JAVA.LANG.CHARACTER CLASS

Introduction

The **java.lang.Character** class wraps a value of the primitive type char in an object. An object of type Character contains a single field whose type is char.

Class declaration

Following is the declaration for **java.lang.Character** class:

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

Field

Following are the fields for **java.lang.Character** class:

- **static byte COMBINING_SPACING_MARK** -- This is the General category "Mc" in the Unicode specification.
- **static byte CONNECTOR_PUNCTUATION** -- This is the General category "Pc" in the Unicode specification.
- **static byte CONTROL** -- This is the General category "Cc" in the Unicode specification.
- **static byte CURRENCY_SYMBOL** -- This is the General category "Sc" in the Unicode specification.
- **static byte DASH_PUNCTUATION** -- This is the General category "Pd" in the Unicode specification.
- **static byte DECIMAL_DIGIT_NUMBER** -- This is the General category "Nd" in the Unicode specification.
- **static byte DIRECTIONALITY_ARABIC_NUMBER** -- This is the Weak bidirectional character type "AN" in the Unicode specification.
- **static byte DIRECTIONALITY_BOUNDARY_NEUTRAL** -- This is the Weak bidirectional character type "BN" in the Unicode specification.
- **static byte DIRECTIONALITY_COMMON_NUMBER_SEPARATOR** -- This is the Weak bidirectional character type "CS" in the Unicode specification.
- **static byte DIRECTIONALITY_EUROPEAN_NUMBER** -- This is the Weak bidirectional character type "EN" in the Unicode specification.
- **static byte DIRECTIONALITY_EUROPEAN_NUMBER_SEPARATOR** -- This is the Weak bidirectional character type "ES" in the Unicode specification.
- **static byte DIRECTIONALITY_EUROPEAN_NUMBER_TERMINATOR** -- This is the Weak bidirectional character type "ET" in the Unicode specification.
- **static byte DIRECTIONALITY_LEFT_TO_RIGHT** -- This is the Strong bidirectional character type "L" in the Unicode specification.
- **static byte DIRECTIONALITY_LEFT_TO_RIGHT_EMBEDDING** -- This is the Strong bidirectional character type "LRE" in the Unicode specification.
- **static byte DIRECTIONALITY_LEFT_TO_RIGHT_OVERRIDE** -- This is the Strong bidirectional character type "LRO" in the Unicode specification.

- **static byte DIRECTIONALITY_NONSPACING_MARK** -- This is the Weak bidirectional character type "NSM" in the Unicode specification.
- **static byte DIRECTIONALITY_OTHER_NEUTRALS** -- This is the Neutral bidirectional character type "ON" in the Unicode specification.
- **static byte DIRECTIONALITY_PARAGRAPH_SEPARATOR** -- This is the Neutral bidirectional character type "B" in the Unicode specification.
- **static byte DIRECTIONALITY_POP_DIRECTIONAL_FORMAT** -- This is the Weak bidirectional character type "PDF" in the Unicode specification.
- **static byte DIRECTIONALITY_RIGHT_TO_LEFT** -- This is the Strong bidirectional character type "R" in the Unicode specification.
- **static byte DIRECTIONALITY_RIGHT_TO_LEFT_ARABIC** -- This is the Strong bidirectional character type "AL" in the Unicode specification.
- **static byte DIRECTIONALITY_RIGHT_TO_LEFT_EMBEDDING** -- This is the Strong bidirectional character type "RLE" in the Unicode specification.
- **static byte DIRECTIONALITY_RIGHT_TO_LEFT_OVERRIDE** -- This is the Strong bidirectional character type "RLO" in the Unicode specification.
- **static byte DIRECTIONALITY_SEGMENT_SEPARATOR** -- This is the Neutral bidirectional character type "S" in the Unicode specification.
- **static byte DIRECTIONALITY_UNDEFINED** -- This is the Undefined bidirectional character type.
- **static byte DIRECTIONALITY_WHITESPACE** -- This is the Neutral bidirectional character type "WS" in the Unicode specification.
- **static byte ENCLOSING_MARK** -- This is the General category "Me" in the Unicode specification.
- **static byte END_PUNCTUATION** -- This is the General category "Pe" in the Unicode specification.
- **static byte FINAL_QUOTE_PUNCTUATION** -- This is the General category "Pf" in the Unicode specification.
- **static byte FORMAT** -- This is the General category "Cf" in the Unicode specification.
- **static byte INITIAL_QUOTE_PUNCTUATION** -- This is the General category "Pi" in the Unicode specification.
- **static byte LETTER_NUMBER** -- This is the General category "Nl" in the Unicode specification.
- **static byte LINE_SEPARATOR** -- This is the General category "Zl" in the Unicode specification.
- **static byte LOWERCASE_LETTER** -- This is the General category "Ll" in the Unicode specification.
- **static byte MATH_SYMBOL** -- This is the General category "Sm" in the Unicode specification.
- **static int MAX_CODE_POINT** -- This is the maximum value of a Unicode code point.
- **static char MAX_HIGH_SURROGATE** -- This is the maximum value of a Unicode high-surrogate code unit in the UTF-16 encoding.
- **static char MAX_LOW_SURROGATE** -- This is the maximum value of a Unicode low-surrogate code unit in the UTF-16 encoding.
- **static int MAX_RADIX** -- This is the maximum radix available for conversion to and from

strings.

- **static char MAX_SURROGATE** -- This is the maximum value of a Unicode surrogate code unit in the UTF-16 encoding.
- **static char MAX_VALUE** -- This is the constant value of this field is the largest value of type char, '\uFFFF'.
- **static int MIN_CODE_POINT** -- This is the minimum value of a Unicode code poin
- **static char MIN_HIGH_SURROGATE** -- This is the minimum value of a Unicode high-surrogate code unit in the UTF-16 encoding.
- **static char MIN_LOW_SURROGATE** -- This is the minimum value of a Unicode low-surrogate code unit in the UTF-16 encoding.
- **static int MIN_RADIX** -- This is the minimum radix available for conversion to and from strings.
- **static int MIN_SUPPLEMENTARY_CODE_POINT** -- This is the minimum value of a supplementary code point.
- **static char MIN_SURROGATE** -- This is the minimum value of a Unicode surrogate code unit in the UTF-16 encoding.
- **static char MIN_VALUE** -- This is the constant value of this field is the smallest value of type char, '\u0000'.
- **static byte MODIFIER_LETTER** -- This is the General category "Lm" in the Unicode specification.
- **static byte MODIFIER_SYMBOL** -- This is the General category "Sk" in the Unicode specification.
- **static byte NON_SPACING_MARK** -- This is the General category "Mn" in the Unicode specification.
- **static byte OTHER_LETTER** -- This is the General category "Lo" in the Unicode specification.
- **static byte OTHER_NUMBER** -- This is the General category "No" in the Unicode specification.
- **static byte OTHER_PUNCTUATION** -- This is the General category "Po" in the Unicode specification.
- **static byte OTHER_SYMBOL** -- This is the General category "So" in the Unicode specification.
- **static byte PARAGRAPH_SEPARATOR** -- This is the General category "Zp" in the Unicode specification.
- **static byte PRIVATE_USE** -- This is the General category "Co" in the Unicode specification.
- **static int SIZE** -- This is the number of bits used to represent a char value in unsigned binary form.
- **static byte SPACE_SEPARATOR** -- This is the General category "Zs" in the Unicode specification.
- **static byte START_PUNCTUATION** -- This is the General category "Ps" in the Unicode specification.
- **static byte SURROGATE** -- This is the General category "Cs" in the Unicode specification.
- **static byte TITLECASE_LETTER** -- This is the General category "Lt" in the Unicode specification.

- **static Class<Character> TYPE** -- This is the Class instance representing the primitive type char.
- **static byte UNASSIGNED** -- This is the General category "Cn" in the Unicode specification.
- **static byte UPPERCASE_LETTER** -- This is the General category "Lu" in the Unicode specification.

Class constructors

S.N.	Constructor & Description
1	**Character***charvalue* This constructs a newly allocated Character object that represents the specified char value.

Class methods

S.N.	Method & Description
1	static int charCount*intcodePoint* This method determines the number of char values needed to represent the specified character *Unicodecodepoint*.
2	char charValue This method returns the value of this Character object.
3	static int codePointAt*char[]a, intindex* This method returns the code point at the given index of the char array.
4	static int codePointAt*char[]a, intindex, intlimit* 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.
5	static int codePointAt*CharSequenceseq, intindex* This method returns the code point at the given index of the CharSequence.
6	static int codePointBefore*char[]a, intindex* This method returns the code point preceding the given index of the char array.
7	static int codePointBefore*char[]a, intindex, intstart* 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.

8

static int codePointBefore*CharSequenceseq, intindex*

This method returns the code point preceding the given index of the CharSequence.

9

static int codePointCount*char[]a, intoffset, intcount*

This method returns the number of Unicode code points in a subarray of the char array argument

10

static int codePointCount*CharSequenceseq, intbeginIndex, intendIndex*

This method returns the number of Unicode code points in the text range of the specified char sequence.

11

int compareTo*CharacteranotherCharacter*

This method compares two Character objects numerically.

12

static int digit*charch, intradix*

This method returns the numeric value of the character ch in the specified radix.

13

static int digit*intcodePoint, intradix*

This method returns the numeric value of the specified character *Unicodecodepoint* in the specified radix.

14

boolean equals*Objectobj*

This method compares this object against the specified object

15

static char forDigit*intdigit, intradix*

This method determines the character representation for a specific digit in the specified radix.

16

static byte getDirectionality*charch*

This method returns the Unicode directionality property for the given character.

17

static byte getDirectionality*intcodePoint*

This method returns the Unicode directionality property for the given character *Unicodecodepoint*.

18

static int getNumericValue*charch*

This method returns the int value that the specified Unicode character represents.

19

static int getNumericValue*intcodePoint*

This method returns the int value that the specified character *Unicodecodepoint* represents.

20

static int getType*charch*

This method returns a value indicating a character's general category.

21

static int getType*intcodePoint*

This method returns a value indicating a character's general category.

22

int hashCode

This method returns a hash code for this Character.

23

static boolean isDefined*charch*

This method determines if a character is defined in Unicode.

24

static boolean isDefined*intcodePoint*

This method determines if a character *Unicodecodepoint* is defined in Unicode.

25

static boolean isDigit*charch*

This method determines if the specified character is a digit.

26

static boolean isDigit*intcodePoint*

This method determines if the specified character *Unicodecodepoint* is a digit.

27

static boolean isHighSurrogate*charch*

This method determines if the given char value is a high-surrogate code unit *alsoknownasleading − surrogatecodeunit*.

28

static boolean isIdentifierIgnorable*charch*

This method determines if the specified character should be regarded as an ignorable character in a Java identifier or a Unicode identifier.

29

static boolean isIdentifierIgnorable*intcodePoint*

This method determines if the specified character *Unicodecodepoint* should be regarded as an ignorable character in a Java identifier or a Unicode identifier.

30

static boolean isISOControl*charch*

This method determines if the specified character is an ISO control character.

31

static boolean isISOControl*intcodePoint*

This method determines if the referenced character *Unicodecodepoint* is an ISO control character.

32

static boolean isJavaIdentifierPart*charch*

This method determines if the specified character may be part of a Java identifier as other than the first character.

33

static boolean isJavaIdentifierPart*intcodePoint*

This method determines if the character *Unicodecodepoint* may be part of a Java identifier as other than the first character.

34

static boolean isJavaIdentifierStart*charch*

This method determines if the specified character is permissible as the first character in a Java identifier.

35

static boolean isJavaIdentifierStart*intcodePoint*

This method determines if the character *Unicodecodepoint* is permissible as the first character in a Java identifier.

36

static boolean isLetter*charch*

This method determines if the specified character is a letter.

37

static boolean isLetter*intcodePoint*

This method determines if the specified character *Unicodecodepoint* is a letter.

38

static boolean isLetterOrDigit*charch*

This method determines if the specified character is a letter or digit.

39

static boolean isLetterOrDigit*intcodePoint*

This method determines if the specified character *Unicodecodepoint* is a letter or digit.

40

static boolean isLowerCase*charch*

This method determines if the specified character is a lowercase character.

41

static boolean isLowerCase*intcodePoint*

This method determines if the specified character *Unicodecodepoint* is a lowercase character.

42

static boolean isLowSurrogate*charch*

This method determines if the given char value is a low-surrogate code unit *alsoknownastrailing − surrogatecodeunit*.

43

static boolean isMirrored*charch*

This method determines whether the character is mirrored according to the Unicode specification.

44

static boolean isMirrored*intcodePoint*

This method determines whether the specified character *Unicodecodepoint* is mirrored according to the Unicode specification.

45

static boolean isSpaceChar*charch*

This method determines if the specified character is a Unicode space character.

46

static boolean isSpaceChar*intcodePoint*

This method determines if the specified character *Unicodecodepoint* is a Unicode space character.

47

static boolean isSupplementaryCodePoint*intcodePoint*

This method determines whether the specified character *Unicodecodepoint* is in the supplementary character range.

48

static boolean isSurrogatePair*charhigh, charlow*

This method determines whether the specified pair of char values is a valid surrogate pair.

49

static boolean isTitleCase*charch*

This method determines if the specified character is a titlecase character.

50

static boolean isTitleCase*intcodePoint*

This method determines if the specified character *Unicodecodepoint* is a titlecase character.

51

static boolean isUnicodeIdentifierPart*charch*

This method determines if the specified character may be part of a Unicode identifier as other than the first character.

52

static boolean isUnicodeIdentifierPart*intcodePoint*

This method determines if the specified character *Unicodecodepoint* may be part of a Unicode identifier as other than the first character.

53

static boolean isUnicodeIdentifierStart*charch*

This method determines if the specified character is permissible as the first character in a Unicode identifier.

54

static boolean isUnicodeIdentifierStart*intcodePoint*

This method determines if the specified character *Unicodecodepoint* is permissible as the first character in a Unicode identifier.

55

static boolean isUpperCase(char ch

This method determines if the specified character is an uppercase character.

56

static boolean isUpperCase*intcodePoint*

This method determines if the specified character *Unicodecodepoint* is an uppercase character.

57

static boolean isValidCodePoint*intcodePoint*

This method determines whether the specified code point is a valid Unicode code point value in the range of 0x0000 to 0x10FFFF inclusive.

58

static boolean isWhitespace*charch*

This method determines if the specified character is white space according to Java.

59

static boolean isWhitespace*intcodePoint*

This method determines if the specified character *Unicodecodepoint* is white space according to Java.

60

static int offsetByCodePoints*char[]a, intstart, intcount, intindex, intcodePointOffset*

This method returns the index within the given char subarray that is offset from the given index by codePointOffset code points

61

static int offsetByCodePoints*CharSequenceseq, intindex, intcodePointOffset*

This method returns the index within the given char sequence that is offset from the given index by codePointOffset code points.

62

static char reverseBytes*charch*

This method returns the value obtained by reversing the order of the bytes in the specified char value.

63

static char[] toChars*intcodePoint*

This method converts the specified character *Unicodecodepoint* to its UTF-16 representation stored in a char array.

64

static int toChars*intcodePoint, char[]dst, intdstIndex*

This method converts the specified character *Unicodecodepoint* to its UTF-16 representation.

65

static int toCodePoint*charhigh, charlow*

This method converts the specified surrogate pair to its supplementary code point value.

66

static char toLowerCase*charch*

This method converts the character argument to lowercase using case mapping information from the UnicodeData file.

67

static int toLowerCase*intcodePoint*

This method converts the character *Unicodecodepoint* argument to lowercase using case mapping information from the UnicodeData file.

68

String toString

This method returns a String object representing this Character's value.

69

static String toString*charc*

This method returns a String object representing the specified char.

70

static char toTitleCase*charch*

This method converts the character argument to titlecase using case mapping information from the UnicodeData file.

71

static int toTitleCase*intcodePoint*

This method converts the character *Unicodecodepoint* argument to titlecase using case mapping information from the UnicodeData file.

72

static char toUpperCase*charch*

This method converts the character argument to uppercase using case mapping information from the UnicodeData file.

73

static int toUpperCase*intcodePoint*

This method converts the character *Unicodecodepoint* argument to uppercase using case mapping information from the UnicodeData file.

74

static Character valueOf*charc*

This method returns a Character instance representing the specified char value.

Methods inherited

This class inherits methods from the following classes:

- java.lang.Object