## String Handling

Contributed by Wilbur Allison  Rate Article ★★★★☆

The String class is defined in the java.lang package and hence is implicitly available to all the programs in Java. The String class is declared as final, which means that it cannot be subclassed. It extends the Object class and implements the Serializable, Comparable, and CharSequence interfaces.

Java implements strings as objects of type String. A string is a sequence of characters. Unlike most of the other languages, Java treats a string as a single value rather than as an array of characters.

The String objects are immutable, i.e., once an object of the String class is created, the string it contains cannot be changed. In other words, once a String object is created, the characters that comprise the string cannot be changed. Whenever any operation is performed on a String object, a new String object will be created while the original contents of the object will remain unchanged. However, at any time, a variable declared as a String reference can be changed to point to some other String object.

Constructors defined in the String class

The String class defines several constructors. The most common constructor of the String class is the one given below:

public String(String value)

This constructor constructs a new String object initialized with the same sequence of the characters passed as the argument. In other words, the newly created String object is the copy of the string passed as an argument to the constructor.

Other constructors defined in the String class are as follows:

public String()

This constructor creates an empty String object. However, the use of this constructor is unnecessary because String objects are immutable.

public String(char[] value)

This constructor creates a new String object initialized with the same sequence of characters currently contained in the array that is passed as the argument to it.

public String(char[] value, int startindex, int len)

This constructor creates a new String object initialized with the same sequence of characters currently contained in the subarray. This subarray is derived from the character array and the two integer values that are passed as arguments to the constructor. The int variable startindex represents the index value of the starting character of the subarray, and the int variable len represents the number of characters to be used to form the new String object.

public String(StringBuffer sbf)

This constructor creates a new String object that contains the same sequence of characters currently contained in the string buffer argument.

public String(byte[] asciichars)

The array of bytes that is passed as an argument to the constructor contains the ASCII character set. Therefore, this array of bytes is first decoded using the default charset of the platform. Then the constructor creates a new String object initialized with same sequence of characters obtained after decoding the array.

public String(byte[] asciiChars, int startindex, int len)

This constructor creates the String object after decoding the array of bytes and by using the subarray of bytes.

Special String Operations

Finding the length of string

The String class defines the length() method that determines the length of a string. The length of a string is the number of characters contained in the string. The signature of the length()

method is given below:

public int length()

String Concatenation using the + operator

The + operator is used to concatenate two strings, producing a new String object as the result. For example,

```
String sale = '500';
String s = 'Our daily sale is' + sale + 'dollars';
System.out.println(s);
```

This code will display the string 'Our daily sale is 500 dollars'.

The + operator may also be used to concatenate a string with other data types. For example,

```
int sale = 500;
String s = 'Our daily sale is' + sale + 'dollars';
System.out.println(s);
```

This code will display the string 'Our daily sale is 500 dollars'. In this case, the variable sale is declared as int rather than String, but the output produced is the same. This is because the int value contained in the variable sale is automatically converted to String type, and then the + operator concatenates the two strings.

String Comparison

The String class defines various methods that are used to compare strings or substrings within strings. Each of them is discussed in the following sections:

equals()

The equals() method is used to check whether the Object that is passed as the argument to the method is equal to the String object that invokes the method. It returns true if and only if the argument is a String object that represents the same sequence of characters as represented by the invoking object. The signature of the equals() method is as follows:

public boolean equals(Object str)

equalsIgnoreCase()

The equalsIgnoreCase() method is used to check the equality of the two String objects without taking into consideration the case of the characters contained in the two strings. It returns true if the two strings are of the same length and if the corresponding characters in the two strings are the same ignoring case. The signature of the equalsIgnoreCase() method is:

public boolean equalsIgnoreCase(Object str)

compareTo()

The compareTo() method is used in conditions where a Programmer wants to sort a list of strings in a predetermined order. The compareTo() method checks whether the string passed as an argument to the method is less than, greater than, or equal to the invoking string. A string is considered less than another string if it comes before it in alphabetical order. The signature of the compareTo() method is as follows:

public int compareTo(String str)

where, str is the String being compared to the invoking String. The compareTo() method returns an int value as the result of String comparison. The meaning of these values are given in the following table:

| Value | Meaning |
|---|---|
| Less than zero | The invoking string is less than the argument string. |
| Zero | The invoking string and the argument string are same. |

| | |
|---|---|
| Greater than zero | The invoking string is greater than the argument string. |

The String class also has the compareToIgnoreCase() method that compares two strings without taking into consideration their case difference. The signature of the method is given below:

public int compareToIgnoreCase(String str)

regionMatches()

The regionMatches() method is used to check the equality of two string regions where the two string regions belong to two different strings. The signature of the method is given below:

public boolean regionMatches(int startindex, String str2, int startindex2, int len)

There is also an overloaded version of the method that tests the equality of the substring ignoring the case of characters in the substring. Its signature is given below:

public boolean regionMatches(boolean ignoreCase, int startindex, String str2, int startindex2, int len)

In both signatures of the method, startindex specifies the starting index of the substring within the invoking string. The str2 argument specifies the string to be compared. The startindex2 specifies the starting index of the substring within the string to be compared. The len argument specifies the length of the substring being compared. However, in the latter signature of the method, the comparison is done ignoring the case of the characters in the substring only if the ignoreCase argument is true.

startsWith()

The startsWith() method is used to check whether the invoking string starts with the same sequence of characters as the substring passed as an argument to the method. The signature of the method is given below:

public boolean startsWith(String prefix)

There is also an overloaded version of the startsWith() method with the following signature:

public boolean startsWith(String prefix, int startindex)

In both signatures of the method given above, the prefix denotes the substring to be matched within the invoking string. However, in the second version, the startindex denotes the starting index into the invoking string at which the search operation will commence.

endsWith()

The endsWith() method is used to check whether the invoking string ends with the same sequence of characters as the substring passed as an argument to the method. The signature of the method is given below:

public boolean endsWith(String prefix)

Modifying a String

The String objects are immutable. Therefore, it is not possible to change the original contents of a string. However, the following String methods can be used to create a new copy of the string with the required modification:

substring()

The substring() method creates a new string that is the substring of the string that invokes the method. The method has two forms:

public String substring(int startindex)

public String substring(int startindex, int endindex)

where, startindex specifies the index at which the substring will begin and endindex specifies the index at which the substring will end. In the first form where the endindex is not present, the substring begins at startindex and runs till the end of the invoking string.

Concat()

The concat() method creates a new string after concatenating the argument string to the end of the invoking string. The signature of the method is given below:

public String concat(String str)

replace()

The replace() method creates a new string after replacing all the occurrences of a particular character in the string with another character. The string that invokes this method remains unchanged. The general form of the method is given below:

public String replace(char old_char, char new_char)

trim()

The trim() method creates a new copy of the string after removing any leading and trailing whitespace. The signature of the method is given below:

public String trim(String str)

toUpperCase()

The toUpperCase() method creates a new copy of a string after converting all the lowercase letters in the invoking string to uppercase. The signature of the method is given below:

public String toUpperCase()

toLowerCase()

The toLowerCase() method creates a new copy of a string after converting all the uppercase letters in the invoking string to lowercase. The signature of the method is given below:

public String toLowerCase()

Searching Strings

The String class defines two methods that facilitate in searching a particular character or sequence of characters in a string. They are as follows:

IndexOf()

The indexOf() method searches for the first occurrence of a character or a substring in the invoking string. If a match is found, then the method returns the index at which the character or the substring first appears. Otherwise, it returns -1. The indexOf() method has the following signatures:

public int indexOf(int ch)

public int indexOf(int ch, int startindex)

public int indexOf(String str)

public int indexOf(String str, int startindex)

lastIndexOf()

The lastIndexOf() method searches for the last occurrence of a character or a substring in the invoking string. If a match is found, then the method returns the index at which the character

or the substring last appears. Otherwise, it returns -1. The lastIndexOf() method has the following signatures:

public int lastIndexOf(int ch)

public int lastIndexOf (int ch, int startindex)

public int lastIndexOf (String str)

public int lastIndexOf (String str, int startindex)