Java.lang.Character Class in Java

Difficulty Level : Medium   Last Updated : 06 Dec, 2018

Java provides a wrapper class **Character** in java.lang package. An object of type Character contains a single field, whose type is char.

- **Creating a Character object :**

  ```
  Character ch = new Character('a');
  ```

  The above statement creates a Character object which contain 'a' of type char. There is only one constructor in Character class which expect an argument of char <u>data type</u>.

- If we pass a primitive char into a method that expects an object, the compiler automatically converts the char to a Character class object. This feature is called <u>Autoboxing and Unboxing</u>.
- **Note :** The Character class is immutable like String class i.e once it's object is created, it **cannot** be changed.

Methods in Character Class :

1. **boolean isLetter(char ch)** : This method is used to determine whether the specified char value(ch) is a letter or not. The method will return true if it is letter([A-Z],[a-z]), otherwise return false. In place of character, we can also pass ASCII value as an argument as char to int is implicitly typecasted in java.

   ```
   Syntax :
   boolean isLetter(char ch)
   Parameters :
   ch - a primitive character
   Returns :
   returns true if ch is a alphabet, otherwise return false
   ```

   ```java
   // Java program to demonstrate isLetter() method
   public class Test
   {
       public static void main(String[] args)
       {
           System.out.println(Character.isLetter('A'));

           System.out.println(Character.isLetter('0'));
   ```

```
        }
    }
```

Output:

```
true
false
```

2. **boolean isDigit(char ch)** : This method is used to determine whether the specified char value(ch) is a digit or not. Here also we can pass ASCII value as an argument.

Syntax :
boolean isDigit(char ch)
Parameters :
ch - a primitive character
Returns :
returns true if ch is a digit, otherwise return false

```java
// Java program to demonstrate isDigit() method
public class Test
{
    public static void main(String[] args)
    {
        // print false as A is character
        System.out.println(Character.isDigit('A'));

        System.out.println(Character.isDigit('0'));

    }
}
```

Output:

```
false
true
```

3. <u>boolean isWhitespace(char ch</u>**)** : It determines whether the specified char value(ch) is white space. A whitespace includes space, tab, or new line.

Syntax :
```
boolean isWhitespace(char ch)
```
Parameters :
```
ch - a primitive character
```
Returns :
```
returns true if ch is a whitespace.
otherwise return false
```

```java
// Java program to demonstrate isWhitespace() method
public class Test
{
    public static void main(String[] args)
    {
        System.out.println(Character.isWhitespace('A'));
        System.out.println(Character.isWhitespace(' '));
        System.out.println(Character.isWhitespace('\n'));
        System.out.println(Character.isWhitespace('\t'));

        //ASCII value of tab
        System.out.println(Character.isWhitespace(9));

        System.out.println(Character.isWhitespace('9'));
    }
}
```

Output:

```
false
true
true
true
true
false
```

4. **boolean isUpperCase(char ch)** : It determines whether the specified char value(ch) is uppercase or not.

Syntax :

```
boolean isUpperCase(char ch)
```

```java
// Java program to demonstrate isUpperCase() method
public class Test
{
  public static void main(String[] args)
  {
    System.out.println(Character.isUpperCase('A'));
    System.out.println(Character.isUpperCase('a'));
    System.out.println(Character.isUpperCase(65));
  }
}
```

Output:

```
true
false
true
```

5. **boolean isLowerCase(char ch)** : It determines whether the specified char value(ch) is lowercase or not.

Syntax :

```
boolean isLowerCase(char ch)
```

```java
// Java program to demonstrate isLowerCase() method
public class Test
{
  public static void main(String[] args)
  {
    System.out.println(Character.isLowerCase('A'));
    System.out.println(Character.isLowerCase('a'));
    System.out.println(Character.isLowerCase(97));
  }
}
```

Output:

```
false
true
true
```

6. **char toUpperCase(char ch)** : It returns the uppercase of the specified char value(ch). If an ASCII value is passed, then the ASCII value of it's uppercase will be returned.

Syntax :
char toUpperCase(char ch)
Parameters :
ch - a primitive character
Returns :
returns the uppercase form of the specified char value.

```java
// Java program to demonstrate toUpperCase() method
public class Test
{
    public static void main(String[] args)
    {
      System.out.println(Character.toUpperCase('a'));
      System.out.println(Character.toUpperCase(97));
      System.out.println(Character.toUpperCase(48));
    }
}
```

Output:

```
A
65
48
```

7. **char toLowerCase(char ch)** : It returns the lowercase of the specified char value(ch).

Syntax :
char toLowerCase(char ch)
Parameters :
ch - a primitive character

Returns :

returns the lowercase form of the specified char value.

```java
// Java program to demonstrate toLowerCase() method
public class Test
{
    public static void main(String[] args)
    {
      System.out.println(Character.toLowerCase('A'));
      System.out.println(Character.toLowerCase(65));
      System.out.println(Character.toLowerCase(48));
    }
}
```

Output:

```
a
97
48
```

8. **toString(char ch)** : It returns a String class object representing the specified character value(ch) i.e a one-character string. Here we **cannot** pass ASCII value.

Syntax :
String toString(char ch)

Parameters :
ch - a primitive character

Returns :
returns a String object.

```java
// Java program to demonstrate toString() method
public class Test
{
    public static void main(String[] args)
    {
     System.out.println(Character.toString('x'));
     System.out.println(Character.toString('Y'));
    }
}
```

Output:

x
y

9. static int charCount(int codePoint): This method determines the number of char values needed to represent the specified character (Unicode code point).

10. char charValue(): This method returns the value of this Character object.

11. **static int codePointAt(char[] a, int index)**: This method returns the code point at the given index of the char array.

12. **static int codePointAt(char[] a, int index, int limit)**: This method returns the code point at the given index of the char array, where only array elements with index less than limit can be used.

13. **static int codePointAt(CharSequence seq, int index)**: This method returns the code point at the given index of the CharSequence.

14. **static int codePointBefore(char[] a, int index)**: This method returns the code point preceding the given index of the char array.

15. **static int codePointBefore(char[] a, int index, int start)**: This method returns the code point preceding the given index of the char array, where only array elements with index greater than or equal to start can be used.

16. **static int codePointBefore(CharSequence seq, int index)**: This method returns the code point preceding the given index of the CharSequence.

17. **static int codePointCount(char[] a, int offset, int count)**: This method returns the number of Unicode code points in a subarray of the char array argument.

18. **static int codePointCount(CharSequence seq, int beginIndex, int endIndex)**: This method returns the number of Unicode code points in the text range of the specified char sequence.

19. **static int codePointOf(String name)**: This method returns the code point value of the Unicode character specified by the given Unicode character name.

20. **static int compare(char x, char y)**: This method compares two char values numerically.

21. **int compareTo(Character anotherCharacter)**: This method compares two Character objects numerically.

22. static int digit(char ch, int radix): This method returns the numeric value of the character ch in the specified radix.

23. static int digit(int codePoint, int radix): This method returns the numeric value of the specified character (Unicode code point) in the specified radix.

24. boolean equals(Object obj): This method compares this object against the specified object.

25. **static char forDigit(int digit, int radix)**: This method determines the character representation for a specific digit in the specified radix.

26. static byte getDirectionality(char ch): This method returns the Unicode directionality property for the given character.

27. static byte getDirectionality(int codePoint): This method returns the Unicode directionality property for the given character (Unicode code point).

28. **static String getName(int codePoint)**: This method returns the Unicode name of the specified character codePoint, or null if the code point is unassigned.

29. **static int getNumericValue(char ch)**: This method returns the int value that the specified Unicode character represents.

30. **static int getNumericValue(int codePoint)**: This method returns the int value that the specified character (Unicode code point) represents.

31. static int getType(char ch): This method returns a value indicating a character's general category.

32. static int getType(int codePoint): This method returns a value indicating a character's general category.

33. int hashCode(): This method returns a hash code for this Character; equal to the result of invoking charValue().

34. static int hashCode(char value): This method returns a hash code for a char value; compatible with Character.hashCode().

35. **static char highSurrogate(int codePoint)**: This method returns the leading surrogate (a high surrogate code unit) of the surrogate pair representing the specified supplementary character (Unicode code point) in the UTF-16 encoding.

36. **static boolean isAlphabetic(int codePoint)**: This method determines if the specified character (Unicode code point) is an alphabet.

37. **static boolean isBmpCodePoint(int codePoint)**: This method determines whether the specified character (Unicode code point) is in the Basic Multilingual Plane (BMP).

38. **static boolean isDefined(char ch)**: This method determines if a character is defined in Unicode.

39. **static boolean isDefined(int codePoint)**: This method determines if a character (Unicode code point) is defined in Unicode.

40. static boolean isHighSurrogate(char ch): This method determines if the given char value is a Unicode high-surrogate code unit (also known as leading-surrogate code unit).

41. static boolean isIdentifierIgnorable(char ch): This method determines if the specified character should be regarded as an ignorable character in a Java identifier or a Unicode identifier.

42. static boolean isIdentifierIgnorable(int codePoint): This method determines if the specified character (Unicode code point) should be regarded as an ignorable character in a Java identifier or a Unicode identifier.

43. **static boolean isIdeographic(int codePoint)**: This method determines if the specified character (Unicode code point) is a CJKV (Chinese, Japanese, Korean and Vietnamese) ideograph, as defined by the Unicode Standard.

44. static boolean isISOControl(char ch): This method determines if the specified character is an ISO control character.

45. static boolean isISOControl(int codePoint): This method determines if the referenced character (Unicode code point) is an ISO control character.

46. static boolean isJavaIdentifierPart(char ch): This method determines if the specified character may be part of a Java identifier as other than the first character.

47. static boolean isJavaIdentifierPart(int codePoint): This method determines if the character (Unicode code point) may be part of a Java identifier as other than the first character.

48. static boolean isJavaIdentifierStart(char ch): This method determines if the specified character is permissible as the first character in a Java identifier.

49. static boolean isJavaIdentifierStart(int codePoint): This method determines if the character (Unicode code point) is permissible as the first character in a Java identifier.

50. static boolean isLowSurrogate(char ch): This method determines if the given char value is a Unicode low-surrogate code unit (also known as trailing-surrogate code unit).

51. static boolean isLetterOrDigit(char ch): This method determines if the specified character is a letter or digit.

52. static boolean isLetterOrDigit(int codePoint): This method determines if the specified character (Unicode code point) is a letter or digit.

53. static boolean isMirrored(char ch): This method determines whether the character is mirrored according to the Unicode specification.

54. static boolean isMirrored(int codePoint): This method determines whether the specified character (Unicode code point) is mirrored according to the Unicode specification.

55. static boolean isSpaceChar(char ch): This method determines if the specified character is a Unicode space character.

56. static boolean isSpaceChar(int codePoint): This method determines if the specified character (Unicode code point) is a Unicode space character.

57. static boolean isSupplementaryCodePoint(int codePoint): This method determines whether the specified character (Unicode code point) is in the supplementary character

range.

58. **static boolean isSurrogate(char ch)**: This method determines if the given char value is a Unicode surrogate code unit.

59. **static boolean isSurrogatePair(char high, char low)**: This method determines whether the specified pair of char values is a valid Unicode surrogate pair.

60. static boolean isTitleCase(char ch): This method determines if the specified character is a titlecase character.

61. static boolean isTitleCase(int codePoint): This method determines if the specified character (Unicode code point) is a titlecase character.

62. static boolean isUnicodeIdentifierPart(char ch): This method determines if the specified character may be part of a Unicode identifier as other than the first character.

63. static boolean isUnicodeIdentifierPart(int codePoint): This method determines if the specified character (Unicode code point) may be part of a Unicode identifier as other than the first character.

64. **static boolean isUnicodeIdentifierStart(char ch)**: This method determines if the specified character is permissible as the first character in a Unicode identifier.

65. **static boolean isUnicodeIdentifierStart(int codePoint)**: This method determines if the specified character (Unicode code point) is permissible as the first character in a Unicode identifier.

66. static boolean isValidCodePoint(int codePoint): This method determines whether the specified code point is a valid Unicode code point value.

67. **static char lowSurrogate(int codePoint)**: This method returns the trailing surrogate (a low surrogate code unit) of the surrogate pair representing the specified supplementary character (Unicode code point) in the UTF-16 encoding.

68. static int offsetByCodePoints(char[] a, int start, int count, int index, int codePointOffset): This method returns the index within the given char subarray that is offset from the given index by codePointOffset code points.

69. static int offsetByCodePoints(CharSequence seq, int index, int codePointOffset): This method returns the index within the given char sequence that is offset from the given index by codePointOffset code points.

70. **static char reverseBytes(char ch)**: This method returns the value obtained by reversing the order of the bytes in the specified char value.

71. **static char[] toChars(int codePoint)**: This method converts the specified character (Unicode code point) to its UTF-16 representation stored in a char array.

72. **static int toChars(int codePoint, char[] dst, int dstIndex)**: This method converts the specified character (Unicode code point) to its UTF-16 representation.

73. **static int toCodePoint(char high, char low)**: This method converts the specified surrogate pair to its supplementary code point value.

74. **static char toTitleCase(char ch)**: This method converts the character argument to titlecase using case mapping information from the UnicodeData file.

75. **static int toTitleCase(int codePoint)**: This method converts the character (Unicode code point) argument to titlecase using case mapping information from the UnicodeData file.

76. static Character valueOf(char c): This method returns a Character instance representing the specified char value.

Escape Sequences :

A character preceded by a backslash (\) is an escape sequence and has special meaning to the compiler. The following table shows the Java escape sequences:

Escape Sequence	Description
\t	Insert a tab in the text at this point.
\b	Insert a backspace in the text at this point.
\n	Insert a newline in the text at this point.
\r	Insert a carriage return in the text at this point.
\f	Insert a formfeed in the text at this point.
\'	Insert a single quote character in the text at this point.
\"	Insert a double quote character in the text at this point.
\\	Insert a backslash character in the text at this point.
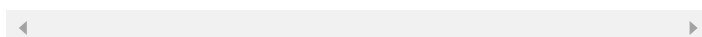
When an escape sequence is encountered in a print statement, the compiler interprets it accordingly. For example, if you want to put quotes within quotes you must use the escape sequence, \", on the interior quotes. To print the sentence

```
She said "Hello!" to me.
```

you would write

```
System.out.println("She said \"Hello!\" to me.");
```

This article is contributed by **Gaurav Miglani**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@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.