1. Need of Wrapper classes:

Java uses primitive types such as int or double, to hold the basic data types supported by the language. We can use the primitive types as an object using Wrapper classes in java. But every time using objects for these values would add an unacceptable overhead to even the simplest of calculations. Thus, the primitive types are not part of the object hierarchy, and they do not inherit Object.
Despite the performance benefit offered by the primitive types, there are times when you will need an object representation. For example, you can not pass a primitive type by reference to a method. Also, many of the standard data structures implemented by Java operate on objects, which means that you can not use these data structures to store primitive types. To handle these situations, Java provides type wrappers,which are classes that encapsulate a primitive type within an object. Wrapper class in java provides the mechanism to convert primitive into object and object into primitive.

2. Types of Wrapper classes:

There are total 8 Wrapper classes which falls into two types:

- **Non-numeric :** Character, and Boolean
- **Numeric :** Double,Float,Long,Integer,Short,Byte,

The wrapper classes resides in java.lang package.

The list of eight wrapper classes are given below:

Primitive Type	Wrapper class
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

2.1 Non-numeric type Wrapper classes :

1. Character :
Character is a wrapper around a char.

Constructor :

- **Character(char ch)**
  Here,ch specifies the character that will be wrapped by the Character object being created.

To obtain the char value contained in a Characterobject, call charValue( ), shown here:

char charValue( )

It returns the encapsulated character.

2. Boolean :

Boolean is a wrapper around boolean values. x

Constructors :

- **Boolean(boolean boolValue)**
  boolValue must be either true or false.

- **Boolean(String boolString)**
  boolString contains the string true ,then the new Boolean object will be true otherwise it will be false.

To obtain a boolean value from a Boolean object, use booleanValue( ), shown here:

boolean booleanValue( )

2.2 Numeric type Wrapper classes :

These are Byte,Short,Integer,Long,Float, and Double. All of the numeric type wrappers inherit the abstract class Number. Number declares methods that return the value of an object in each of the different number formats.

These methods are:

- **byte** byteValue( )
- **double** doubleValue( )
- **float** floatValue( )
- **int** intValue( )
- **long** longValue( )
- **short** shortValue( )

1. Double and Float

Double and Float are wrappers for floating-point values of type double and float, respectively.

Constructors for Float :

- Float(**double** num)
- Float(**float** fnum)
- Float(String str) **throws** NumberFormatException

Float objects can be constructed with values of type float or double. They can also be constructed from the string representation of a floating-point number.

Constructors for Double :

- Double(**double** num)
- Double(String str) **throws** NumberFormatException

2. Byte, Short, Integer and Long :
These are the Wrappers for byte, short, int and long respectively.

Constructors :

- Byte(**byte** num)
- Byte(String str) throws NumberFormatException
- Short(**short** num)
- Short(String str) throws NumberFormatException
- Integer(**int** num)
- Integer(String str) throws NumberFormatException
- Long(**long** num)
- Long(String str) throws NumberFormatException

Wrapper class Example:
1. Primitive to Wrapper

```
public class WrapperExample1
{
        public static void main(String args[]){
        //Converting int into Integer
                int a=20;
                Integer i=Integer.valueOf(a);//converting int into Integer
                Integer j=a;//autoboxing, now compiler will write Integer.valueOf(a) internally
                System.out.println(a+" "+i+" "+j);
        }
}
```

Output:
20 20 20

2. Wrapper to Primitive

```
public class WrapperExample2{
public static void main(String args[]){
//Converting Integer to int
Integer a=new Integer(3);
int i=a.intValue();//converting Integer to int
int j=a;//unboxing, now compiler will write a.intValue() internally
System.out.println(a+" "+i+" "+j);
}}
```

Output:
3 3 3