Wrapper classes in Java

The **wrapper class in Java** provides the mechanism *to convert primitive into object and object into primitive*.

Since J2SE 5.0, **autoboxing** and **unboxing** feature convert primitives into objects and objects into primitives automatically. The automatic conversion of primitive into an object is known as autoboxing and vice-versa unboxing.

Use of Wrapper classes in Java

Java is an object-oriented programming language, so we need to deal with objects many times like in Collections, Serialization, Synchronization, etc. Let us see the different scenarios, where we need to use the wrapper classes.

ADVERTISEMENT

* **Change the value in Method:** Java supports only call by value. So, if we pass a primitive value, it will not change the original value. But, if we convert the primitive value in an object, it will change the original value.
* **Serialization:** We need to convert the objects into streams to perform the serialization. If we have a primitive value, we can convert it in objects through the wrapper classes.
* **Synchronization:** Java synchronization works with objects in Multithreading.
* **java.util package:** The java.util package provides the utility classes to deal with objects.
* **Collection Framework:** Java collection framework works with objects only. All classes of the collection framework (ArrayList, LinkedList, Vector, HashSet, LinkedHashSet, TreeSet, PriorityQueue, ArrayDeque, etc.) deal with objects only.

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

|  |  |
| --- | --- |
| **Primitive Type** | **Wrapper class** |
| boolean | [Boolean](https://www.javatpoint.com/java-boolean) |
| char | [Character](https://www.javatpoint.com/post/java-character) |
| byte | [Byte](https://www.javatpoint.com/java-byte) |
| short | [Short](https://www.javatpoint.com/java-short) |
| int | [Integer](https://www.javatpoint.com/java-integer) |
| long | [Long](https://www.javatpoint.com/java-long) |
| float | [Float](https://www.javatpoint.com/java-float) |
| double | [Double](https://www.javatpoint.com/java-double) |

Autoboxing

The automatic conversion of primitive data type into its corresponding wrapper class is known as autoboxing, for example, byte to Byte, char to Character, int to Integer, long to Long, float to Float, boolean to Boolean, double to Double, and short to Short.

Since Java 5, we do not need to use the valueOf() method of wrapper classes to convert the primitive into objects.

**Wrapper class Example: Primitive to Wrapper**

1. //Java program to convert primitive into objects
2. //Autoboxing example of int to Integer
3. **public** **class** WrapperExample1{
4. **public** **static** **void** main(String args[]){
5. //Converting int into Integer
6. **int** a=20;
7. Integer i=Integer.valueOf(a);//converting int into Integer explicitly
8. Integer j=a;//autoboxing, now compiler will write Integer.valueOf(a) internally
10. System.out.println(a+" "+i+" "+j);
11. }}

Output:

20 20 20

Unboxing

The automatic conversion of wrapper type into its corresponding primitive type is known as unboxing. It is the reverse process of autoboxing. Since Java 5, we do not need to use the intValue() method of wrapper classes to convert the wrapper type into primitives.

**Wrapper class Example: Wrapper to Primitive**

ADVERTISEMENT

ADVERTISEMENT

1. //Java program to convert object into primitives
2. //Unboxing example of Integer to int
3. **public** **class** WrapperExample2{
4. **public** **static** **void** main(String args[]){
5. //Converting Integer to int
6. Integer a=**new** Integer(3);
7. **int** i=a.intValue();//converting Integer to int explicitly
8. **int** j=a;//unboxing, now compiler will write a.intValue() internally
10. System.out.println(a+" "+i+" "+j);
11. }}

Output:

3 3 3

Java Wrapper classes Example

1. //Java Program to convert all primitives into its corresponding
2. //wrapper objects and vice-versa
3. **public** **class** WrapperExample3{
4. **public** **static** **void** main(String args[]){
5. **byte** b=10;
6. **short** s=20;
7. **int** i=30;
8. **long** l=40;
9. **float** f=50.0F;
10. **double** d=60.0D;
11. **char** c='a';
12. **boolean** b2=**true**;
14. //Autoboxing: Converting primitives into objects
15. Byte byteobj=b;
16. Short shortobj=s;
17. Integer intobj=i;
18. Long longobj=l;
19. Float floatobj=f;
20. Double doubleobj=d;
21. Character charobj=c;
22. Boolean boolobj=b2;
24. //Printing objects
25. System.out.println("---Printing object values---");
26. System.out.println("Byte object: "+byteobj);
27. System.out.println("Short object: "+shortobj);
28. System.out.println("Integer object: "+intobj);
29. System.out.println("Long object: "+longobj);
30. System.out.println("Float object: "+floatobj);
31. System.out.println("Double object: "+doubleobj);
32. System.out.println("Character object: "+charobj);
33. System.out.println("Boolean object: "+boolobj);
35. //Unboxing: Converting Objects to Primitives
36. **byte** bytevalue=byteobj;
37. **short** shortvalue=shortobj;
38. **int** intvalue=intobj;
39. **long** longvalue=longobj;
40. **float** floatvalue=floatobj;
41. **double** doublevalue=doubleobj;
42. **char** charvalue=charobj;
43. **boolean** boolvalue=boolobj;
45. //Printing primitives
46. System.out.println("---Printing primitive values---");
47. System.out.println("byte value: "+bytevalue);
48. System.out.println("short value: "+shortvalue);
49. System.out.println("int value: "+intvalue);
50. System.out.println("long value: "+longvalue);
51. System.out.println("float value: "+floatvalue);
52. System.out.println("double value: "+doublevalue);
53. System.out.println("char value: "+charvalue);
54. System.out.println("boolean value: "+boolvalue);
55. }}

Output:

---Printing object values---

Byte object: 10

Short object: 20

Integer object: 30

Long object: 40

Float object: 50.0

Double object: 60.0

Character object: a

Boolean object: true

---Printing primitive values---

byte value: 10

short value: 20

int value: 30

long value: 40

float value: 50.0

double value: 60.0

char value: a

boolean value: true

# Utility Methods of Wrapper Classes in Java

**Last Updated :**09 Feb, 2022

**Prerequisite:** [Wrapper Classes](https://www.geeksforgeeks.org/wrapper-classes-java/)

The objective of the Wrapper class is to define several utility methods which are required for the primitive types. There are 4 utility methods for primitive type which is defined by the Wrapper class:

### 1. valueOf() method:

We can use the valueOf() method to create a Wrapper object for a given primitive or String. There are 3 types of valueOf() methods:

**A. Wrapper valueOf(String s):**Every wrapper class except Character class contains a static valueOf() method to create Wrapper class object for a given String.

**Syntax:**

public static Wrapper valueOf(String s);

* Java

|  |
| --- |
| // Java program to illustrate valueof() Method    class GFG {      public static void main(String[] args)      {          Integer I = Integer.valueOf("10");          System.out.println(I);            Double D = Double.valueOf("10.0");          System.out.println(D);            Boolean B = Boolean.valueOf("true");          System.out.println(B);            // Here we will get RuntimeException          Integer I1 = Integer.valueOf("ten");      }  } |

**Output:**

10

10.0

true

Exception in thread "main" java.lang.NumberFormatException: For input string: "ten"

**B. Wrapper valueOf(String s, int radix):** Every Integral Wrapper class Byte, Short, Integer, Long) contains the following valueOf() method to create a Wrapper object for the given String with specified radix. The range of the radix is 2 to 36.

**Syntax:**

public static Wrapper valueOf(String s, int radix)

* Java

|  |
| --- |
| // Java program to illustrate valueof() Method    class GFG {      public static void main(String[] args)      {          Integer I = Integer.valueOf("1111", 2);          System.out.println(I);            Integer I1 = Integer.valueOf("1111", 4);          System.out.println(I1);      }  } |

**Output**

15

85

**3. Wrapper valueOf(primitive p):** Every Wrapper class including the Character class contains the following method to create a Wrapper object for the given primitive type.

**Syntax:**

public static Wrapper valueOf(primitive p);

* Java

|  |
| --- |
| // Java program to illustrate valueof() Method    class GFG {      public static void main(String[] args)      {          Integer I = Integer.valueOf(10);          Double D = Double.valueOf(10.5);          Character C = Character.valueOf('a');            System.out.println(I);          System.out.println(D);          System.out.println(C);      }  } |

**Output**

10

10.5

a

### 2. xxxValue() Method

We can use xxxValue() methods to get the primitive for the given Wrapper Object. Every number type Wrapper class( Byte, Short, Integer, Long, Float, Double) contains the following 6 methods to get primitive for the given Wrapper object:

1. public byte byteValue()
2. public short shortValue()
3. public int intValue()
4. public long longValue()
5. public float floatValue()
6. public float doubleValue()

### 3. parseXxx() Method

We can use parseXxx() methods to convert String to primitive. There are two types of parseXxx() methods:

**A. primitive parseXxx(String s):** Every Wrapper class except the character class contains the following parseXxx() method to find primitive for the given String object.

**Syntax:**

public static primitive parseXxx(String s);

* Java

|  |
| --- |
| // Java program to illustrate parseXxx() Method    class GFG {      public static void main(String[] args)      {          int i = Integer.parseInt("10");          double d = Double.parseDouble("10.5");          boolean b = Boolean.parseBoolean("true");            System.out.println(i);          System.out.println(d);          System.out.println(b);      }  } |

**Output**

10

10.5

true

**B. parseXxx(String s, int radix):** Every Integral type Wrapper class (Byte, Short, Integer, Long) contains the following parseXxx() method to convert specified radix String to primitive.

**Syntax:**

public static primitive parseXxx(String s, int radix);

* Java

|  |
| --- |
| // Java program to illustrate parseXxx() Method    class GFG {      public static void main(String[] args)      {          int i = Integer.parseInt("1000", 2);          long l = Long.parseLong("1111", 4);            System.out.println(i);          System.out.println(l);      }  } |

**Output**

8

85

### 4. toString() Method

We can use the toString() method to convert the Wrapper object or primitive to String. There are a few forms of the toString() method:

**A. public String toString():** Every wrapper class contains the following toString() method to convert Wrapper Object to String type.

**Syntax:**

public String toString();

* Java

|  |
| --- |
| // Java program to illustrate toString() Method    class GFG {      public static void main(String[] args)      {          Integer I = new Integer(10);          String s = I.toString();          System.out.println(s);      }  } |

**Output:**

10

**B. toString(primitive p):** Every Wrapper class including the Character class contains the following static toString() method to convert primitive to String.

**Syntax:**

public static String toString(primitive p);

* Java

|  |
| --- |
| // Java program to illustrate toString()    class GFG {      public static void main(String[] args)      {          String s = Integer.toString(10);          System.out.println(s);            String s1 = Character.toString('a');          System.out.println(s1);      }  } |

**Output**

10

a

**C. toString(primitive p, int radix):**Integer and Long classes contain the following toString() method to convert primitive to specified radix String.

**Syntax:**

public static String toString(primitive p, int radix);

* Java

|  |
| --- |
| // Java program to illustrate toString() Method    class GFG {      public static void main(String[] args)      {          String s = Integer.toString(15, 2);          System.out.println(s);            String s1 = Long.toString(11110000, 4);          System.out.println(s1);      }  } |

**Output**

1111

222120121300