Top of Form

#### 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.

* **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:

#### 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**

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

#### Custom Wrapper class in Java

Java Wrapper classes wrap the primitive data types, that is why it is known as wrapper classes. We can also create a class which wraps a primitive data type. So, we can create a custom wrapper class in Java.

1. //Creating the custom wrapper class
2. **class** Javatpoint{
3. **private** **int** i;
4. Javatpoint(){}
5. Javatpoint(**int** i){
6. **this**.i=i;
7. }
8. **public** **int** getValue(){
9. **return** i;
10. }
11. **public** **void** setValue(**int** i){
12. **this**.i=i;
13. }
14. @Override
15. **public** String toString() {
16. **return** Integer.toString(i);
17. }
18. }
19. //Testing the custom wrapper class
20. **public** **class** TestJavatpoint{
21. **public** **static** **void** main(String[] args){
22. Javatpoint j=**new** Javatpoint(10);
23. System.out.println(j);
24. }}

**Output:**

10

Bottom of Form

Top of Form

#### Java Boolean class

The Boolean class wraps a value of the primitive type boolean in an object. Its object contains only a single field whose type is boolean.

1. **public** **class** JavaBooleanExample1 {
2. **public** **static** **void** main(String[] args) {
3. Boolean b1= **true**;
4. **boolean** b2=**false**;
5. //assigning boolean value of b1 to b3
6. Boolean b3= b1.booleanValue();
7. String str1 = "Value of boolean object "+b1+" is "+b3+".";
8. System.out.println(str1);
9. //compare b1 and b2
10. **int** val1 = Boolean.compare(b1,b2);
11. **if**(val1>0){
12. System.out.println("b1 is true.");
13. }
14. **else**{
15. System.out.println("b2 is true");
16. }
17. // logicalAnd() with return the same result as AND operator
18. Boolean val2 = Boolean.logicalAnd(b1,b2);
19. System.out.println("Logical And will return "+val2);
20. }
21. }

[**Test it Now**](https://compiler.javatpoint.com/opr/test.jsp?filename=JavaBooleanExample1)**Output:**

Value of boolean object true is true.

b1 is true.

Logical And will return false

#### Example 2

1. **public** **class** JavaBooleanExample2 {
2. **public** **static** **void** main(String[] args) {
3. Boolean b1 = **true**;
4. System.out.println("Boolean value = " + b1);
5. //prints the hash code of the boolean value
6. System.out.println("Hash Code for boolean value = " + b1.hashCode());
7. //converting boolean value tp String
8. String str = b1.toString();
9. System.out.println("String value = " + str);
10. System.out.println("Hash Code for String Value = " + str.hashCode());
11. // will return a boolean instance corresponding to Boolean b1
12. Boolean b2 = Boolean.valueOf(b1);
13. System.out.println("valueOf() method will return = " + b2);
14. **boolean** val1 = **false**;
15. **for** (**int** i = 0; i < 10; i++) {
16. **if** (i == 5) {
17. System.setProperty("val1", "true");
18. **break**;
19. }
20. }
21. **boolean** b3 = Boolean.getBoolean("val1");
22. System.out.println("value of val is " + b3);
23. }
24. }

[**Test it Now**](https://compiler.javatpoint.com/opr/test.jsp?filename=JavaBooleanExample2)**Output:**

Boolean value = true

Hash Code for boolean value = 1231

String value = true

Hash Code for String Value = 3569038

valueOf() method will return = true

value of val is true

Bottom of Form

Top of Form

#### Java Integer Class

The [Java](https://www.javatpoint.com/java-tutorial) Integer [class](https://www.javatpoint.com/object-and-class-in-java#class) comes under the **Java.lang.Number** package. This class wraps a value of the primitive type int in an [object](https://www.javatpoint.com/object-and-class-in-java#object). An object of Integer class contains a single field of type int value.

#### Java Integer Methods

The Java Integer class provides several methods for converting an int to a [String](https://www.javatpoint.com/java-string) and a String to an int, as well as other constants and methods dealing with an int. The various [Java](https://www.javatpoint.com/features-of-java) Integer methods are as follows:

SN Modifier & Type Method Discription

1) static int bitCount() It returns the number of 1-bits in the 2's complement binary representation of the specified int value.

2) byte byteValue() It converts the given number into a primitive byte type and returns the value of integer object as byte.

3) Static int compare() It compares two int values numerically and returns the result in integer equivalent.

4) Int compareTo() It compares two integer objects numerically and returns the result as -1, 0 or 1.

5) static int compareUnsigned() It compares two integer objects numerically treating the values as unsigned and returns the result as -1, 0 or 1.

6) static Integer decode() It decodes a String object into an Integer value.

7) static int divideUnsigned() It returns the unsigned quotient of dividing the first argument by the second argument where each argument and the result is interpreted as an unsigned value.

8) double doubleValue() It converts the given Integer value and returns the result as a double equivalent.

9) boolean equals() It compares the value of the parameter to the value of the current Integer object and returns boolean ( True or False ).

10) float floatValue() It converts the given Integer value and returns the result as a float equivalent.

11) Static Integer getInteger() It determines the integer value of the system property with the specified name.

12) static int hashCode() It returns a hash code for the given Integer.

13) static int highestOneBit() It returns int value with at most a single one-bit, in the position of the highest-order ("leftmost") one-bit in the specified int value.

14) static int lowestOneBit() It eturns int value with at most a single one-bit, in the position of the lowest-order ("rightmost") one-bit in the specified int value.

15) static int max() It returns the maximum value amongst the two method argument.

16) static int min() It returns the minimum value amongst the two method argument.

17) int intValue() It returns the value of the specified number as an int.

18) long longValue() It returns the value of the specified long object as long equivalent.

19) static int numberOfLeadingZeros() It returns the total number of zero bits preceding the highest-order ("leftmost") one-bit in the 2's complement binary representation of the specified int value.

20) static int numberOfTrailingZeros() It returns the total number of zero bits following the lowest-order ("rightmost") one-bit in the 2's complement binary representation of the specified int value.

21) static int parseInt() It parses the String argument as a signed decimal Integer object.

22) static int parseUnsignedInt() It parses the String argument as an unsigned decimal Integer object.

23) static int remainderUnsigned() It returns the unsigned remainder from dividing the first argument by the second argument where each argument and the result is interpreted as an unsigned value.

24) static int reverse() It returns the value obtained by reversing the order of the bits in the 2's complement binary representation of the specified int value.

25) static int reverseBytes() It returns the value obtained by reversing the order of the bytes in the 2's complement binary representation of the specified int value.

26) static int rotateLeft() It returns the value obtained by rotating the 2's complement binary representation of the specified int value left by the specified number of bits.

27) static int rotateRight() It returns the value obtained by rotating the 2's complement binary representation of the specified int value right by the specified number of bits.

28) short shortValue() It returns the value of this Integer as a short type after a primitive conversion.

29) static int signum() It returns the signum function of the specified int value.

30) static int sum() It returns the sum by adding two integers together as per the + operator.

31) static String toBinaryString() It returns a string representation of the integer argument as an unsigned integer in binary base 2.

32) static String toHexString() It returns a string representation of the integer argument as an unsigned integer in binary base 16.

33) static String toOctalString() It returns a string representation of the integer argument as an unsigned integer in binary base 8.

34) String toString() It returns a String object representing the value of the Number Object.

35) static String toUnsignedString() It converts the argument to a long by an unsigned conversion.

36) static long toUnsignedLong() It returns a string representation of the argument as an unsigned decimal value.

37) static Integer valueOf() It returns the relevant Integer Object holding the value of the argument passed.

Bottom of Form

Top of Form

#### Java Character class

The Character class generally wraps the value of all the primitive type char into an object. Any object of the type Character may contain a single field whose type is char.

All the fields, methods, and constructors of the class Character are specified by the Unicode Data file which is particularly a part of Unicode Character Database and is maintained by the Unicode Consortium.

A set of characters ranging from U+0000 to U+FFFF is sometimes known as the Basic Multilingual Plane(i.e. BMP). The characters whose codePoints are greater than U+FFFF are known as supplementary characters. The Java language generally uses the UTF-16 encoding method to represent the char arrays in String or String Buffer. In this type of representation, the supplementary characters are represented as a pair of characters, the first which is from the high-surrogate range(\uD800-\uDBFF) and the second which is from the low-surrogate range(\uDc00-\uDBFF).

#### Methods

Method Description

charCount(int codePoint) Determines the number of char values which are required to represent the given character.

charValue() Returns the value of the given Character object.

codePointAt(char[]a, int index) Returns the codePoint for the specified index of the given array.

codePointAt(char[]a, int index, int limit ) Returns the codePoint of the char array at the specified index where only the elements of the array with the index less than the specified limit being used.

codePointAt(CharSequence seq, int index) Returns the codePoint at the specified index for the given CharSequence.

codePointBefore(char[]a, int index) Returns the codePoint for the given array in the preceding index.

codePointBefore(char[]a, int index, int start) Returns the codePoint for the given array in the preceding index where only the elements of the array with index greater than or equal to start are used.

codePointBefore(CharSequence, int index) Returns the CodePoint for the given CharSequence int the preceding index.

codePointCount(char[]a, int offset, int count) Returns the total number of Unicode codePoints in the given subarray of the char array argument.

codePointCount(CharSequence seq, int beginIndex, int endIndex) Returns the number of Unicode codePoints provided in the given text range for the specified char sequence.

codePointOf(String name) Returns the value of the codePoint for the given Unicode character specified by a particular character name.

compare(char x, char y) Compares two character type values numerically.

compareTo(Character anotherCharacter) Compares two Character type objects numerically.

digit(char ch, int radix) Returns the numeric value for the given character in the specified index.

digit(int codePoint, int radix) Returns the numeric value for the given character in the specified radix.

equals(object obj) Compares the given object with the specified object.

forDigit(int digit, int radix) Determines the presentation of the character for a particular digit in the given radix.

getDirectionality(char ch) Returns the Unicode directionality property for the specified character.

getDirectionality(int codePoint) Returns the Unicode directionality property for the specified codePoint.

getName(int codePoint) Returns the Unicode name for the given codePoint or may return null if the codePoint is not assigned.

getNumericValue(char ch) Returns the integer type value of the specified Unicode character.

getNumericValue(int codePoint) Returns the integer type value of the specified Unicode codePoint.

getType(char ch) Returns the value which indicates the general category of the character.

getType(int codePoint) Returns the value which indicates the general category of the character.

hashcode() Returns the hash code for the specified character. This method is equal to the result of invoking the charValue() method.

hashcode(char value) Returns the hash code for a particular char value.

highSurrogate(int codePoint) Returns the leading surrogate codePoint of a particular surrogate pair which represents the supplementary character in the UTF-16 encoding.

isAlphabetic(int codePoint) Determines whether the given character is an alphabet or not.

isBmpCodePoint(int codePoint) Determines whether the given character is in the Basic Multilingual Plane or not.

isDefined(char ch) Determines whether the character is defined in the Unicode or not.

isDefined(int codePoint) Determines whether the character(Unicode code point) is defined in the Unicode or not.

isDigit(char ch) Determines whether the given character is a digit or not.

isDigit(int codePoint) Determines whether the given character(Unicode code point) is a digit or not.

isHighSurrogate(char ch) Determines whether the given char values is a Unicode high-surrogate code point or not.

isIndentifierIgnorable(char ch) Determines whether the given character is considered to be an ignorable character in the Java identifier or not.

isIndentifierIgnorable(int codePoint) Determines whether the given character(Unicode codePoint) is considered to be an ignorable character in the Java identifier or not.

isIdeographic(int codePoint) Determines whether the specified character is a CJKV(Chinese, Japanese, Korean and Vietnamese) ideograph or not.

isISOControl(char ch) Determines whether the given character is an ISO Control character or not.

isISOControl(int codePoint) Determines whether the given character(Unicode code point) is an ISO Control character or not.

isJavaIdentifierPart(char ch) Determines whether the given character is a part of the Java identifier other than the first character or not.

isJavaIdentifierPart(int codePoint) Determines whether the given character(Unicode code point) is a part of the Java identifier other than the first character or not.

isJavaIdentifierStart(char ch) Determines whether the given character is permissible as the first character in the Java identifier or not.

isJavaIdentifierStart(int codePoint) Determines whether the given character(Unicode code point is permissible as the first character in the Java identifier or not.

isJavaLetter(char ch) Replaced by the isJavaIdentifierStart(char).

isJavaLetterOrDigit(int codePoint) Replaced by the isJavaIdentifierPart(char).

isLetter(char ch) Determines whether the specified character is a letter or not.

isLetter(int codePoint) Determines whether the specified character(Unicode code point) is a letter or not.

isLetterOrDigit(char ch) Determines whether the specified character is a letter or digit or not.

isLetterOrDigit(int codePoint) Determines whether the specified character(Unicode code point) is a letter or digit or not.

isLowerCase(char ch) Determines whether the specified character is a lowercase character or not.

isLowerCase(int codePoint) Determines whether the specifed character(Unicode code point) is a lowercase character or not.

isLowSurrogate(char ch) Determines whether the given char value is a Unicode low-surrogate code unit or not.

isMirrored(char ch) Determines whether the character is mirrored or not.

isMirrored(int codePoint) Determines whether the character(Unicode code point) is mirrored or not.

isSpace(char ch) Replaced by isWhiteSpace(char ch) method.

isSpaceChar(char ch) Determines whether the specified character is a Unicode space character or not.

isSupplementaryCodePoint(int codePoint) Determines whether the given character(Unicode code point) is in the range of supplementary characters or not.

isSurrogate(char ch) Determines whether the given char value is a Unicode surrogate code unit.

isSurrogatePair(char high, char low) Determines whether the given pair of char value is a valid Unicode surrogate pair or not.

isTitleCase(char ch) Determines whether the given character is a titlecase character or not.

isTitleCase(int codePoint) Determines whether the given character(Unicode code point) is a titlecase character or not.

isUnicodeIdentifierPart(char ch) Determines whether the given character is a part of Unicode identifier or not.

isUnicodeIdentifierPart(int codePoint) Determines whether the given character(unicofde code point) is a part of Unicode identifier or not.

isUnicodeIdentifierStart(char ch) Determines whether the given character is permissible as the first character in the Unicode identifier or not.

isUnicodeIdentifierStart(int codePoint) Determines whether the given character(Unicode code point) is permissible as the first character in the Unicode identifier or not.

isUpperCase(char ch) Determines whether the given character is an uppercase character or not.

isUpperCase(int codePoint) Determines whether the given character(Unicode code point) is an uppercase character or not.

isValidCodePoint(int codePoint) Determines whether the given character is a valid Unicode code point value or not.

isWhitespace(char ch) Determines whether the given character is a white space or not.

isWhitespace(int codePoint) Determines whether the given character(Unicode code point)is a white space or not .

lowSurrogate(int codePoint) Returns the low surrogate code unit(i.e trailing surrogate)of the surrogate pair which represents the supplementary character in the UTF-16 encoding.

offsetByCodePoints(char[]a, int start, int count, int index, int codePointOffset) Returns the index for the given char subarray which is the offset from the given index by the codePointOffset code points.

offsetByCodePoints(CharSequence seq, int index, int codePointOffset) Returns the index for the given char subarray which is the offset from the given index by the codePointOffset code points.

reverseBytes(char ch) Returns the value obtained by reversing the order of the given bytes in the specified char values.

toChars(int codePoint) Converts the specified character(Unicode code point) into its UTF-16 representation which is stored in a char array.

toChars(int codePoint, char[] dst, int dstIndex) Converts the specified character(Unicode code point) into its UTF-16 representation.

toCodePoint(char high, char low) Converts the given surrogate pair to its corresponding supplementary code point value.

toLowerCase(char ch) Converts the character type argument to lowercase using case mapping information obtained from the Unicode Data file.

toLowerCase(int codePoint) Converts the character type argument(Unicode code point) to lowercase using case mapping information obtained from the Unicode Data file.

toString() Returns a String type object which represents the value of the character.

toString(char ch) Returns a String type object which represents the specified character

toTitleCase(char ch) Converts the character type argument to titlecase using case mapping information obtained from the Unicode Data file.

toTitleCase(int codePoint) Converts the character type argument(Unicode code point) to lowercase using case mapping information obtained from the Unicode Data file.

toUpperCase(char ch) Converts the character type argument to uppercase using case mapping information obtained from the Unicode Data file.

toUpperCase(int codePoint) Converts the character type argument(Unicode code point) to uppercase using case mapping information obtained from the Unicode Data file.

valueOf(char c) Returns the instance of a Character which represents the specified character value.

#### Example 1

1. **import** java.util.Scanner;
2. **public** **class** JavaCharacterExample1 {
3. **public** **static** **void** main(String[] args) {
4. // Ask the user for the first input.
5. System.out.print("Enter the first input:");
6. // Use the Scanner class to get the user input.
7. Scanner scanner = **new** Scanner(System.in);
8. // Gets the user input.
9. **char**[] value1 = scanner.nextLine().toCharArray();
10. **int** result1 = 0;
11. // Count the characters for a specific character.
12. **for** (**char** ch1 : value1) {
13. result1 = Character.charCount(ch1);
14. }
15. // Print the result.
16. System.out.print("The value comes to: "+result1+"\n");
18. System.out.print("Enter the second input:");
19. **char**[] value2 = scanner.nextLine().toCharArray();
20. **for** (**char** ch2 : value2) {
21. **int** result2 = Character.hashCode(ch2);
22. System.out.print("The hash code for the character '"+ch2+"' is given as:"+result2+"\n");
23. }
24. System.out.print("Enter the third input:");
25. **char**[] value3 = scanner.nextLine().toCharArray();
26. **for** (**char** ch3 : value3) {
27. **boolean** result3 = Character.isDigit(ch3);
28. **if**(result3){
29. System.out.println("The character '" + ch3 + "' is a digit. ");
30. }
31. **else**{
32. System.out.println("The character '" + ch3 + "' is not a digit.");
33. }
34. System.out.print("Enter the fourth input:");
35. **char**[] value4 = scanner.nextLine().toCharArray();
36. **for** (**char** ch4 : value4) {
37. **boolean** result4 = Character.isISOControl(ch4);
38. System.out.println("The fourth character '"+ch4+"' is an ISO Control:"+result4);
39. }
40. }
41. }
42. }

**Output:**

Enter the first input:89

The value comes to: 1

Enter the second input:J

The hash code for the character 'J' is given as:74

Enter the third input:5

The character '5' is a digit.

Enter the fourth input:h

The fourth character 'h' is an ISO Control:false

Bottom of Form

Top of Form

**Reference Links:**

https://www.javatpoint.com/post/java-character

https://www.javatpoint.com/wrapper-class-in-java

https://www.javatpoint.com/java-boolean

**Video Link:**

https://youtu.be/9ch\_rkRwk1M

https://youtu.be/RBHtgLSUiw0

Bottom of Form