```
Wrapper class
==========
      1. To wrap primtive into object form so that we can handle primtive also just
like objects
      2. To define several utility function which are requried for primitives.
      3. Wrapper classes are a part of "java.lang" package.

primtive data types
==============
  1. byte,short,int,long
  2. float,double
  3. char
  4. boolean

For every primitive type we have equavilent Wrapper class as shown below
      byte -> Byte
      short -> Short
      int    -> Integer
      long -> Long
      float -> Float
      double-> Double
      ***char  -> Character(1 constructor)
      ***boolean -> Boolean(2 constructor(String is important))

With Respect to wrapper class how is toString() implemented?

class Object{
      public String toString(){
            // returns the reference(address/hashCodeValue) of the object
      }
}
public final class Integer extends Object{

      @Override
      public String toString(){
            //returns the data present in the Object
      }
}
```

Almost every Wrapper class contains 2 constructors which takes
            a. primtive type as the argument.
            b. String type as the argument.

eg#1.
```
Integer i1 = new Integer(10);
System.out.println(i1);//jvm calls i1.toString()
Integer i2 = new Integer("10");
System.out.println(i2);//jvm calls i1.toString()
```

output
```
10
10
```

eg#2
 If the String input is not properlyformatted,mean if it is not reprsenting any number then we will get an Exception called
 "NumberFormatException"
```
   Integer i2 = new Integer("ten");//NumberFormatException
```

eg#3.
      Character class contains only constructor which can take only primitive argument of type char only.
```
            Character c1=new Character('a');
            System.out.println(c1);

            Character c1=new Character("a");//Compile Time Error.
            System.out.println(c1);
```

eg#4.
```
Boolean b=new Boolean(true);
System.out.println(b);//true

Boolean b=new Boolean(false);
System.out.println(b);//false

Boolean b=new Boolean(True);//CE
Boolean b=new Boolean(False);//CE
```

Note: If we are passing String argument, then case is not important and content is important.
         if the content is case insensitive String of true then it is treated as true and in all other cases it is false.
eg#5
```
Boolean b1=new Boolean("false");
System.out.println(b1);//false

Boolean b2=new Boolean("False");
System.out.println(b2);//false
```

eg#6
```
Boolean b1=new Boolean("true");
System.out.println(b1);//true

Boolean b2=new Boolean("True");
System.out.println(b2);//true
```

```
eg#7.
Boolean b1=new Boolean("yes");
System.out.println(b1);//false

Boolean b2=new Boolean("no");
System.out.println(b2);//false

Boolean b1=new Boolean("tRuE");
System.out.println(b1);//true

Boolean b2=new Boolean("TrUe");
System.out.println(b2);//true

Object class methods
===============
public class java.lang.Object {
  public java.lang.Object();
  public final native java.lang.Class<?> getClass();
  public native int hashCode();
  public boolean equals(java.lang.Object);
  protected native java.lang.Object clone() throws
java.lang.CloneNotSupportedException;
  public java.lang.String toString();
  public final native void notify();
  public final native void notifyAll();
  public final native void wait(long) throws java.lang.InterruptedException;
  public final void wait(long, int) throws java.lang.InterruptedException;
  public final void wait() throws java.lang.InterruptedException;
  protected void finalize() throws java.lang.Throwable;
  static {};
}

String toString()
      JVM will always call toString() when we try to print any reference variable.
      reference varaible can be
            a. inbuilt class
            b. user defined class

eg#1.

class Object{
      public String toString(){
            // returns the reference(address/hashCodeValue) of the object
      }
}
public final class String extends Object{

      @Override
      public String toString(){
            //returns the data present in the Object
      }
}

String name= new String("sachin");
System.out.println(name);// jvm internally calls name.toString()

eg#2.

class Object{
        public String
         toString(){
        // returns the reference(address/hashCodeValue) of the object
      }
}
public class Student extends
      Object{ String name;

      Student(String
            name){
            this.name
            =name;
      }

      public String toString(){
        // returns the reference(address/hashCodeValue) of the object
      }
}

Student student = new Student("sachin");
   System.out.println(student);//JVM calls
   student.toString()

output: hashCode value of Student object


eg#3.
class Object{
      public String toString(){
            // returns the reference(address/hashCodeValue) of the object
      }
}
public class Student extends
      Object{ String name;

      Student(String
            name){
            this.name
            =name;
      }

      @Override
      public String
            toString(){
            return
            this.name;
      }
}

Student student = new Student("sachin");
   System.out.println(student);//JVM calls
   student.toString()

output: sachin
```