Interface

→ An interface 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 & multiple Inheritance in Java.

→ Interface also represents the IS-A relationship

→ It can't be instantiated just like the abstract class.

→ Since Java 8, we can have default & static methods in an interface.

→ Since Java 9, we can have private methods in an interface

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 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 constants fields
    // declare methods that abstract
    // by default
}
```

```
interface Printable
{ int MIN=5;
  void print();
}
```

→ [Compiler] →

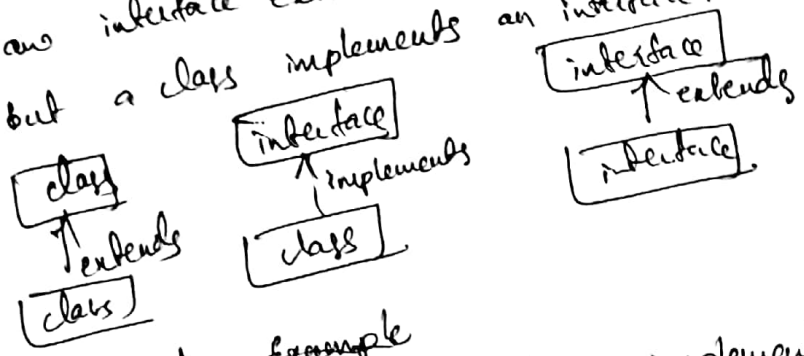
```
interface Printable
{ public static final int MIN=5;
  public abstract void print();
}
```

The relationship b/w classes & interfaces

→ a class extends another class

→ an interface extends another interface / more interfaces

→ but a class implements an interface / more interfaces



~~Java Interface Example~~

→ A class can inherit another class & implement an interface

→ An int

→ Java Interface Example

```
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();
   }
}
```

O/p : Hello

→ In the above example Printable interface has only one method, and its implementation is provided in the A6 class.

Example 2 :-

```
interface Drawable
{   void draw();
}
class Rectangle implements Drawable
{  public void draw()
   {  S.o.println("drawing rectangle");
   }
}
class Circle implement Drawable
{  public void draw()
   {  S.o.println("drawing Circle");
   }
}
```

```
class TestInterface
{  public static void main
   (String args[])
   {  Drawable d=new Circle();
      d.draw();
   }
}
```

o/p: drawing Circle.

Differences b/w Interface and abstract class

Interface

1) Java interface are implicitly abstract & can't have implementations
2) variables declared in a Java interface is by default final.
3) Members of Java are public by default.
4) Java interface should be implemented using keyword "implements"
5) An interface can be extended another Java Interface only
6) Interface is absolutely abstract and can't be instantiated
7) Java interfaces are slow as it requires extra indirection

Abstract:-

1) Java abstract class can have ~~been~~ instance methods that implements a default behaviour.
2) An abstract class may contain non-final variables.
3) A java abstract class can have the usual flavours of class members like private, protected, etc.
4) A java abstract class should be extended using keyword "extends"
5) an abstract class can be extend another java class and implement multiple Java Interface.
6) A java abstract class also can't be instantiated, but can be invoked if a main() exists.
7) Comparatively fast.

Implementing interfaces

→ Once the interface has been defined, one or more classes can implement that interface

→ To implement an interface, include the implements clause in a class definition.

Syntax:-

```
class classname [extends superclass] [implements interface [,interface ...]]
{
    // class body
}
```

→ The methods that implement an interface must be declared public

→ The type signature of the implementing method must match exactly the type signature specified in the interface definition.

→
```
interface Callback
{ void callback(int param);
}
class Client implements Callback
{   //implements Callback's interface
    public void callback(int p)
    { System.out.println("Callback called with "+p);
    }
}
```

→ Also you can take additional members of Client.

```
class Client implements Callback
{ public void callback(int p)
  { S.o.pln("Callback called with = "+p); }
  void nonInterfaceMeth()
  { System.o.pln("Classes that implement interfaces" +
                 may also define other members, too");
  }
}
```

Accessing Implementations through Interface References

→ First you have to create reference variable of interface rather than a class. assign class object to reference variable

→ ~~create~~ through the reference variable you can call a method. then the correct version ~~of the~~ will be called based on the actual instance of the interface being referred to.

→ This is one of the key features of Java.

→ the method to be executed is looked up dynamically at run time.

Example:-

```
interface Callback
{
    void callback(int param);
}
class Client implements Callback
{
    public void callback(int p)
    {
        System.out.println("callback is called with "+p);
    }
}
class AnotherClient implements Callback
{
    public void callback(int p)
    {
        S.o.println("Another version of callback");
        System.out.println("p squared is "+(p*p));
    }
}
class TestInterface2
{
    public static void main(String args[])
    {
        Callback c=new Client();
        AnotherClient ob=new AnotherClient();
        c.callback(42);
        c=ob;          // c now refers to AnotherClient object
        c.callback(42);
    }
}
```

o/p:- callback called with 42
Another version of callback
p squared is 1764

Partial Implementation

If a class includes an interface but doesn't fully implement the methods defined by that interface, then that class must be declared as abstract

Ex:- abstract class Incomplete implements Callback

```
{   int a,b;
    void show()
    { System.out.println(a+" "+b);
    }
    //...
}
```

→ The class Incomplete doesn't implement callback() and must be declared as abstract. Any class that inherits Incomplete must implement callback() or be declared abstract itself.

Nested Interfaces

→ An interface can be declared a member of a class or another interface. Such an interface is called a member interface or a nested interface.

→ A nested interface can be declared as public, private, or protected. or use the default access level.

→ When a nested interface is used outside of its enclosing scope, it must be qualified by the name of the class or interface of which it is a member.

Ex:- Write a java program for demonstrating a nested interface.

```
class A
{   //this is a nested interface
    public interface NestedIF
    {  boolean isNotNegative(int x);
    }
}
//B implements the nested interface
class B implements A.NestedIF
{ public boolean isNotNegative(int x)
   {   return x<0?false:true;
   }
}
class NestedIFDemo
{ public static void main(String args[])
   {
      A.NestedIF nif = new B();
      if(nif.isNotNegative(10))
      S.o.println("10 is not negative");
      if(nif.isNotNegative(-12))
       S.o.println("this won't be displayed");
   }
}
```

Multiple Inheritance in Java using Interface :-

```
interface vehicleone
{  int speed=90;
   public void distance();
}
interface vehicletwo
{  int distance=100;
   public void speed();
}
class vehicle implements vehicleone, vehicletwo
{  public void distance()
   {  int distance=speed*100;
      System.out.println("distance travelled is "+
                         distance);
   }
   public void speed()
   {  int speed=distance/100;
      System.out.println("speed is "+speed+" km/sec");
   }
}
class multipleInheritanceUsingInterface
{  public static void main(String args[])
   {  System.out.println("vehicle");
      vehicle obj=new vehicle();
      obj.distance();
      obj.speed();
   }
}
```

o/p: vehicle
distance travelled is 9000
speed is 1 hr/sec

Multiple Inheritance is not supported in 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.

naming confusion

Syntax:-

```
public interface A
{  //Do something
}
public interface B
{  //Do something
}
public interface C extends A, B
{  //Do something
}
```