Java – String Class and its methods explained with examples

BY CHAITANYA SINGH | FILED UNDER: STRING HANDLING

String is a sequence of characters, for e.g. “Hello” is a string of 5 characters. In java, string is an immutable object which means it is constant and can cannot be changed once it has been created. In this tutorial we will learn about String class and String methods in detail along with many other Java String tutorials.

Creating a String

There are two ways to create a String in Java

1. String literal
2. Using new keyword

String literal

In java, Strings can be created like this: Assigning a String literal to a String instance:

```
String str1 = "Welcome";
String str2 = "Welcome";
```

The problem with this approach: As I stated in the beginning that String is an object in Java. However we have not created any string object using new keyword above. The compiler does that task for us it creates a string object having the string literal (that we have provided , in this case it is “Welcome”) and assigns it to the provided string instances.

But if the object already exist in the memory it does not create a new Object rather it assigns the same old object to the new instance, that means even though we have two string instances above(str1 and str2) compiler only created on string object (having the value “Welcome”) and assigned the same to both the instances. For example there are 10 string instances that have same value, it means that in memory there is only one object having the value and all the 10 string instances would be pointing to the same object.

What if we want to have two different object with the same string? For that we would need to create strings using **new keyword**.

Using New Keyword

As we saw above that when we tried to assign the same string object to two different literals, compiler only created one object and made both of the literals to point the same object. To overcome that approach we can create strings like this:

```
String str1 = new String("Welcome");
String str2 = new String("Welcome");
```

In this case compiler would create two different object in memory having the same string.

A Simple Java String Example

```
public class Example{
   public static void main(String args[]){
        //creating a string by java string literal
        String str = "Beginnersbook";
        char arrch[]={'h','e','l','l','o'};
        //converting char array arrch[] to string str2
        String str2 = new String(arrch);

        //creating another java string str3 by using new keyword
        String str3 = new String("Java String Example");

        //Displaying all the three strings
        System.out.println(str);
        System.out.println(str2);
        System.out.println(str3);
   }
}
```

Output:

```
Beginnersbook
hello
Java String Example
```

Java String Methods

Here are the list of the methods available in the Java String class. These methods are explained in the separate tutorials with the help of examples. Links to the tutorials are provided below:

1. char charAt(int index): It returns the character at the specified index. Specified index value should be between 0 to length() -1 both inclusive. It throws IndexOutOfBoundsException if index<0||>= length of String.
2. boolean equals(Object obj): Compares the string with the specified string and returns true if both matches else false.

3. boolean equalsIgnoreCase(String string): It works same as equals method but it doesn't consider the case while comparing strings. It does a case insensitive comparison.
4. int compareTo(String string): This method compares the two strings based on the Unicode value of each character in the strings.
5. int compareToIgnoreCase(String string): Same as CompareTo method however it ignores the case during comparison.
6. boolean startsWith(String prefix, int offset): It checks whether the substring (starting from the specified offset index) is having the specified prefix or not.
7. boolean startsWith(String prefix): It tests whether the string is having specified prefix, if yes then it returns true else false.
8. boolean endsWith(String suffix): Checks whether the string ends with the specified suffix.
9. int hashCode(): It returns the hash code of the string.
10. int indexOf(int ch): Returns the index of first occurrence of the specified character ch in the string.
11. int indexOf(int ch, int fromIndex): Same as indexOf method however it starts searching in the string from the specified fromIndex.
12. int lastIndexOf(int ch): It returns the last occurrence of the character ch in the string.
13. int lastIndexOf(int ch, int fromIndex): Same as lastIndexOf(int ch) method, it starts search from fromIndex.
14. int indexOf(String str): This method returns the index of first occurrence of specified substring str.
15. int lastindexOf(String str): Returns the index of last occurrence of string str.
16. String substring(int beginIndex): It returns the substring of the string. The substring starts with the character at the specified index.
17. String substring(int beginIndex, int endIndex): Returns the substring. The substring starts with character at beginIndex and ends with the character at endIndex.
18. String concat(String str): Concatenates the specified string "str" at the end of the string.
19. String replace(char oldChar, char newChar): It returns the new updated string after changing all the occurrences of oldChar with the newChar.
20. boolean contains(CharSequence s): It checks whether the string contains the specified sequence of char values. If yes then it returns true else false. It throws NullPointerException of 's' is null.
21. String toUpperCase(Locale locale): Converts the string to upper case string using the rules defined by specified locale.
22. String toUpperCase(): Equivalent to toUpperCase(Locale.getDefault()).
23. public String intern(): This method searches the specified string in the memory pool and if it is found then it returns the reference of it, else it allocates the memory space to the specified string and assign the reference to it.

24. public boolean isEmpty(): This method returns true if the given string has 0 length. If the length of the specified Java String is non-zero then it returns false.
25. public static String join(): This method joins the given strings using the specified delimiter and returns the concatenated Java String
26. String replaceFirst(String regex, String replacement): It replaces the first occurrence of substring that fits the given regular expression “regex” with the specified replacement string.
27. String replaceAll(String regex, String replacement): It replaces all the occurrences of substrings that fits the regular expression regex with the replacement string.
28. String[] split(String regex, int limit): It splits the string and returns the array of substrings that matches the given regular expression. limit is a result threshold here.
29. String[] split(String regex): Same as split(String regex, int limit) method however it does not have any threshold limit.
30. String toLowerCase(Locale locale): It converts the string to lower case string using the rules defined by given locale.
31. public static String format(): This method returns a formatted java String
32. String toLowerCase(): Equivalent to toLowerCase(Locale. getDefault()).
33. String trim(): Returns the substring after omitting leading and trailing white spaces from the original string.
34. char[] toCharArray(): Converts the string to a character array.
35. static String copyValueOf(char[] data): It returns a string that contains the characters of the specified character array.
36. static String copyValueOf(char[] data, int offset, int count): Same as above method with two extra arguments – initial offset of subarray and length of subarray.
37. void getChars(int srcBegin, int srcEnd, char[] dest, int destBegin): It copies the characters of **src**array to the **dest** array. Only the specified range is being copied(srcBegin to srcEnd) to the dest subarray(starting fromdestBegin).
38. static String valueOf(): This method returns a string representation of passed arguments such as int, long, float, double, char and char array.
39. boolean contentEquals(StringBuffer sb): It compares the string to the specified string buffer.
40. boolean regionMatches(int srcoffset, String dest, int destoffset, int len): It compares the substring of input to the substring of specified string.
41. boolean regionMatches(boolean ignoreCase, int srcoffset, String dest, int destoffset, int len): Another variation of regionMatches method with the extra boolean argument to specify whether the comparison is case sensitive or case insensitive.
42. byte[] getBytes(String charsetName): It converts the String into sequence of bytes using the specified charset encoding and returns the array of resulted bytes.

43. byte[] getBytes(): This method is similar to the above method it just uses the default charset encoding for converting the string into sequence of bytes.
44. int length(): It returns the length of a String.
45. boolean matches(String regex): It checks whether the String is matching with the specified regular expression regex.
46. int codePointAt(int index):It is similar to the charAt method however it returns the Unicode code point value of specified index rather than the character itself.