Interfaces

```
abstract class Test1
{
    abstract public void meth1();
    abstract public void meth2();
}

class Test2 extends Test1
{
    public void meth1()
    {
    =
    }
    public void meth2()
    {
    =
    }
}
```

```
Test1 t=new Test2();
```

```
interface Test1
{
    void meth1();
    void meth2();
}

class Test2 implements Test1
{
    public void meth1()
    {
    =
    }
    public void meth2()
    {
    =
    }
}
```

--> An interface is a fully abstract class which includes a group of abstract methods (i.e. methods without a body).

--> Interfernce ek trah sai abstract class hi hai bas ise mai "abstract class" likhna ki jga ham "interface" likhta hai. Also jab "inheritance" karna hota hai to "extend" - keyword ka place par "implements" - keyword ka use karta hai.

--> Jis trah "abstract - class" ka object create nahi hota ha waisa hi "interface" ka bhi nahi hota hai. Bas ham uska reference create kar sakta hai.

--> Aur, interfaces super class kai reference ka use kar ka subclass ka help object create ho sakta hai.

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** 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.

```
//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 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();
```

```
}
}
```

Test it Now

Output:

```
I am a
I am b
I am c
I am d
```

Abstract vs interface

Abstract class

- If we are talking about implementation but not completely (partial implementation) then we should go for abstract class.
- Every method present inside abstract class need not be public and abstract.
- There are no restrictions on abstract class method modifiers.
- Every abstract class variable need not be a public static final.
- Inside the abstract class we can take constructor.

interface

- If we don't' know anything about implementation just we have requirement specification then we should go for an interface.
- Every method present inside the interface is always public and abstract whether we are declaring or not.
- We can't declare interface methods with the modifiers private, protected, final, static, synchronized, native, strictfp.
- Every interface variable is always a public static final whether we are declaring or not following modifiers. private, protected, transient, volatile.
- Inside the interface we can't take constructor.

Main.java 2, U Java8Fe.java U

Java8Fe.java > A > config()

```java
interface A
{
    void show();
    default void config()
    {

    }
    static void abc()
    {
        System.out.println("in abc");
    }
}

class B implements A
{
    public void show()
    {
        System.out.println("in show");
    }
}

public class Java8Fe {
    Run | Debug
    public static void main(String[] args) {
        A.abc();
        A obj = new B();
        obj.show();
        obj.config();
    }
}
```

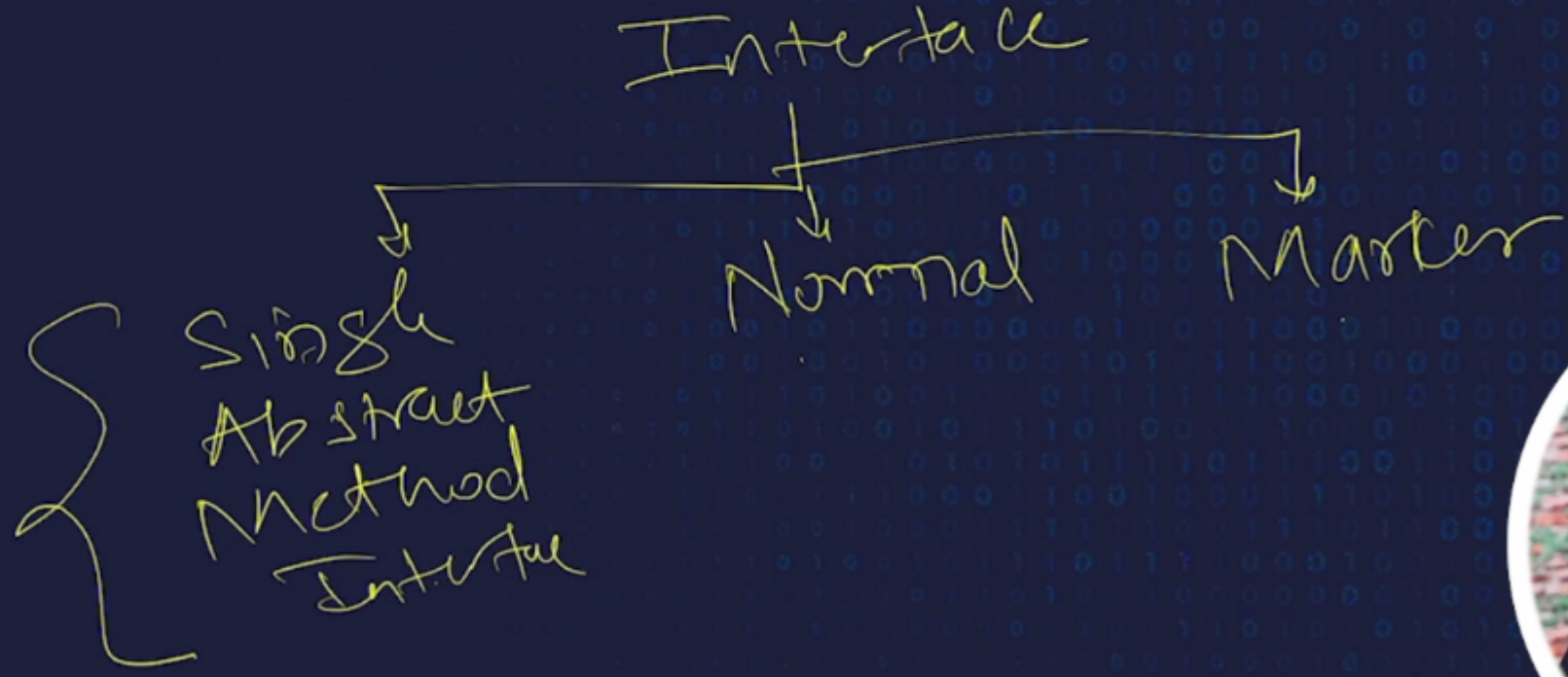
--> Above version of java 8, ham interface mai "method" ko define kar sakta hai but sirf hma uska aaga "default - keyboard" ka use karna hota hai.

--> Also, interface ka andar ham static method bhi define kar sakta hai.

--> Static mth. ko ham direct access kar sakta hai but non static method ko use karna ka liya hma uska class ka object create karna hota, then uska use karta hai.

JAVA 8 Features over Interfaces:

1. Default Methods in Interfaces
2. Static Methods in Interfaces
3. Functional Interfaces



Functional Interface

- If any Java interface allows only one abstract method then it is called a "Functional Interface".

- To make any interface as Functional Interface then we have to use the following annotation just above of the interface. @FunctionalInterface
  Ex: java.lang.Runnable
  java.lang.Comparable

```java
package interfacepractice;

interface Test
{
    final static int X=10;
    public abstract void meth1();
    public abstract void meth2();

    private void meth3()
    {
        System.out.println("Meth3 of Test");
    }
    default void meth5()
    {
        meth3();
    }

}

interface Test2 extends Test
{
    void meth4();
}

class My implements Test2
{
    public void meth1(){}
    public void meth2(){}
    public void meth4(){}
}
public class InterfacePractice
{
    public static void main(String[] args)
    {
        System.out.println(Test.X);
        My m=new My();
        m.meth3();

    }
}
```

Do or Don't for interfaces

--> By default, methods are Public and Abstract.

--> As methods are to be implemented by the classes, they can't be made private.

--> Identifiers can be used in interfaces but the Identifiers must be given in Uppercases.

--> Identifiers are by default final and static.

--> Method inside an interface cannot have body but the method can have body if the method is static.

--> Static members can be accessed in main method by using interface name and dot operator.

--> An interface can be extended from another interfaces.