Wrapper Classes in Java

In Java, wrapper classes allow primitive data types (like int, char, boolean) to be used as objects. They provide a way to wrap primitive values into object types and unwrap them back into primitives.

Since Java 5 (J2SE 5.0), the introduction of autoboxing and unboxing automates this conversion process:

- **Autoboxing:** Automatically converts a primitive to its wrapper object.
- **Unboxing:** Automatically converts a wrapper object back to its primitive.

Advantages of Wrapper Classes

Wrapper classes offer several benefits over primitives:

- Used in Collections (which only support objects).
- Enable method usage like compareTo(), equals(), toString(), etc.
- Support cloning, serialization, and null values.
- Thread-safe and immutable, ideal for concurrent programming.
- Facilitate type conversion and utility operations.

Use of Wrapper Classes in Java

Java is an object-oriented language, meaning many features, such as collections, serialization, and synchronization, require working with objects rather than primitive types. Wrapper classes help bridge this gap by converting primitives into their corresponding objects.

1. Changing the Value in a Method

Java uses call-by-value, so when a primitive is passed to a method, the original value does not change. However, we can mimic pass-by-reference behavior using wrapper objects (which are passed as references).

2. Serialization

Serialization converts objects into byte streams. Since primitives cannot be directly serialized, wrapper classes are used to wrap them as objects.

3. Synchronization

Java synchronization works only with objects. When using thread synchronization, primitive types must be converted to wrapper objects.

4. Use in java.util Package

The java.util package provides utility classes like ArrayList, HashMap, and others that work only with objects. Hence, wrapper classes are required when storing primitives.

5. Java Collection Framework

All data structures in the Java Collections Framework (like ArrayList, HashSet, TreeSet, etc.) can store only objects, not primitives. Wrapper classes are essential for primitives to be used in these collections.

Wrapper Classes List

The eight classes of the java.lang package are known as wrapper classes in Java. The list of eight wrapper classes is given below:

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

Autoboxing

Autoboxing is the automatic conversion of a primitive type to its corresponding wrapper class.

Example: Primitive to Wrapper

```
public class WrapperExample1 {
    public static void main(String[] args) {
        int a = 20;
```

```
        Integer i = Integer.valueOf(a); // Manual boxing
        Integer j = a; // Autoboxing

        System.out.println(a + " " + i + " " + j);
    }
}
```

Output:

20 20 20

Unboxing

Unboxing is the automatic conversion of a wrapper object to its primitive type.

Example: Wrapper to Primitive

```
public class WrapperExample2 {
    public static void main(String[] args) {
        Integer a = new Integer(3);
        int i = a.intValue(); // Manual unboxing
        int j = a; // Unboxing
        System.out.println(a + " " + i + " " + j);
    }
}
```

Output:

3 3 3

Complete Example: All Types

```
public class WrapperExample3 {
    public static void main(String[] args) {
        byte b = 10;
        short s = 20;
        int i = 30;
        long l = 40L;
        float f = 50.0f;
        double d = 60.0;
        char c = 'a';
        boolean b2 = true;

        // Autoboxing
        Byte byteObj = b;
        Short shortObj = s;
        Integer intObj = i;
        Long longObj = l;
        Float floatObj = f;
        Double doubleObj = d;
        Character charObj = c;
        Boolean boolObj = b2;

        System.out.println("--- Printing Object Values ---");
```

```
System.out.println("Byte: " + byteObj);
System.out.println("Short: " + shortObj);
System.out.println("Integer: " + intObj);
System.out.println("Long: " + longObj);
System.out.println("Float: " + floatObj);
System.out.println("Double: " + doubleObj);
System.out.println("Character: " + charObj);
System.out.println("Boolean: " + boolObj);

// Unboxing
byte byteVal = byteObj;
short shortVal = shortObj;
int intVal = intObj;
long longVal = longObj;
float floatVal = floatObj;
double doubleVal = doubleObj;
char charVal = charObj;
boolean boolVal = boolObj;

System.out.println("--- Printing Primitive Values ---");
System.out.println("byte: " + byteVal);
System.out.println("short: " + shortVal);
System.out.println("int: " + intVal);
```

```
        System.out.println("long: " + longVal);
        System.out.println("float: " + floatVal);
        System.out.println("double: " + doubleVal);
        System.out.println("char: " + charVal);
        System.out.println("boolean: " + boolVal);
    }
}
```

Output:

```
--- Printing Object Values ---
Byte: 10
Short: 20
Integer: 30
Long: 40
Float: 50.0
Double: 60.0
Character: a
Boolean: true

--- Printing Primitive Values ---
byte: 10
short: 20
int: 30
long: 40
```

float: 50.0

double: 60.0

char: a

boolean: true