

Wrapper class

=====

The need of wrapper class is to wrap primitives into objects, so that we can handle primitives also just like Objects.

Constructor summary

=====

byte -> Byte(byte, String)
short -> Short(short, String)
int -> Integer(int, String)
long -> Long(long, String)
float -> Float(float, String, Double)
double -> Double(double, String)

char -> Character(char)
boolean -> Boolean(boolean, String)

Note: In all wrapper class toString() is overridden to return the content

Object class method

=====

```
public class Object{
    public String toString(){
        //return the reference of the Object
    }
    public boolean equals(Object o){
        //compares the reference
    }
}
```

```
public final class String{

    @Override
    public String toString(){
        //returns the content of the Object
    }

    @Override
    public boolean equals(Object o){
        //compares the content of the String
    }
}
```

Note: equals() method is also overridden in all Wrapper class to compare the content.

```
public final class Integer{

    @Override
    public String toString(){
        //returns the content of the Object
    }

    @Override
    public boolean equals(Object o){
        //compares the content.
    }
}
```

```
eg#1.
Integer i1=new Integer(10);
Integer i2=new Integer(10);
System.out.println(i1); //i1.toString() -> 10
System.out.println(i1.equals(i2));//true
```

Setter methods and Getter Methods

```
=====
public void setXXXX(XXXX data){

}
public XXXX getXXXX(){

}
```

Usage of Wrapper class

utility method(helper methods/static methods)

```
1.valueOf()
2.XXXValue()
3.parseXXX()
4.toString()
```

1.valueOf()

```
signature: public static wrapper valueOf(primitive data)
           public static wrapper valueOf(String data)
```

It is used to create wrapper object for the given primitive or String type of data.

It is alternative to constructor, but good practise is to use valueOf() only.

eg#1.

//constructor usage of Wrapper class to create Wrapper Object

```
Integer i1= new Integer(10);
Integer i2= new Integer("10");
```

//usage of utility methods to create Wrapper Object

```
Integer i3= Integer.valueOf(10);
Integer i4= Integer.valueOf("10");
```

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

```
System.out.println();
```

```
System.out.println(i3);
System.out.println(i4);
```

output

10

10

10

10

Note: valueOf() is also a part of Character class.

eg#2.

```
Integer i1=Integer.valueOf(10);
```

```
Double d1= Double.valueOf(10.5);
Boolean b1=Boolean.valueOf("Nitin");
Character c1=Character.valueOf('a');
```

```
System.out.println(i1);//10
System.out.println(d1);//10.5
System.out.println(b1);//false
System.out.println(c1);//a
```

2.xxxxValue()

We can use xxxxValue() to convert wrapper to primitive type.

Every Number type wrapper class(Byte,Short,Integer,Long,Float,Double) contains the following 6 xxxxValue() method to convert the wrapper object to primitive type.

Number

- Byte
- Short
- Integer
- Long
- Float
- Double

- Character
- Boolean

eg#1.

```
Integer i=new Integer(130);
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
```

Note: 130 is not in the range of byte so jvm will perform operation in the following manner

```
range = -128 to 127
result = -128, -127, -126
last value will be stored.
```

eg#2.

```
Character c1=new Character('c');
char c2= c1.charValue();
System.out.println(c2);//c

Boolean b1=new Boolean("nitin");
Boolean b2=b1.booleanValue();
System.out.println(b2);//false
```

3.parseXXX()

Every wrapper class except Character class Contains parseXXXX() to convert String to

Corresponding primitive type.

signature: public static xxxx parseXXX(String data)

eg#1.

```
int i1= Integer.parseInt("10");
```

```

System.out.println(i1);//10

boolean b1=Boolean.parseBoolean("TrUE");
System.out.println(b1);//true

short s1=Short.parseShort("Ten");
System.out.println(s1);//NumberFormatException

```

4. toString()

We can use toString() to convert wrapper object/primitive data to String.

```

signature:    public static String toString(XXXX data)
               public static String toString(xxxx data)

```

eg#1.

```

Integer i=Integer.valueOf("10");
System.out.println(i);//10(in String format)
System.out.println(i.toString());//10(in String format)

System.out.println();

```

```

String s1=Integer.toString(10);
String s2= Boolean.toString(true);
String s3= Character.toString('a');

```

```

System.out.println(s1);//10(in string format)
System.out.println(s2);//true(in string format)
System.out.println(s3);//a(in string format)

```

refer :diagram for conversion chart

AutoBoxing and AutoUnBoxing(JDK1.5V)

=====

```

valueOf()      -> To convert String/primitive to Wrapper Object
xxxxValue()    -> To convert Wrapper to primitive type.

```

```

Integer i = 10;

```

```

|
|compiler will make the following change
|

```

```

Integer i = Integer.valueOf(10);

```

Automatic conversion of primitive type to wrapper type done by the compiler is called "AutoBoxing".

```

Integer i1= new Integer(10);
int i2 = i1;

```

```

|
|Compiler will do the following change
|

```

```

int i2= i1.intValue();
Automatic conversion of wrapper type to primitive type done by the compiler is
called "AutoUnBoxing".

```

Autoboxing and UnBoxing

=====

```
eg#1.
class TestApp
{
    static Integer I=10;//AutoBoxing(valueOf())
    public static void main(String[] args)
    {
        int i=I;//AutoUnBoxing(intValue())
        System.out.println(i);//10
    }
}
```

```
eg#2.
class TestApp
{
    static Integer I=0;//AutoBoxing(valueOf())
    public static void main(String[] args)
    {
        int i=I;//AutoUnBoxing(intValue())
        System.out.println(i);//0
    }
}
```

```
eg#3.
class TestApp
{
    static Integer I=null;//AutoBoxing(valueOf())
    public static void main(String[] args)
    {
        int i=I;//AutoUnBoxing(intValue())//NullPointerException
        System.out.println(i);//
    }
}
```

- a. null
- b. 0
- c. CE
- d. NumberFormatException
- e. NullPointerException
- f. None of the above

Note:

Immutable Object -> String, all wrapper classes
(if we try to make a change, then with the change new object will be created)

```
Integer x=10;
Integer y=x;
x++;
System.out.println(x);//11
System.out.println(y);//10
System.out.println(x==y);//false
```

Snippets

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

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

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

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

```
Integer x=100;
Integer y=100;
System.out.println(x==y);//true
```

```
Integer x=1000;
Integer y=1000;
System.out.println(x==y);//false
```

Note:

byte, short, int, long, float, double the buffer concept which internally jvm maintains is "byte range only".

character -> 0 to 127

Boolean -> always(true or false)

