

java.lang.Character class methods | Set 1

Last Updated : 10 Sep, 2021

Method	Syntax	Description
charCount()	<code>public static int charCount(int argchar)</code>	2 if the character is valid i.e. > or = to 0x1000 (supplementary character); else 1.
charValue()	<code>public char charValue()</code>	Primitive character value of the defined char Object.
codePointAt()	<code>public static int codePointAt(char[] array, int position)</code>	Unicode point of the character array present at the given position.
codePointBefore()	<code>public static int codePointBefore(char[] array, int position)</code> or <code>public static int codePointBefore(char[] array, int position, int start)</code>	Unicode point of the character array present before the given position.
codePointCount()	<code>public static int codePointCount(char[] array, int start, int len)</code>	No. of Unicode Point of the sub-character array
compareTo()	<code>public int compareTo(Character argchar)</code>	=0 - characters are equal >0 - this character is greater <0 - argument character is greater
equals()	<code>public boolean equals(Object charObj)</code>	True if both the objects are equal, else false.
getNumericValue()	<code>public static int getNumericValue(char arg)</code>	int value for the specific Unicode character. If Unicode value doesn't exist -1 is returned
getType()	<code>public static int getType(char arg)</code>	int value for the argument character representing its general type category.

lang.Character class wraps the value of a primitive data type – char to an object of datatype char and this object contains single field having the data type – char. This class provides no. of methods regarding character manipulations like convert them from lowercase to uppercase. Character class is based on Unicode Standards to provide character information.

Class Declaration:

```
public final class Character
    extends Object
        implements Serializable, Comparable
```

Following methods of Character class are discussed here :

1. **charCount() : java.lang.charCount()** method uses Unicode point to return the number of char values to represent the argument char values. A **Unicode code point** is used for character values in the range between U+0000 and U+10FFFF and for 16-bit char values that are code units of the UTF-16 encoding.

Syntax :

```
public static int charCount(int argchar)
```

Parameters :

argchar : char i.e. it's Unicode point to be counted

Return :

2 if the character is valid i.e. > or = to 0X1000(supplementary character).



2. **charValue() : java.lang.Character.charValue()** method returns primitive character value of defined char Object.

Syntax :

```
public char charValue()
```

Return :

primitive character value of defined char Object.

3. **codePointAt() : java.lang.Character.codePointAt(char[] array, int position)** method returns Unicode Point of the character array present at the argued position.

Syntax :

```
public static int codePointAt(char[] array, int position)
```

Parameters :

array : character array

position : array index of character whose Unicode Point value you need.

Return :

Unicode point of the character array present at the given position

Java code explaining use of charCount(), charValue(), codePointat() methods

```
// Java program explaining Character class methods
// charCount(), charValue(), codePointat()
import java.lang.Character;
public class NewClass
{
    public static void main(String[] args)
    {
        // Use of charCount() method
        int geek = 0x9999,    // < 0x10000
            geek1 = 0x10000,  // = 0x10000
            geek2 = 0x10001;  // > 0x10000

        int check = Character.charCount(geek);
        int check1 = Character.charCount(geek1);
        int check2 = Character.charCount(geek2);

        if (check == 2)    // Checking for geek
            System.out.println("Valid Character geek");
        else
            System.out.println("Invalid Character geek");
    }
}
```

```
if (check1 ==2)    // Checking for geek1
    System.out.println("Valid Character geek1");
else
    System.out.println("Invalid Character geek1");

if (check2 ==2)    // Checking for geek2
    System.out.println("Valid Character geek2");
else
    System.out.println("Invalid Character geek2");

System.out.println("");

// Use of charValue() method
Character m;        // Character object m

m = new Character('g'); // Assigning value g to m;

char gfg;
gfg = m.charValue();
System.out.println("Primitive value of gfg : " +gfg);
System.out.println("");

// Use of codePointAt()
char[] arg = new char[] { 'g', 'e', 'e', 'k', 's' };
int val, val1, position = 3;

val = Character.codePointAt(arg, position);
val1 = Character.codePointAt(arg, 0);

System.out.println( "Unicode code point at " + position
                    + " : "+val );
System.out.println( "Unicode code point at 0 : " + val1);
}
```

Output:

```
Invalid Character geek
Valid Character geek1
Valid Character geek2
```

```
Primitive value of gfg : g
```

```
Unicode code point at 3 : 107
Unicode code point at 0 : 103
```

4. codePointBefore() : java.lang.Character.codePointBefore(char[] array, int position) method returns Unicode Point of the character array present before the argued position.

Syntax :

```
public static int codePointBefore(char[] array, int position)
                                or
public static int codePointBefore(char[] array, int position, int start)
```

Parameters :

array : character array

position : array index of character following the Unicode Point value you want

start : start index of the character array

Return :

Unicode point of the character array present before the given position



5. codePointCount() : java.lang.Character.codePointCount() method returns no. of Unicode Point of the sub-character array.

Syntax :

```
public static int codePointCount(char[] array, int start, int len)
```

Parameters :

array : character array

start : starting index of the array

length : length of the character sub-array

Return :

no. of Unicode Point of the sub-character array.

Exception :

--> NullPointerException

--> IndexOutOfBoundsException

6. compareTo() : java.lang.Character.compareTo(Character argChar) method compares given character with argued character.

Syntax :

```
public int compareTo(Character argChar)
```

Parameters :

argChar : character to be compared with

Return :

= 0 : if both characters are equal

> 0 : if given this character is greater

< 0 : if argued character is greater

Java code explaining use of codePointBefore(), codePointCount(), compareTo() methods

```
// Java program explaining Character class methods
// codePointBefore(), codePointCount(), compareTo()
import java.lang.Character;
public class NewClass
{
    public static void main(String[] args)
    {
        // Use of codePointBefore()
        char[] arg = new char[] { 'g', 'e', 'e', 'k', 's' };
        int position = 4;

        int val = Character.codePointBefore(arg, position);
        int val1 = Character.codePointBefore(arg, 1);
        int val2 = Character.codePointBefore(arg, 3, 1);

        System.out.println( "Unicode code point before " + position +
                               " : " + val );
        System.out.println( "Unicode code point before 1 : " + val1 );
        System.out.println( "Unicode code point before 3 to 1 : "
                               + val2);

        System.out.println("");

        // Use of codePointCount()
        int count = Character.codePointCount(arg, 1,3 );

        System.out.println("No. of Unicode points : " + count);
        System.out.println("");

        // Use of compareTo()
        Character g1 = new Character('g');
        Character g2 = new Character('o');

        int check = g1.compareTo(g2);
        System.out.println("g1 < g2 : " + check);
        int check1 = g2.compareTo(g1);
        System.out.println("g2 > g1 : " + check1);
        int check2 = g2.compareTo(g2);
        System.out.println("g2 = g2 : " + check2);
    }
}
```

Output:

```
Unicode code point before 4 : 107
Unicode code point before 1 : 103
Unicode code point before 3 to 1 : 101
```

No. of Unicode points : 3

```
g1 g1 : 8
g2 = g2 : 0
```

7. **equals() : java.lang.Character.equals()** method compares the present char object with the argumented char object.

Syntax :

```
public boolean equals(Object charObj)
```

Parameters :

charObj : char object to compare with

Return :

true if both the objects are equal, else false.

8. **getNumericValue() : java.lang.Character.getNumericValue(char arg)** method returns int value for the specific Unicode character.

A – Z value ranges u0041 to u005A

a -z value ranges u0061 to u007A

Syntax :

```
public static int getNumericValue(char arg)
```

Parameters :

arg : char value

Return :

int value for the specific Unicode character.

if Unicode value doesn't exists -1 is returned.

9. **getType() : java.lang.Character.getType(char arg)** method identifies the general type of character

A – Z value ranges u0041 to u005A

a -z value ranges u0061 to u007A

Syntax :

```
public static int getType(char arg)
```

Parameters :

arg : char value

Return :

int value for the argumented character representing its general type category

Java code explaining use of equals(), getNumericValue(), getType() methods

```
// Java program explaining Character class methods
// equals(), getNumericValue(), getType()
import java.lang.Character;
public class NewClass
{
    public static void main(String[] args)
    {
        // Use of equals() method
        Character g1 = new Character('g');
        Character g2 = new Character('O');

        boolean check = g1.equals(g2);
        boolean check1 = g1.equals(g1);
        System.out.println("Are g and o equal? : " + check);
        System.out.println("Are g and g equal? : " + check1);
        System.out.println("");

        // Use of getNumericValue() method
        int c = Character.getNumericValue(g1);
        int c1 = Character.getNumericValue(g2);
        System.out.println("Int value for g : " + c);
        System.out.println("Int value for A : " + c1);
        System.out.println("");

        // Use of getType() method
        Character g3 = new Character('$');
        Character g4 = new Character('6');

        int r1 = Character.getType(g1);
        int r2 = Character.getType(g2);
        int r3 = Character.getType(g3);
        int r4 = Character.getType(g4);

        System.out.println("Type for lowercase : " + r1);
        System.out.println("Type for uppercase : " + r2);
        System.out.println("Type for currency : " + r3);
        System.out.println("Type for numeric : " + r4);
    }
}
```

Output:

```
Are g and o equal? : false
```

```
Are g and g equal? : true
```

```
Int value for g : 16
```

```
Int value for A : 24
```

```
Type for lowercase : 2
```

```
Type for uppercase : 1
```

```
Type for currency  : 26
```

```
Type for numeric   : 9
```

This article is contributed by **Mohit Gupta**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.