Abstraction

* It is the process of hiding the internal implementation and showing the necessary data to the user, is called abstraction.

Eg:- Sending messages, we just type the text and press on the send button. We don't know the internal processing, how it is being send.

* In java, abstraction can be achieved in two ways-

i) Using abstract class

ii) Using interfaces

* Abstract class -

If 'abstract' keyword is used before the class then it is called as abstract class.

→ If nothing is written before the class then it is called concrete class (Normal class that we write)

X→ An abstract class will always have atleast one abstract method.X

* Abstract method-

A method which is not having a body is known as Abstract method. and the method must be declared as abstract.

* An abstract class can have abstract and non-abstract method.

Eg:-
```
abstract class super //abstract class
{
    super()
    {
        S.o.P(" Super");
    }
    void meth1()     // Normal method
    {
        S.o.P (" Method1");
    }
    abstract void meth2(); // abstract method
}
Class sub extends super
{
    void meth2()  //Override the undefined method
    {
        S.o.P(" Method 2");
    }
}
Public class Test
{
    Public static void main (String a[])
    {
        super s1; // Reference of abstract class is
                                        allowd.
        Sub s2 = new sub();
    }
}
```

Note:

* If anyother class inherits abstract class then that class also becomes abstract class but to become a concrete class, the subclass must overrides the all undefined method.

* Do's and Don't's of Abstract class -

→ An abstract class can not be final because, if it is made final then it cannot be extended whereas abstract class is ment of inheritance.

→ An abstract method can not be final because if it made 'final' then it cannot be overridden whereas abstract method is ment for overriding.

→ Abstract class and method can neither be final nor static.

→ A sub class must override an abstract method or else it will become abstract class.

* Abstract class is used for achieving polymorphism as well as inheritance.

* A class is abstract class if at least one of the method is abstract.

* We cannot create an obj of abstract class because it contains abstract method and it doesn't have any body to execute.

* Reference of abstract class ~~is~~ can be created.

```
Eg:- super s1 = new super() // Not allowed

     Super s1;  // allowed
```

Eg:-
```
abstract class AbClass
{
    AbClass()
    {
        S.o.p(" AbClass : constructor called");
    }
    abstract class myFun();
}

class Sample extends AbClass
{
    Sample ()
    {
        S.o.p("Sample : constructor called");
    }
    void myFun ()
    {
        S.o.p ("MyFun() called");
    }
}

Public class Test
{
    public static void main (String [] arg)
    {
        Sample sm = new Sample()
            sm. myFun();
    }
}
```

O/P-
```
AbClass: constructor called
Sample: Constructor called
MyFun() called
```

Interface

- An interface is a collection of abstract methods and constants. but without any implementation

- It is a way to achieve abstraction, as it allows the programmer to focus on the behaviour of an object rather than its implementation.

- An ~~inf~~ interface has to be represented with 'interface' keyword.

  Syntax :

```
interface    interfaceName
{
        // Body of interface
}
```

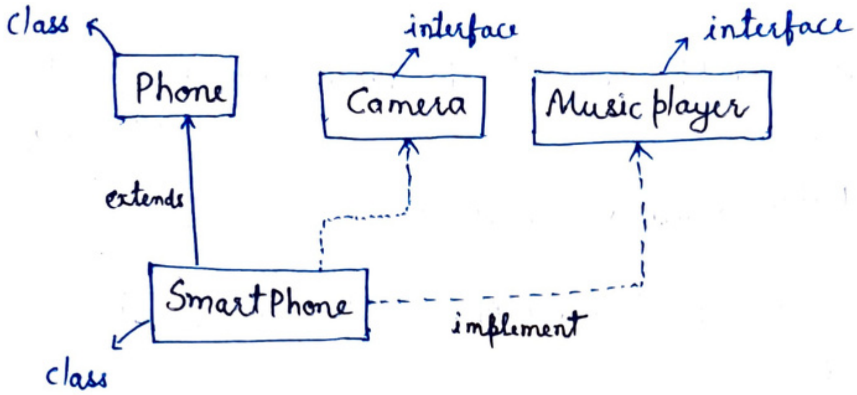
- All the methods of interface are by default public and abstract whether we write or don't write.

- In interface, we can not create an object of interface because all methods are by default abstract.

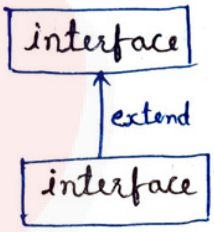
- But we can create a reference of interface ~~variable~~ and can be assigned the object of that class which is implemented.

- A class can extend from only one class at a time

- But a class can implement multiple interface at a time.



* An interface can be extented from another interface.



Example

```
class Phone
{
   Public void call ()
   {
      S.o.P(" Phone call");
   }
   Public void sms ()
   {
      S. o.P(" Phone sending sms");
   }
}
interface ICamera
 {
    void click();
    void record ();
    void videoCall ();
 }
interface IMusicPlayer
 {
    void play ();
    void stop ();
 ,
class smartPhone extends Phone implements ICamera,
                                           IMusicPlayer
{
   Public void videoCall()
   {
      S.o.p ("Smart Phone video calling");
   }
   Public void click()
   {
      S.o.p("Smart Phone clicking Photo");
   }
   Public void record()
   {
      S.o.P(" SmartPhone recording video");
   }
   Public void play()
   {
      S.o.P("SMart Phone playing video");
   }
   Public void stop()
   {
      S.o.P("Smart Phone stopped playing music");
   }
}
Public class Test
{
     public static void main( String a[])
      {
          IMusicPlayer SP = new SMartPhone();
             SP. play();
             SP. stop();          Reference of
                                     interface
      }
}
```

* Inheritance in interface

An interface can inherit from one or more interfaces using the 'extends' keyword, and the sub-interface can then use the methods of the parent interfaces.

```
interface A
{
  void showA();
}
interface B extends A
{
    void showB();
}
class InterfaceDemo implements B
{
    pubic void showA()
    {
        S.o.P("Method of interface 'A');
    }
    public void show B()
    {
        S.o.P(" Method of interface 'B');
    }
 }
Public class Test
{
   Public static void main(String[] args)
   {
       InterfaceDemo d = new InterfaceDemo();
          d.showA();
          d.showB();
   }
}
```