What is an interface

Interface looks like class but it is not a class. An interface can have methods and variables just like the class but the methods declared in interface are by default abstract (only method signature, no body). **Also, the variables declared in an interface are public, static & final by default.** We will discuss these points in detail later in this post.

What is the use of interfaces

As stated above they are used for abstraction. Since methods in interfaces does not have body, they have to be implemented by the class before you can access them. The class that implements interface must implement all the methods of that interface. Also, java programming language does not support multiple inheritance, using interfaces we can achieve this as a class can implement more than one interfaces, however it cannot extend more than one classes.

Declaration
Interfaces are declared by specifying a keyword "interface". E.g.:

```
interface MyInterface
{
  /* All the methods are public abstract by default
   * Note down that these methods are not having body
   */
  public void method1();
  public void method2();
}
```

Interface Implementation

This is how a class implements an interface. It has to provide the body of all the methods that are declared in interface.
Note: Class implements interface but an interface extends another interface.

```
interface MyInterface
{
  public void method1();
  public void method2();
}
class XYZ implements MyInterface
{
 public void method1()
 {
    System.out.println("implementation of method1");
 }
 public void method2()
```

```
  {
     System.out.println("implementation of method2");
  }
  public static void main(String arg[])
  {
     MyInterface obj = new XYZ();
     obj. method1();
  }
}
```

Output:

```
implementation of method1
```

Interface and Inheritance

As I discussed above that one interface can not implement another interface. It has to extend the other interface if required. See the below example where we have two interfaces Inf1 and Inf2. Inf2 extends Inf1 so If class implements the Inf2 it has to provide implementation of all the methods of interfaces Inf1 and Inf2.

```
public interface Inf1{
   public void method1();
}
public interface Inf2 extends Inf1 {
   public void method2();
}
public class Demo implements Inf2{
  public void method1(){
    //Implementation of method1
  }
  public void method2(){
    //Implementation of method2
  }
}
```

In the above program my “Demo” class is implementing only one interface “Inf2″ however it has to provide the implementation of all the methods of interface “Inf1″ too, because interface Inf2 extends Inf1.

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 other interface but cannot implement it. Class implements interface and interface extends interface.

12) A **class** can implements any **number of interfaces**.

13) If there are having **two or more same methods** in two interfaces and a class implements both interfaces, implementation of one method 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) Methods with same signature but different return type can't be implemented at a time for two or more interfaces.

```
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:

1. Without bothering about the implementation part, we can achieve the security of implementation
2. In java, **multiple inheritance** is not allowed, However by using interfaces you 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.

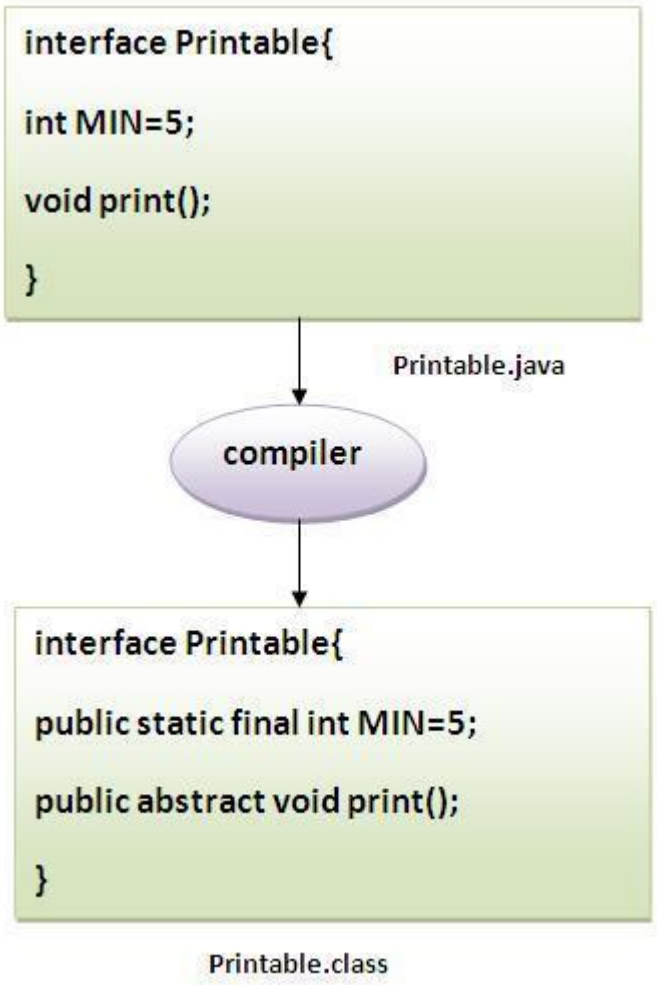
Why use Java interface?

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

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

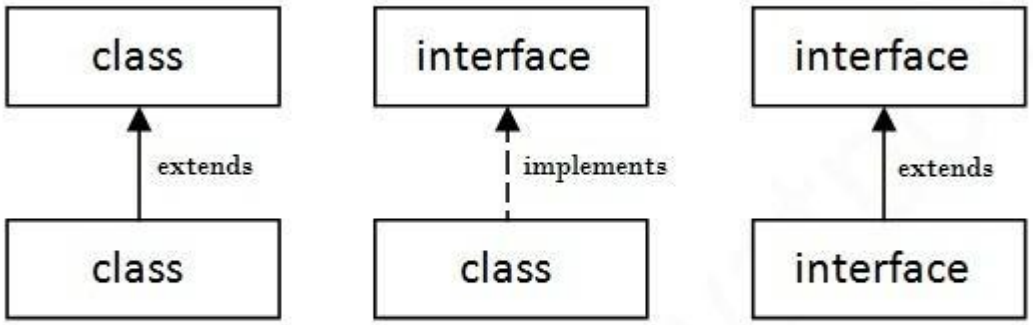
The java compiler adds public and abstract keywords before the interface method and public, static and final keywords before data members.

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.

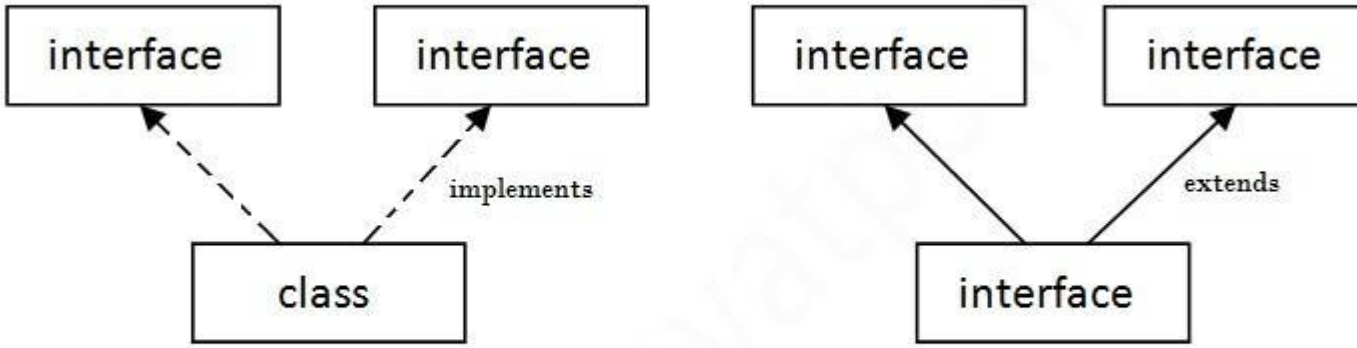
```
1. interface printable{
2. void print();
3. }
4.
5. class A6 implements printable{
6. public void print(){System.out.println("Hello");}
7.
8. public static void main(String args[]){
9. A6 obj = new A6();
10. obj.print();
11.  }
12. }
```

Test it Now

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.



Multiple Inheritance in Java

```
1.  interface Printable{
2.  void print();
3.  }
4.
5.  interface Showable{
6.  void show();
7.  }
8.
9.  class A7 implements Printable,Showable{
10.
11. public void print(){System.out.println("Hello");}
12. public void show(){System.out.println("Welcome");}
13.
14. public static void main(String args[]){
15. A7 obj = new A7();
16. obj.print();
17. obj.show();
18.  }
19. }
```

Test it Now

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:

```
1.  interface Printable{
```

```
2.  void print();
3.  }
4.
5.  interface Showable{
6.  void print();
7.  }
8.
9.  class testinterface1 implements Printable,Showable{
10.
11. public void print(){System.out.println("Hello");}
12.
13. public static void main(String args[]){
14. testinterface1 obj = new testinterface1();
15. obj.print();
16.  }
17. }
```

Test it Now

Hello

As you can see in the above example, Printable and Showable interface have same methods but its implementation is provided by class A, 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 Testinterface2 implements Showable{
8.
9.  public void print(){System.out.println("Hello");}
10. public void show(){System.out.println("Welcome");}
11.
12. public static void main(String args[]){
13. Testinterface2 obj = new Testinterface2();
14. obj.print();
15. obj.show();
16.  }
17. }
```

Test it Now

Hello
Welcome