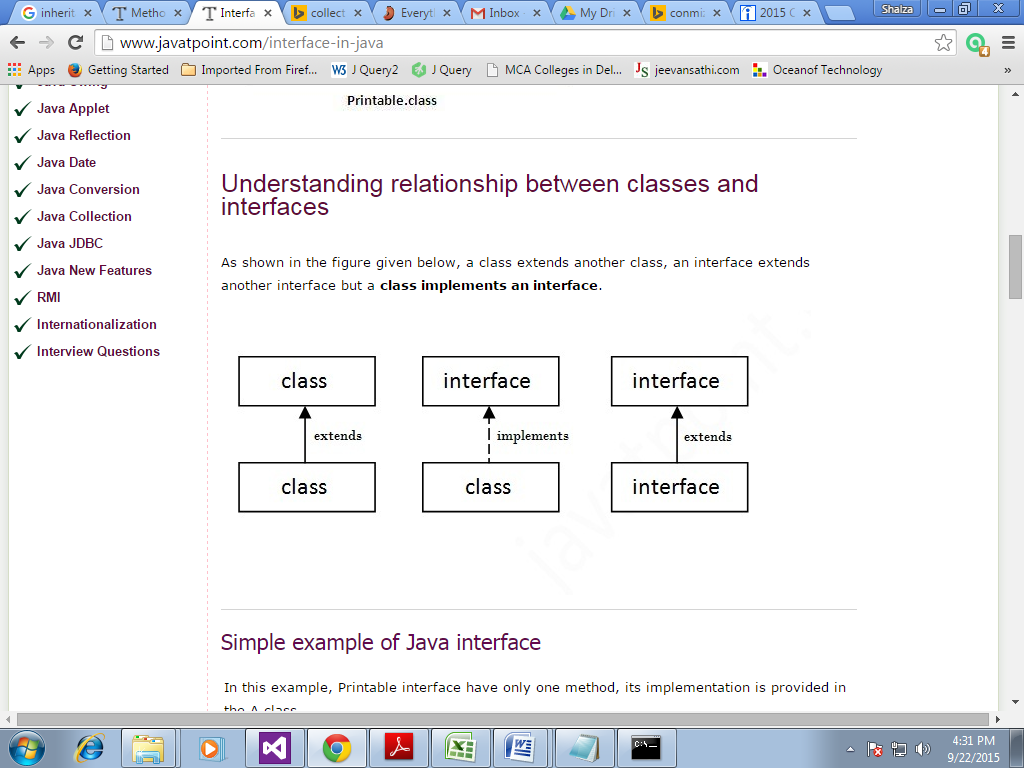
3) It can be used to achieve loose coupling.

**In other words, Interface fields are public, static and final bydefault, 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.



**Simple example of Java interface**

In this example, Printable interface have only one method, its implementation is provided in the A 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();

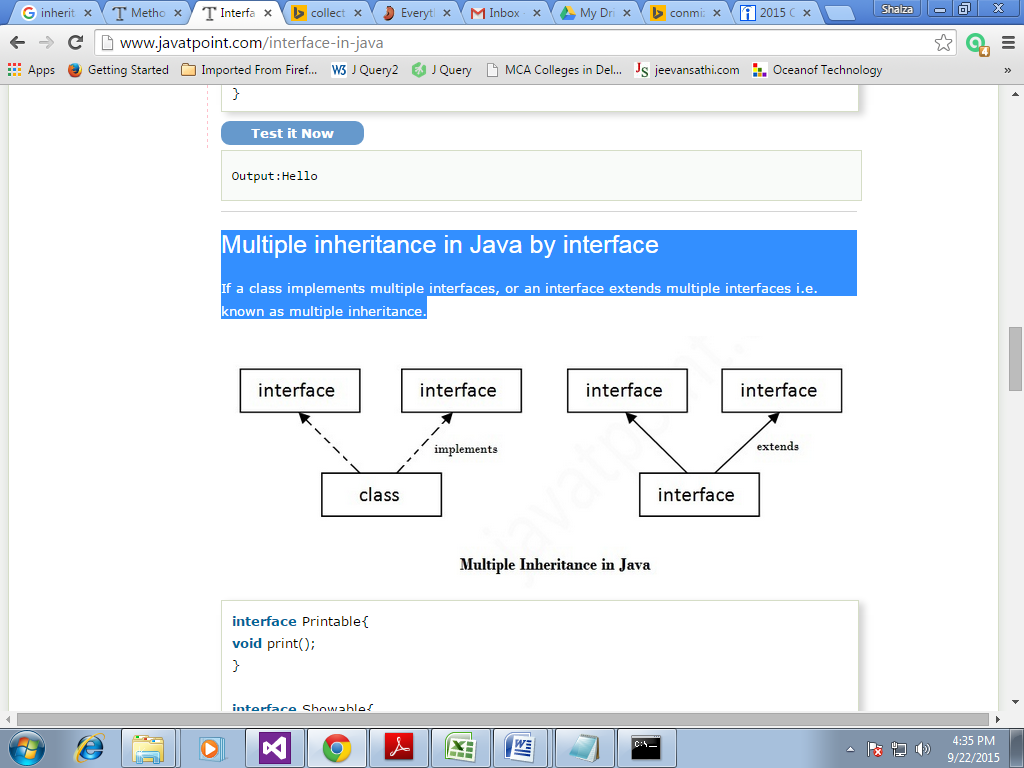
obj.print();

} }

Output:Hello

**Multiple inheritance in Java by interface**

If a class implements multiple interfaces, or an interface extends multiple interfaces i.e. 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();

}}

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. But it is supported in case of interface because there is no ambiguity as implementation is provided by the implementation class. For example:

interface Printable{

void print();

}

interface Showable{

void print();

}

class TestTnterface1 implements Printable,Showable{

public void print(){System.out.println("Hello");}

public static void main(String args[]){

TestTnterface1 obj = new TestTnterface1();

obj.print();

} }

**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 .

interface Printable{

void print(); }

interface Showable extends Printable{

void show();}

class Testinterface2 implements Showable{

public void print(){System.out.println("Hello");}

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

public static void main(String args[]){

Testinterface2 obj = new Testinterface2();

obj.print();

obj.show();

} }

**Output:**

Hello

Welcome

=====

**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.

//How Serializable interface is written?

public interface Serializable{

}

**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:

interface printable{

void print();

interface MessagePrintable{

void msg();

}

}

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(); } |

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 methods

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

**Java Nested Interface**

An interface i.e. declared within another interface or class is known as nested interface. The nested interfaces are used to group related interfaces so that they can be easy to maintain. The nested interface must be referred by the outer interface or class. It can't be accessed directly.

**Points to remember for nested interfaces**

There are given some points that should be remembered by the java programmer.

1. Nested interface must be public if it is declared inside the interface but it can have any access modifier if declared within the class.
2. Nested interfaces are declared static implicitly.

**Syntax of nested interface which is declared within the interface**

interface interface\_name{

...

interface nested\_interface\_name{

...

} }

**Syntax of nested interface which is declared within the class**

class class\_name{

...

interface nested\_interface\_name{

...

} }

**Example of nested interface which is declared within the interface**

In this example, we are going to learn how to declare the nested interface and how we can access it.

interface Showable{

void show();

interface Message{

voidmsg();

} }

class TestNestedInterface implements Showable.Message{

public void msg(){System.out.println("Hello nested interface");}

public static void main(String args[]){

Showable.Message message=new TestNestedInterface(); //upcasting//here

message.msg();

} }

**Internal code generated by the java compiler for nested interface Message**

The java compiler internally creates public and static interface as displayed below:.

public static interface Showable$Message

{

public abstract void msg();

}

**Example of nested interface which is declared within the class**

See how can we define an interface inside the class and how can we access it.

class A{

interface Message{

voidmsg();

}}

class TestNestedInterface implements A.Message{

public void msg(){System.out.println("Hello nested interface");}

public static void main(String args[]){

A.Message message=new TestNestedInterface();//upcasting here

message.msg();

} }

**Output**:hello nested interface

**Q.Can we define a class inside the interface?**

Yes, If we define a class inside the interface, java compiler creates a static nested class. Let's see how can we define a class within the interface:

interface M{

class A{} }

**Key points: Here are the key points to remember about interfaces:**

1) We can’t instantiate an interface in java.

2) Interface provides complete abstraction as none of its methods can have body. On the other hand, abstract class provides partial abstraction as it can have abstract and concrete(methods with body) methods both.

3) implements keyword is used by classes to implement an interface.

4) While providing implementation in class of any method of an interface, it needs to be mentioned as public.

5) Class implementing any interface must implement all the methods, otherwise the class should be declared as “abstract”.

6) Interface cannot be declared as private, protected or transient.

7) All the interface methods are by default abstract and public.

8) Variables declared in interface are public, static and final by default.

interface Try

{

int a=10;

public int a=10;

public static final int a=10;

final int a=10;

static int a=0;

}

**All of the above statements are identical.**

9) Interface variables must be initialized at the time of declaration otherwise compiler will through an error.

interface Try

{

int x;//Compile-time error

}

Above code will throw a compile time error as the value of the variable x is not initialized at the time of declaration.

10) Inside any implementation class, you cannot change the variables declared in interface because by default, they are public, static and final. Here we are implementing the interface “Try” which has a variable x. When we tried to set the value for variable x we got compilation error as the variable x is public static final by default and final variables can not be re-initialized.

Class Sample implements Try

{

public static void main(String arg[])

{

x=20; //compile time error

}

}

11) Any interface can extend any interface but cannot implement it. Class implements interface and interface extends interface.

12) A class can implement any number of interfaces.

13) If there are two or more same methods in two interfaces and a class implements both interfaces, implementation of the method once is enough.

interface A

{

public void aaa();

}

interface B

{

public void aaa();

}

class Central implements A,B

{

public void aaa()

{

//Any Code here

}

public static void main(String arg[])

{

//Statements

}

}

14) A class cannot implement two interfaces that have methods with same name but different return type.

interface A

{

public void aaa();

}

interface B

{

public int aaa();

}

class Central implements A,B

{

public void aaa() // error

{

}

public int aaa() // error

{

}

public static void main(String arg[])

{

}

}

15) Variable names conflicts can be resolved by interface name e.g:

interface A

{

int x=10;

}

interface B

{

int x=100;

}

class Hello implement A,B

{

public static void Main(String arg[])

{

System.out.println(x); // reference to x is ambiguous both variables are x

System.out.println(A.x);

System.out.println(B.x);

}

}

**Benefits of having interfaces:**

Following are the advantages of interfaces:

Without bothering about the implementation part, we can achieve the security of implementation

In java, multiple inheritance is not allowed, However by using interfaces we can achieve the same . A class can extend only one class but can implement any number of interfaces. It saves you from Deadly Diamond of Death(DDD) problem.

**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(); } |

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 methods

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