When to go interface, abstract class and concrete class?
interface:: It is prefered when we speak only about specification(no implementation).
abstract class:: It is prefered when we speak about partial implementation.
concreate class:: It is prefered when we speak about complete implementation and ready to provide service then we go for concrete class.

Difference b/w interface and abstract class?
Interface:: If we dont know anything about implementation just we have requirement specification then we should go for interface.
Abstract class: If we are talking about implementation but not completely then we should go for abstract class.

Interface:: Every method present inside the interface is always public and abstract whether we are declaring or not.
Abstract :: Every method present inside abstract class need not be public and abstract.

Interface:: We can't declare interface methods with the modifiers like private,protected,final,static,synchronized,native, strictfp.
Abstract :: There are not restrictions on abstract class method modifiers.

Interface:: Every interface variable is always public static final whether we are declaring or not.
Abstract:: Every abstract class variable need not be public static final.

Interface:: Every interface variable is always public static final we can't declare with the following modifiers like private,protected,transient,volatile.
Abstract:: No restriction on access modifiers

Interface:: For every interface variable compulsorily we should perform initialisation at the time of declaration, otherwise we get compile time error.
Abstract:: Not required to perform initialisation for abstract class variables at the time of declaration.

Interface:: Inside interface we can't write static and instance block.
Abstract :: Inside abstract class we can write static and instance block.

Interface:: Inside interface we can't write constructor.
Abstract :: Inside abstract class we can write constructor.

Note:
static block   => .class file loading happends and used to initialize static variables.
instance block => during the creation of an object,just before the constructor call used for initialization instance variable.
constructor   => during the creation of an object, used for initialization instance variable.

What is the need of abstract class? can we create an object of abstract class, Does it contains constructor?
=> abstract class object cannot be created becoz it is abstract.

```
   => But constructor is need for constructor to initialize the object.

eg::
class Parent {
      Parent() {
            System.out.println(this.hashCode());
      }
}
class Child extends Parent {
      public Child() {
            System.out.println(this.hashCode());
      }
}
public class TestApp {
      public static void main(String[] args) {
            Child c = new Child();
            System.out.println(c.hashCode());
      }
}
```

Why abstract class can contain constructor where as interface doesnot contain constructor?

```
   abstract class =>constructor  it is used to perform initialization of the
object.
                                  it is used to provide the value for the instance
variable.
                                  it is used to contain instance variable which are
requried for child
                                  object to perform intialisation for those
instance variables.

    interface => every variable is always static,public and final their is no
chance of existing instance variable inside the class.
                      so we should perform initialisation at the time of
declaration.
                      so constructor is not required for interface.
```

```
eg#1.
abstract class Person{
      String name;
      int age;
      int height;
      int weight;

      Person(String name,int age,int height,int weight){
            super();
            this.name=name;
            this.age=age;
            this.height=height;
            this.weight weight;
      }
}

class Student extends Person{

      int rollno;
      int marks;

        Student(String name,int age,int height,int weight,int rollno,int marks){
            super(name,age,height,weight,rollno);
            this.rollno=rollno;
            this.marks=marks;
      }
}
```

Question1:
   Can reference be created for abstract class?

```
         Person p  =new Student("sachin",49,5.6f,71,10,100);
```

   Can reference be created for interface?

```
         ISample sample = null;
```

Note::Every method present inside the interface is abstract, but in abstract class also we take only abstract methods then
           what is the need of interface concept?

```
abstract class Sample{
      public abstract void m1();
      public abstract void m2();
}

interface ISample{
       void m1();
       void m2();
}
```

=> we can replace interface concept with abstract class, but it is not a good programming practise.

eg#1

```
interface X{
      ...
      ...
}
class Test implements X{
      ...
        ...
}
  Test t=new Test();
```

i. performance is high.
ii.While implementing X we can extends
   one more class,through which we can bring reusablity.

eg#2.

```
  abstract X{
      ...
      ...
  }
  class Test extends X{
        ...
        ...
  }
    Test t=new Test();
```

i.performance is low.
ii.While extending X we can't extends
   any other classes so reusablity is not brought.

```
Note: If everything is abstract then it is recommended to go for interface.

Wrapper class
============
Purpose
 1. To wrap primitives into object form so that we can handle primitives also just
like objects.
 2. To define several utility functions which are required for the primitives.

Constructors
===========
 Almost all the Wrapper class have 2 constructors
   a. one taking primitive type.
   b. one taking String type.

eg: Integer i=new Integer(10);
      Integer i=new Integer("10");

    Double d=new Double(10.5);
    Double d=new Double("10.5");

Note: If String argument is not properly defined then it would result in
RunTimeException called
                "NumberformatException".
      eg:: Integer i=new Integer("ten");//RE:NumberFormatException

Wrapper class and its associated constructor
   Byte   => byte and String
   Short  => short and String
  Integer => int and String
  Long    => long and String
  **Float => float ,String and double
   Double => double and String
 **Character=> character
 ***Boolean  => boolean and String

eg::
 1) Float f=new Float (10.5f);
 2) Float f=new Float ("10.5f");
 3) Float f=new Float(10.5);
 4) Float f=new Float ("10.5");

eg::
 1) Charcter c=new Character('a');
 2) Character c=new Character("a"); //invalid

eg::
Boolean b=new Boolean(true);
Boolean b=new Boolean(false);
Boolean b1=new Boolean(True);//C.E
Boolean b=new Boolean(False);//C.E
Boolean b=new Boolean(TRUE);//C.E

eg::
 Boolean b1=new Boolean("true");
 Boolean b2=new Boolean("True");
 Boolean b3=new Boolean("false");
 Boolean b4=new Boolean("False");
 Boolean b5=new Boolean("nitin");
 Boolean b6=new Boolean("TRUE");
 System.out.println(b1);//true
 System.out.println(b2);//true
 System.out.println(b3);//false
 System.out.println(b4);//false
 System.out.println(b5);//false
 System.out.println(b6);//true


eg::
class Test
{
      public static void main(String[] args)
      {
            Boolean b1 =new Boolean("yes");//false
            Boolean b2 =new Boolean("no");//false
            System.out.println(b1);
            System.out.println(b2);

            System.out.println(b1.equals(b2));//false.equals(false)-> true
            System.out.println(b1 == b2);//false


            Integer i1 = new Integer(10);
            Integer i2 = new Integer(10);
            System.out.println(i1);//10
            System.out.println(i2);//10
            System.out.println(i1.equals(i2));//true


      }
}
```
Note: In case of Boolean constructor, boolean value be treated as true w.r.t to case insensitive
part of "true",for all others it would be treated as "false".

Note:
If we are passing String argument then case is not important and content is not important.
If the content is case insensitive String of true then it is treated as true in all other cases it is treated as false.

Note: In case of Wrapper class,toString() is overriden to print the data.
In case of Wrapper class,equals() is overriden to check the content.
Just like String class, Wrapper classes are also treated as "Immutable class".

Immutable class
=============
If we create an Object and if we try to make a change, with that change new object will be created and those changes
will not reflected in the old copy.

Can we make our userdefined class as Immutable?
ans. yes possible as shown below

```
final class Test
```

```
{
      int i;
      Test(int i){
            this.i = i;
      }
      public Test modify(int i){
                  if (this.i == i)
                        return this;
                  else
                        return new Test(i);

      }
      public static void main(String[] args)
      {
            Test t1 = new Test(10);
            Test t2 = t1.modify(10);
            Test t3 = t1.modify(100);
            Test t4 = t3.modify(100);

            System.out.println(t1==t2);//true
            System.out.println(t1==t3);//false
            System.out.println(t2==t3);//false
            System.out.println(t3==t4);//true

      }
}
```