Wrapper class utiltiy methods
==========
1. valueOf() method.
2. XXXValue() method.
3. parseXxx() method.
4. toString() method.

```
  public static wrapper valueOf(String data, int radix) throws
java.lang.NumberFormatException;
  public static wrapper valueOf(String data) throws
java.lang.NumberFormatException;
  public static wrapper valueOf(int data);
```

valueOf() method
==========
 To create a wrapper object from primitive type or String we use valueOf().
 It is alternative to constructor of Wrapper class, not suggestable to use.
 Every Wrapper class,except character class contain static valueOf() to create a Wrapper Object.

eg#1.
```
Integer i=Integer.valueOf("10");
Double  d=Double.valueOf("10.5");
Boolean b=Boolean.valueOf("nitin");
   System.out.println(i);
   System.out.println(d);
   System.out.println(b);
```

eg#2.
```
     public static valueOf(String s,int radix)
                                |=> binary : 2(0,1)
                                |=> octal  : 8(0-7)
                                |=> decimal : 10(0-9)
                                |=> hexadecimal : 16(0-9,a,b,c,d,e,f)
                                |=> base : 36(0-9,a-z)

Integer i1=Integer.valueOf("1111");
    System.out.println(i1);//1111
Integer i2=Integer.valueOf("1111",2);
    System.out.println(i2);//15
Integer i3=Integer.valueOf("ten");
    System.out.println(i3);//RE:NumberFormatException
Integer i4=Integer.valueOf("1111",37);
    System.out.println(i4);//RE:NumberFormatException
```

eg#3.
```
    public static valueOf(primitivetype x)

Integer i1=Integer.valueOf(10);
Double  d1=Double.valueOf(10.5);
Character c=Character.valueOf('a');
Boolean b=Boolean.valueOf(true);
```
   Primtive/String =>valueOf() => WrapperObject

2. xxxValue()
     We can use xxxValue() to get primitive type for the given Wrapper Object.
     These methods are a part of every Number type Object.
     (Byte,Short,Integer,Long,Float,Double) all these classes have these 6 methods which is

Written as shown below.

```
Methods
=======
  public byte byteValue();
  public short shortValue();
  public int intValue();
  public long longValue();
  public float floatValue();
  public double doubleValue();

eg#1.
 Integer i=new Integer(130);

// result = minrange +(total -maxrange -1)
 System.out.println(i.byteValue());//-126
 System.out.println(i.shortValue());//130
 System.out.println(i.intValue());//130
 System.out.println(i.longValue());//130
 System.out.println(i.floatValue());//130.0
 System.out.println(i.doubleValue());//130.0

3. charValue()
       Character class contains charValue() to get Char primitive for the given
Character
       Object.
      public char charValue()
eg#1.
  Character c=new Character('c');
  char ch= c.charValue();
  System.out.println(ch);

4. booleanValue()
         Boolean class contains booleanValue() to get boolean primitive for the
given boolean
         Object.
      public boolean booleanValue()

eg#1.
   Boolean b=new Boolean("nitin");
   boolean b1=b.booleanValue();
    System.out.println(b1);//false

In total xxxValue() are 36 in number.
 => xxxValue() => convert the Wrapper Object => primitive.

parseXXXX()
===========
 We use parseXXXX() to convert String object into primitive type.

form-1
======
public static primitive parseXXX(String s)
  Every wrapper class,except Character class has parseXXX() to convert String into
primitive type.

eg: int i=Integer.parseInt("10");
    double d =Double.parseInt("10.5");
    boolean b=Boolean.parseBoolean("true");
```

usage of Wrapper class in realtime coding
==========

```
//WAP to take inputs from the command line and perform arithemetic operations
class Test
{
      public static void main(String[] args)
      {
            //valueOf()     => Converts String/Primitive to Wrapper type
            //xxxValue()   =>  Converts Wrapper type to Primitive type
            //parseXXX()    => converts String to primitive type

            //commandline arguments => String inputs = args[0],args[1]

            int i1 = Integer.parseInt(args[0]);
            int i2 = Integer.parseInt(args[1]);

            System.out.println(i1+i2);
            System.out.println(i1-i2);
            System.out.println(i1*i2);
            System.out.println(i1/i2);

            //args -> String, convert into primitive type and process

      }
}
```

form-2
======

```
public static primitive parseXXXX(String s, int radix)
                                        |=> range is from 2 to 36
```

Every Integral type Wrapper class(Byte,Short,Integer,Long) contains the following parseXXXX()
to convert Specified radix String to primitive type.

```
eg: int i=Integer.parseInt("1111",2);
    System.out.println(i);//15
```

Note: String => parseXXX() => primitive type

toString()
==========

To convert the Wrapper Object or primitive to String.
Every Wrapper class contain toString()

form1
=====

```
public String toString()
```

1. Every wrapper class (including Character class) contains the above toString() method to convert wrapper object to String.
2. It is the overriding version of Object class toString() method.
3. Whenever we are trying to print wrapper object reference internally this toString() method only executed

```
eg:  Integer i=Integer.valueOf("10");
     System.out.println(i);//internally it calls toString() and prints the Data.
```

form2
=====
   public static String toString(primitivetype)

1. Every wrapper class contains a static toString() method to convert primitive to String.

String s=Integer.toString(10);
                          |=> primitive type int.

eg:
 String s=Integer.toString(10);
 String s=Boolean.toString(true);
 String s=Character.toString('a');

form3
=====
Integer and Long classes contains the following static toString() method to convert the
primitive to specified radix String form.

 public static String toString(primitive p,int radix)
                                    |=> 2 to 36

eg: String s=Integer.toString(15,2)
    System.out.println(s); // 1111

form4
=====
Integer and Long classes contains the following toXxxString() methods.
public static String toBinaryString(primitive p);
public static String toOctalString(primitive p);
public static String toHexString(primitive p);

Example:
```
class WrapperClassDemo {
   public static void main(String[] args) {
      String s1=Integer.toBinaryString(7);
      String s2=Integer.toOctalString(10);
      String s3=Integer.toHexString(20);
      String s4=Integer.toHexString(10);

      System.out.println(s1);//111
      System.out.println(s2);//12
      System.out.println(s3);//14
      System.out.println(s4);//a
  }
}
```
Note:
String class
```
  public static  String valueOf(boolean);
  public static  String valueOf(char);
  public static  String valueOf(int);
  public static  String valueOf(long);
  public static  String valueOf(float);
  public static  String valueOf(double);
```

```
String data = String.valueOf('a');//static factory methods
String data = "sachin".toUpperCase();//instance factory methods


AutoBoxing and AutoUnBoxing
=========================
untill 1.4Version, we can't provide wrapper class objects in place of primitive and
primitive in place of wrapper object all
the required conversions should be done by the programmer.
But from jdk1.5 Version onwards,we can provide primtive in place of wrapper and in
place of wrapper we can keep primitive
also.All the requried conversion will be done by the compiler automatically, this
mechanism is called as "AutoBoxing" and
"AutoUnBoxing".


eg#1.
Boolean b1 = Boolean.valueOf(true);
if (b1)
   System.out.println("hello");

eg#2.
ArrayList al = new ArrayList();
  al.add(10);

Autoboxing
=========
  Automatic conversion of primtive type to wrapper object by the compiler is called
"AutoBoxing".
      Integer i1 = 10;
                |
             |After compilation the code would be
             |
             |
  Integer i1 = Integer.valueOf(10);

Note: Autoboxing is done by the compiler using a method called "valueOf()".

AutoUnBoxing
============
   Automatic conversion of wrapper object to primtive type by compiler is called
"AutoUnBoxing".

      Integer i1 = new Integer(10);
        int i2 = i1;
            |
            |compiler converts Integer to int type using intValue()
            |
       int i2 = i1.intValue();

Note: AutoUnboxing is done by the compiler using a method called "xxxValue()"

Case1:
=====
class Test
{
      static Integer i1 = 10;// AutoBoxing
      public static void main(String[] args)
```

```
	{
		int i2 = i1;//AutoUnBoxing
		m1(i2);
	}
	public static void m1(Integer i2){//AutoBoxing
		int k = i2;//AutoUnBoxing
		System.out.println(k);//10
	}
}
```

Compiler is responsible for conversion of primitive to wrapper and wrapper to primitive using the concept of
"AutoBoxing and AutoUnBoxing".

case2:

```
class Test
{
	static Integer i1;//i1 = null
	public static void main(String[] args)
	{
		int i2 = i1;// int i2 = i1.intValue() :: NullPointerException
		System.out.println(i2);
	}
}
```

Case3 :

```
Integer i1 = 10;//AutoBoxing
Integer i2 = i1;
i1++; = > i1 = i1+1

System.out.println(i1);
System.out.println(i2);
System.out.println(i1==i2);
```

Case4:

```
Integer x = new Integer(10);
Integer y = new Integer(10);
System.out.println(x == y);//false
```

Case5:

```
Integer x = new Integer(10);//memory from heap area
Integer y = 10;//AutoBoxing ===> Integer y = Integer.valueOf(10);
System.out.println(x == y);//false
```

Case6:

```
Integer x = new Integer(10);
Integer y = x; ===> reference is reused so pointing to same object
System.out.println(x == y);//true
```

Case7:

```
Integer x = 10;
Integer y = 10;
System.out.println(x == y);

Integer a = 100;
Integer b = 100;
System.out.println(a == b);

Integer i = 1000;
Integer j = 1000;
```

```
System.out.println(i == j);
```

Note:
1. To implement autoboxing concpet in wrapper class a buffer of object will be created at the time of class loading.
2. During AutoBoxing,if an object has to be created first jvm will check whether the object is already available inside buffer
    or not.
3. If it is available, then JVM will reuse the buffered object instead of creating a new Object.
4. If the Object is not available inside buffer, then jvm will create a new object in the heap area, this approach improves the
    performance and memory utilization

But this buffer concept is applicable only for few cases
1. Byte   => -128 to +127
2. Short   => -128 to +127
3. Integer=> -128 to +127
4. Long   => -128 to +127
5. Character => 0 to 127
6. Boolean   => true,false
In the remaining cases new object will be created.

```
// String/primtive to wrapper   => valueOf()
// Wrapper type    to primitive => xxxValue()

class Test
{
      public static void main(String[] args)
      {
            Integer x = 128;
            Integer y = 128;
            System.out.println(x == y);//false

            Integer a = 127;
            Integer b = 127;
            System.out.println(a == b);//true

            Boolean b1 = true;
            Boolean b2 = true;
            System.out.println(b1==b2);//true

            Double d1 = 10.0;
            Double d2 = 10.0;
            System.out.println(d1==d2);//false


      }
}
```