Interface :

- As we know that java doesn't support multiple inheritance and its related combination using classes , but we can achieve them by using "interface"
- Interface is the standard and public ways of defining the classes
- Interface is also known as pure abstract class
  - A class is collection of non abstract methods
  - An abstract class is the collection of both abstract or non abstract methods
  - An interface is the collection of abstract methods only
- What is the difference between CLASS | ABSTRACT CLASS | INTERFACE ?

class	Abstract class	Interface
1.we can declare all type of variables [instance | static fields | final]	We can declare all types of variables	We can declare only final static combination
2.we can define constructor	We can define constructor	No Constructors
3.we can define non abstract methods	We can define non abstract methods or abstract methods	We can declare only abstract methods
4.it is optional to inherit	Need to be inherited [extends]	Need to be implemented [implements]
5.class can be referenced and it can be instantiated	Abstract class can be referenced but can't be instantiated	Interface can be referenced but can't be instantiated.

- By default all the variables in the interface "public final static"
- By default all the methods in the interface "public abstract"

Shashi Kumar - SSSIT

Syn: [modifiers] <interface> <InterfaceName>
{
Declare final static fields
Declare public abstract methods
}

Example :

```
//IntA.java
interface IntA
{
        int x=10;   //public static final
        void method1();  //public abstract
}
```

```
E:\Java_Online11\INTERFACE>javac IntA.java
E:\Java_Online11\INTERFACE>javap IntA
Compiled from "IntA.java"
interface IntA {
  public static final int x;
  public abstract void method1();
}
```

Note:

What is difference between interface , Functional Interface and Marker interface ?

- **An Interface is collection of abstract methods**
- **Functional interface is an interface which allows you to declare only one abstract method in it, if you want to define a functional interface then we to use [@FunctionalInterface]**

Shashi Kumar - SSSIT

```
//IA.java
@FunctionalInterface
interface IA
{
        public abstract void method1();
}
```

- Marker interface is an interface which is not having any abstract methods in it. In simple word it is an empty interface
  - java.lang.Cloneable | java.awt.event.ActionListener
  - java.io.Serializable

Note :

- If any class is implemented by an interface with abstract methods then we must override all the abstract methods of the interface in the sub class otherwise we have declare the subclass as “abstract”

```
//Test.java
interface IA{
    public abstract void method1();
}
abstract class SubB implements IA
{
}
```

Shashi Kumar - SSSIT

Example:

```
//Test.java
interface IA{
        public abstract void method1();
}
class SubB implements IA
{
        @Override
        public void method1()
        {}
}
```

Note:

- **Creating an object for an interface is nothing but creating an object for any of it's implemented class**

Example:

```
//Test.java
interface IA{
        public abstract void method1();
}

class SubB implements IA
{
    @Override
        public void method1()
        { System.out.println("OR m1 of IA"); }
}
```

Shashi Kumar - SSSIT

```
class IntEx1{
        public static void main(String args[]){
                IA ia=new SubB();
                    ia.method1();
        }
}
```

Example 2:

```
interface IA
{  void method1(); }  //public abstract

interface IB
{ void method2(); } //public abstract

class SubB implements IA,IB //M.Inher
{
        @Override
        public void method1()
        { System.out.println("OR m1 of IA "); }

        @Override
        public void method2()
        { System.out.println("OR m2 of IB "); }
}

class IntEx2
{  public static void main(String args[])
        {
                IA ia=new SubB( );
                ia.method1();
                        // ia.method2(); CE
```

Shashi Kumar - SSSIT

```
            IB ib=new SubB();
               ib.method2();
                  //ib.method1(); CE
        }
}
```

Default Method In JDK1.8 Interfaces :

Note: Upto JDK 1.7 the interface will have only abstract methods From JDK1.8 onwards they added default methods in the interface

Assume the an interface with one abstract method, Which is implemented in the its subclass, by chance if we add any abstract methods in the interface then it will leads an error at implementation classes.

In order to provide Some new functionalities in the implemented classes with out disturbing the implemented classes then they added new feature is called "default methods in interfaces"

Shashi Kumar - SSSIT

Eg:

```
interface IntA
{  public abstract void method1(); }

class SubB implements IntA
{
  @Override
  public void method1()
  { System.out.println("Overloaded method1 of intA"); }

  public static void main(String args[ ])
   { IntA ia=new SubB();
         Ia.method1();
   }
}
```

It will compile and execute successfully, By chance if we add any abstract methods in interface IntA then it will fail @ compilation time. To Overcome this limitation we have to use "default methods".

Eg:

Eg:

```
interface IntA
{  public abstract void method1();
  default void  method2()
  { System.out.println("Def method "); }
}
```

Shashi Kumar - SSSIT

```
class SubB implements IntA
{
  @Override
  public void method1()
  { System.out.println("Overloaded method1 of intA"); }

  public static void main(String args[ ])
   { IntA ia=new SubB();
         ia.method1();
         ia.method2();
   }
}
```

Note: In the above situation it is not compulsory to Override default method it is optional.

JDK1.9 private methods in Interfaces:

- **As we know that in JDK1.8 we have default methods in the interfaces,Some times while developing the real time projects we may required same logic for more than one default methods , Then that logic is recommended to define inside of the private methods, we can use it for multiple times, Thus we can avoid repetition of code , code optimization , memory optimization and efficiency of the code will be increase.**

- **Private methods we can use only inside of that interface but not outside of the interfaces.**

Shashi Kumar - SSSIT

Example:

```
interface IntA
{
  default void method1()
  {
    [==================
     ==================] common logic

     XXXXXXXXXXXXXXXXXX
     XXXXXXXXXXXXXXXXXX
  }
 default void method2()
  {
  [==================
     ==================] common logic

     XXXXXXXXXXXXXXXXXX
     XXXXXXXXXXXXXXXXXX
  }

 public abstract void method3();.....
}
```

In the above example we have used the common logic in multiple default functions. In order to eliminate repetition of code then we have to use "private functions " in interface.

Shashi Kumar - SSSIT

```
interface IntA
{

private void commonlogic() //Some name
{
  [==================
     ==================] common logic
}
   default void method1()
   {
       commonlogic( );
       XXXXXXXXXXXXXXXXXX
       XXXXXXXXXXXXXXXXXX
   }
  default void method2()
   {
      commonlogic();
       XXXXXXXXXXXXXXXXXX
       XXXXXXXXXXXXXXXXXX
   }

 public abstract void method3();.....
}
```

Shashi Kumar - SSSIT

Example:

```
interface IA
{
        private void clogic()
        {
                System.out.println
                        ("Privat Mtd In Interface From JDK1.9");
        }
        default void method2() //JDK1.8
          {
                clogic();
                System.out.println("New Fea From JDK1.8");
          }
        abstract void method1();

}

class SubB implements IA
{
        @Override
        public void method1()
        {System.out.println("OR mtd-1 of IA"); }
}

class Testing3
{       public static void main(String args[ ])
        {
                IA a=new SubB();
                   a.method1();
```

Shashi Kumar - SSSIT

```
                a.method2();
        }
}
```

Shashi Kumar - SSSIT