java.lang

Class String

java.lang.Object
    java.lang.String

All Implemented Interfaces:

Serializable, CharSequence, Comparable<String>

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

The `String` class represents character strings. All string literals in Java programs, such as `"abc"`, are implemented as instances of this class.

Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because String objects are immutable they can be shared. For example:

```
String str = "abc";
```

is equivalent to:

```
char data[] = {'a', 'b', 'c'};
String str = new String(data);
```

Here are some more examples of how strings can be used:

```
System.out.println("abc");
String cde = "cde";
System.out.println("abc" + cde);
String c = "abc".substring(2,3);
String d = cde.substring(1, 2);
```

The class `String` includes methods for examining individual characters of the sequence, for comparing strings, for searching strings, for extracting substrings, and for creating a copy of a string with all characters translated to uppercase or to lowercase. Case mapping is based on the Unicode Standard version specified by the `Character` class.

The Java language provides special support for the string concatenation operator (+), and for conversion of other objects to strings. String concatenation is implemented through the `StringBuilder`(or `StringBuffer`) class and its `append` method. String conversions are implemented through the method `toString`, defined by `Object` and inherited by all classes in Java. For additional information on string concatenation and conversion, see Gosling, Joy, and Steele, *The Java Language Specification*.

Unless otherwise noted, passing a `null` argument to a constructor or method in this class will cause a `NullPointerException` to be thrown.

A `String` represents a string in the UTF-16 format in which *supplementary characters* are represented by *surrogate pairs* (see the section Unicode Character Representations in the `Character` class for more information). Index values refer to `char` code units, so a supplementary character uses two positions in a `String`.

The `String` class provides methods for dealing with Unicode code points (i.e., characters), in addition to those for dealing with Unicode code units (i.e., `char` values).

Since:

JDK1.0

See Also:

`Object.toString()`, `StringBuffer`, `StringBuilder`, `Charset`, Serialized Form

Field Summary

Fields

Modifier and Type	Field and Description
`static Comparator<String>`	**`CASE_INSENSITIVE_ORDER`** A Comparator that orders `String` objects as by `compareToIgnoreCase`.

Constructor Summary

Constructors

Constructor and Description
`String``()` Initializes a newly created `String` object so that it represents an empty character sequence.
`String``(byte[] bytes)` Constructs a new `String` by decoding the specified array of bytes using the platform's default charset.
`String``(byte[] bytes, Charset charset)` Constructs a new `String` by decoding the specified array of bytes using the specified **charset**.
`String``(byte[] ascii, int hibyte)` **Deprecated.** *This method does not properly convert bytes into characters. As of JDK 1.1, the preferred way to do this is via the `String` constructors that take a **`Charset`**, charset name, or that use the platform's default charset.*
`String``(byte[] bytes, int offset, int length)` Constructs a new `String` by decoding the specified subarray of bytes using the platform's default charset.
`String``(byte[] bytes, int offset, int length, Charset charset)` Constructs a new `String` by decoding the specified subarray of bytes using the specified **charset**.
`String``(byte[] ascii, int hibyte, int offset, int count)` **Deprecated.** *This method does not properly convert bytes into characters. As of JDK 1.1, the preferred way to do this is via the `String` constructors that take a **`Charset`**, charset name, or that use the platform's default charset.*
`String``(byte[] bytes, int offset, int length, String charsetName)` Constructs a new `String` by decoding the specified subarray of bytes using the specified charset.
`String``(byte[] bytes, String charsetName)` Constructs a new `String` by decoding the specified array of bytes using the specified **charset**.
`String``(char[] value)` Allocates a new `String` so that it represents the sequence of characters currently contained in the character array argument.
`String``(char[] value, int offset, int count)` Allocates a new `String` that contains characters from a subarray of the character array argument.
`String``(int[] codePoints, int offset, int count)` Allocates a new `String` that contains characters from a subarray of the **Unicode code point** array argument.
`String``(String original)` Initializes a newly created `String` object so that it represents the same sequence of characters as the argument; in other words, the newly created string is a copy of the argument string.
`String``(StringBuffer buffer)` Allocates a new string that contains the sequence of characters currently contained in the string buffer argument.
`String``(StringBuilder builder)`

Allocates a new string that contains the sequence of characters currently contained in the string builder argument.

Method Summary

Methods

Modifier and Type	Method and Description
char	**charAt**(int index) Returns the char value at the specified index.
int	**codePointAt**(int index) Returns the character (Unicode code point) at the specified index.
int	**codePointBefore**(int index) Returns the character (Unicode code point) before the specified index.
int	**codePointCount**(int beginIndex, int endIndex) Returns the number of Unicode code points in the specified text range of this String.
int	**compareTo**(**String** anotherString) Compares two strings lexicographically.
int	**compareToIgnoreCase**(**String** str) Compares two strings lexicographically, ignoring case differences.
String	**concat**(**String** str) Concatenates the specified string to the end of this string.
boolean	**contains**(**CharSequence** s) Returns true if and only if this string contains the specified sequence of char values.
boolean	**contentEquals**(**CharSequence** cs) Compares this string to the specified CharSequence.
boolean	**contentEquals**(**StringBuffer** sb) Compares this string to the specified StringBuffer.
static **String**	**copyValueOf**(char[] data) Returns a String that represents the character sequence in the array specified.
static **String**	**copyValueOf**(char[] data, int offset, int count) Returns a String that represents the character sequence in the array specified.
boolean	**endsWith**(**String** suffix) Tests if this string ends with the specified suffix.
boolean	**equals**(**Object** anObject) Compares this string to the specified object.
boolean	**equalsIgnoreCase**(**String** anotherString) Compares this String to another String, ignoring case considerations.
static **String**	**format**(**Locale** l, **String** format, **Object**... args) Returns a formatted string using the specified locale, format string, and arguments.
static **String**	**format**(**String** format, **Object**... args) Returns a formatted string using the specified format string and arguments.
byte[]	**getBytes**() Encodes this String into a sequence of bytes using the platform's default charset, storing the result into a new byte array.
byte[]	**getBytes**(**Charset** charset) Encodes this String into a sequence of bytes using the given **charset**, storing the result into a new byte array.
void	**getBytes**(int srcBegin, int srcEnd, byte[] dst, int dstBegin) **Deprecated.** *This method does not properly convert characters into bytes. As of JDK 1.1, the preferred way to do this is via the **getBytes()** method, which uses the platform's default charset.*
byte[]	**getBytes**(**String** charsetName)

	Encodes this `String` into a sequence of bytes using the named charset, storing the result into a new byte array.
`void`	**`getChars`**`(int srcBegin, int srcEnd, char[] dst, int dstBegin)` Copies characters from this string into the destination character array.
`int`	**`hashCode`**`()` Returns a hash code for this string.
`int`	**`indexOf`**`(int ch)` Returns the index within this string of the first occurrence of the specified character.
`int`	**`indexOf`**`(int ch, int fromIndex)` Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.
`int`	**`indexOf`**`(`**`String`**` str)` Returns the index within this string of the first occurrence of the specified substring.
`int`	**`indexOf`**`(`**`String`**` str, int fromIndex)` Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.
`String`	**`intern`**`()` Returns a canonical representation for the string object.
`boolean`	**`isEmpty`**`()` Returns `true` if, and only if, **`length()`** is `0`.
`int`	**`lastIndexOf`**`(int ch)` Returns the index within this string of the last occurrence of the specified character.
`int`	**`lastIndexOf`**`(int ch, int fromIndex)` Returns the index within this string of the last occurrence of the specified character, searching backward starting at the specified index.
`int`	**`lastIndexOf`**`(`**`String`**` str)` Returns the index within this string of the last occurrence of the specified substring.
`int`	**`lastIndexOf`**`(`**`String`**` str, int fromIndex)` Returns the index within this string of the last occurrence of the specified substring, searching backward starting at the specified index.
`int`	**`length`**`()` Returns the length of this string.
`boolean`	**`matches`**`(`**`String`**` regex)` Tells whether or not this string matches the given **regular expression**.
`int`	**`offsetByCodePoints`**`(int index, int codePointOffset)` Returns the index within this `String` that is offset from the given `index` by `codePointOffset` code points.
`boolean`	**`regionMatches`**`(boolean ignoreCase, int toffset, `**`String`**` other, int ooffset, int len)` Tests if two string regions are equal.
`boolean`	**`regionMatches`**`(int toffset, `**`String`**` other, int ooffset, int len)` Tests if two string regions are equal.
`String`	**`replace`**`(char oldChar, char newChar)` Returns a new string resulting from replacing all occurrences of `oldChar` in this string with `newChar`.
`String`	**`replace`**`(`**`CharSequence`**` target, `**`CharSequence`**` replacement)` Replaces each substring of this string that matches the literal target sequence with the specified literal replacement sequence.
`String`	**`replaceAll`**`(`**`String`**` regex, `**`String`**` replacement)` Replaces each substring of this string that matches the given **regular expression** with the given replacement.
`String`	**`replaceFirst`**`(`**`String`**` regex, `**`String`**` replacement)` Replaces the first substring of this string that matches the given **regular expression** with the given replacement.
`String``[]`	**`split`**`(`**`String`**` regex)`

	Splits this string around matches of the given **regular expression**.
`String[]`	**`split`**`(String regex, int limit)` Splits this string around matches of the given **regular expression**.
`boolean`	**`startsWith`**`(String prefix)` Tests if this string starts with the specified prefix.
`boolean`	**`startsWith`**`(String prefix, int toffset)` Tests if the substring of this string beginning at the specified index starts with the specified prefix.
`CharSequence`	**`subSequence`**`(int beginIndex, int endIndex)` Returns a new character sequence that is a subsequence of this sequence.
`String`	**`substring`**`(int beginIndex)` Returns a new string that is a substring of this string.
`String`	**`substring`**`(int beginIndex, int endIndex)` Returns a new string that is a substring of this string.
`char[]`	**`toCharArray`**`()` Converts this string to a new character array.
`String`	**`toLowerCase`**`()` Converts all of the characters in this `String` to lower case using the rules of the default locale.
`String`	**`toLowerCase`**`(Locale locale)` Converts all of the characters in this `String` to lower case using the rules of the given `Locale`.
`String`	**`toString`**`()` This object (which is already a string!) is itself returned.
`String`	**`toUpperCase`**`()` Converts all of the characters in this `String` to upper case using the rules of the default locale.
`String`	**`toUpperCase`**`(Locale locale)` Converts all of the characters in this `String` to upper case using the rules of the given `Locale`.
`String`	**`trim`**`()` Returns a copy of the string, with leading and trailing whitespace omitted.
`static String`	**`valueOf`**`(boolean b)` Returns the string representation of the `boolean` argument.
`static String`	**`valueOf`**`(char c)` Returns the string representation of the `char` argument.
`static String`	**`valueOf`**`(char[] data)` Returns the string representation of the `char` array argument.
`static String`	**`valueOf`**`(char[] data, int offset, int count)` Returns the string representation of a specific subarray of the `char` array argument.
`static String`	**`valueOf`**`(double d)` Returns the string representation of the `double` argument.
`static String`	**`valueOf`**`(float f)` Returns the string representation of the `float` argument.
`static String`	**`valueOf`**`(int i)` Returns the string representation of the `int` argument.
`static String`	**`valueOf`**`(long l)` Returns the string representation of the `long` argument.
`static String`	**`valueOf`**`(Object obj)` Returns the string representation of the `Object` argument.

Methods inherited from class java.lang.Object

`clone`, `finalize`, `getClass`, `notify`, `notifyAll`, `wait`, `wait`, `wait`