String in JAVA

In Java, the `String` class is used to represent a sequence of characters. In Java, the String class is used to represent a sequence of characters. It is part of the core Java library and is one of the most commonly used classes in Java programming.

Here are some key characteristics of the `String` class in Java:

1. Immutable: Strings in Java are immutable, which means that once a `String` object is created, its value cannot be changed. Any operation that appears to modify a string actually creates a new string object. This immutability ensures string integrity and allows for various optimizations.

2. String Pool: Java maintains a string pool, also known as the string literal pool, to improve memory efficiency. When you create a string using a string literal, Java checks if an equivalent string already exists in the pool. If it does, the existing string is reused instead of creating a new object.

3. Widely Used: The `String` class is one of the most widely used classes in Java. It provides extensive functionality and methods for string manipulation, searching, comparing, and more. Strings are fundamental for working with textual data and play a crucial role in Java applications.

4. Unicode Support: Strings in Java are encoded using the Unicode character set, which allows for representing a wide range of characters from different languages and scripts. This Unicode support ensures compatibility across different platforms and enables multilingual applications.

5. String Literal Syntax: In Java, you can create strings using string literals enclosed in double quotes (`"`) or by using the `new` keyword with the `String` constructor. String literals are convenient and widely used for creating strings.

6. String Concatenation: Java provides several ways to concatenate strings. You can use the `+` operator to concatenate strings, or you can use the `concat()` method of the `String` class. The `StringBuilder` or `StringBuffer` classes are also available for efficient string concatenation, especially when dealing with large or frequent concatenations.

7. String Equality: The `String` class overrides the `equals()` method to compare the contents of two strings. The `equals()` method checks if two strings have the same characters in the same order. You can also use the `compareTo()` method for lexicographic comparison of strings.

By: Sanjay Madaan
https://in.linkedin.com/in/sanjay-madaan-9569452043

8. String Manipulation: The `String` class provides various methods for manipulating strings, such as `substring()`, `replace()`, `toLowerCase()`, `toUpperCase()`, `trim()`, `split()`, and more. These methods allow you to extract substrings, replace characters, change case, remove whitespace, split strings into arrays, and perform other common string operations.

9. Performance Considerations: Since strings are immutable in Java, it is important to be mindful of performance when performing repeated string manipulations. Creating new string objects can consume memory and impact performance. In such cases, using the `StringBuilder` or `StringBuffer` classes can provide more efficient string manipulation capabilities.

It is one of the most commonly used classes in Java and provides several methods for manipulating strings.

Commonly used methods of the `String` class in Java:

1. `length()`: Returns the length of the string.

```java
String str = "Hello, World!";
int length = str.length(); // 13
```

2. `charAt(int index)`: Returns the character at the specified index.

```java
String str = "Hello";
char ch = str.charAt(0); // 'H'
```

3. `substring(int beginIndex)`: Returns a substring starting from the specified index.

```java
String str = "Hello, World!";
String substring = str.substring(7); // "World!"
```

4. `substring(int beginIndex, int endIndex)`: Returns a substring within the specified index range.

```java
String str = "Hello, World!";
```

```
String substring = str.substring(7, 12); // "World"
```

5. `concat(String str)`: Concatenates the specified string to the end of the current string.

```java
String str1 = "Hello";
String str2 = "World!";
String concatenated = str1.concat(str2); // "HelloWorld!"
```

6. `toLowerCase()`: Converts the string to lowercase.

```java
String str = "Hello, World!";
String lowercase = str.toLowerCase(); // "hello, world!"
```

7. `toUpperCase()`: Converts the string to uppercase.

```java
String str = "Hello, World!";
String uppercase = str.toUpperCase(); // "HELLO, WORLD!"
```

8. `replace(char oldChar, char newChar)`: Replaces all occurrences of a specified character with another character.

```java
String str = "Hello, World!";
String replaced = str.replace('o', 'a'); // "Hella, Warld!"
```

9. `startsWith(String prefix)`: Checks if the string starts with the specified prefix.

```java
String str = "Hello, World!";
boolean startsWithHello = str.startsWith("Hello"); // true
```

10. `endsWith(String suffix)`: Checks if the string ends with the specified suffix.

```java
String str = "Hello, World!";
boolean endsWithWorld = str.endsWith("World!"); // true
```

Commonly used string manipulation methods available in the `String` class in Java:

1. `trim()`: Removes leading and trailing whitespace from the string.

```java
String str = "   Hello, World!   ";
String trimmed = str.trim(); // "Hello, World!"
```

2. `toLowerCase()` and `toUpperCase()`: Converts the string to lowercase or uppercase.

```java
String str = "Hello, World!";
String lowercase = str.toLowerCase(); // "hello, world!"
String uppercase = str.toUpperCase(); // "HELLO, WORLD!"
```

3. `replace(CharSequence target, CharSequence replacement)`: Replaces all occurrences of a specified sequence with another sequence.

```java
String str = "Hello, World!";
String replaced = str.replace("o", "a"); // "Hella, Warld!"
```

4. `replaceAll(String regex, String replacement)`: Replaces all occurrences of a regular expression with a specified string.

```java
String str = "Hello, World!";
String replaced = str.replaceAll("[oO]", "a"); // "Hella, Warld!"
```

5. `split(String regex)`: Splits the string into an array of substrings based on a delimiter.

```java
String str = "Hello, World!";
String[] words = str.split(", "); // ["Hello", "World!"]
```

By: Sanjay Madaan
https://in.linkedin.com/in/sanjay-madaan-9569452043

6. `startsWith(String prefix)` and `endsWith(String suffix)`: Checks if the string starts with a specific prefix or ends with a specific suffix.

```java
String str = "Hello, World!";
boolean startsWithHello = str.startsWith("Hello"); // true
boolean endsWithWorld = str.endsWith("World!"); // true
```

7. `contains(CharSequence sequence)`: Checks if the string contains a specified sequence.

```java
String str = "Hello, World!";
boolean containsHello = str.contains("Hello"); // true
```

8. `indexOf(String str)` and `lastIndexOf(String str)`: Returns the index of the first or last occurrence of a specified substring.

```java
String str = "Hello, World!";
int firstIndex = str.indexOf("o"); // 4
int lastIndex = str.lastIndexOf("o"); // 8
```

9. `startsWith(String prefix, int offset)` and `endsWith(String suffix, int offset)`: Checks if the string starts or ends with a specific prefix or suffix, starting from a specified offset.

```java
String str = "Hello, World!";
boolean startsWithHello = str.startsWith("Hello", 0); // true
boolean endsWithWorld = str.endsWith("World!", 7); // true
```

By: Sanjay Madaan
https://in.linkedin.com/in/sanjay-madaan-9569452043

Commonly used string searching methods available in the `String` class in Java:

1. `indexOf(String str)`: Returns the index of the first occurrence of a specified substring within the string.

```java
String str = "Hello, World!";
int index = str.indexOf("World"); // 7
```

2. `lastIndexOf(String str)`: Returns the index of the last occurrence of a specified substring within the string.

```java
String str = "Hello, World!";
int lastIndex = str.lastIndexOf("o"); // 8
```

3. `startsWith(String prefix)`: Checks if the string starts with a specified prefix.

```java
String str = "Hello, World!";
boolean startsWithHello = str.startsWith("Hello"); // true
```

4. `endsWith(String suffix)`: Checks if the string ends with a specified suffix.

```java
String str = "Hello, World!";
boolean endsWithWorld = str.endsWith("World!"); // true
```

5. `contains(CharSequence sequence)`: Checks if the string contains a specified sequence.

```java
String str = "Hello, World!";
boolean containsHello = str.contains("Hello"); // true
```

6. `matches(String regex)`: Checks if the string matches a specified regular expression.

By: Sanjay Madaan
https://in.linkedin.com/in/sanjay-madaan-9569452043

```java
String str = "Hello, World!";
boolean matches = str.matches("Hello,.*"); // true
```

7. `regionMatches(boolean ignoreCase, int toffset, String other, int ooffset, int len)`: Compares a specific region of the string with another string.

```java
String str = "Hello, World!";
boolean matches = str.regionMatches(true, 0, "hello", 0, 5); // true
```

8. `equalsIgnoreCase(String anotherString)`: Compares the string to another string, ignoring case differences.

```java
String str = "Hello, World!";
boolean equalsIgnoreCase = str.equalsIgnoreCase("hello, world!"); // true
```

9. `startsWith(String prefix, int offset)`: Checks if the string starts with a specific prefix, starting from a specified offset.

```java
String str = "Hello, World!";
boolean startsWithWorld = str.startsWith("World", 7); // true
```

10. `endsWith(String suffix, int offset)`: Checks if the string ends with a specific suffix, starting from a specified offset.

```java
String str = "Hello, World!";
boolean endsWithHello = str.endsWith("Hello", 5); // true
```

In Java, the `String` class provides a `split()` method that allows you to split a string into an array of substrings based on a specified delimiter. Here's an example:

```java
String str = "Hello, World!";
String[] words = str.split(", ");

for (String word : words) {
    System.out.println(word);
}
```

Output:
```
Hello
World!
```

In the example above, the `split(", ")` method is used to split the string `str` at each occurrence of `", "`, resulting in an array of substrings stored in the `words` variable. Then, we iterate over the array and print each substring.

You can split a string using various delimiters, such as a single character, a sequence of characters, or a regular expression pattern. Here are a few more examples:

Splitting with a single character:
```java
String str = "one,two,three,four";
String[] parts = str.split(",");

for (String part : parts) {
    System.out.println(part);
}
```

Output:
```
one
two
three
```

By: Sanjay Madaan
https://in.linkedin.com/in/sanjay-madaan-9569452043

four
```

Splitting with a sequence of characters:
```java
String str = "apple,banana;cherry-orange";
String[] fruits = str.split(",|;|-");

for (String fruit : fruits) {
    System.out.println(fruit);
}
```

Output:
```
apple
banana
cherry
orange
```

Splitting with a regular expression pattern:
```java
String str = "Hello123World456";
String[] parts = str.split("\\d+");

for (String part : parts) {
    System.out.println(part);
}
```

Output:
```
Hello
World
```

In the last example, the regular expression `\\d+` is used as the delimiter to split the string at one or more consecutive digits.

By: Sanjay Madaan
https://in.linkedin.com/in/sanjay-madaan-9569452043
```

You can customize the delimiter according to your specific needs by providing the appropriate regular expression or character sequence to the `split()` method. Additionally, you can use the overloaded version of `split()` that takes a `limit` parameter to specify the maximum number of substrings to be returned.
Remember to handle any potential exceptions when working with the `split()` method, such as a pattern compilation error if an invalid regular expression is used.

Here are a few examples of how strings can be used in Java:

1. Creating a String:
```java
String greeting = "Hello, World!";
String name = new String("John Doe");
```

2. Concatenating Strings:
```java
String firstName = "John";
String lastName = "Doe";
String fullName = firstName + " " + lastName;
```

3. Getting the Length of a String:
```java
String message = "This is a string.";
int length = message.length();
```

4. Accessing Characters in a String:
```java
String word = "Hello";
char firstChar = word.charAt(0);
char lastChar = word.charAt(word.length() - 1);
```

5. Comparing Strings:
```java
String str1 = "Hello";
String str2 = "Hello";
boolean areEqual = str1.equals(str2); // true
```

```
String str3 = "World";
boolean areEqualIgnoreCase = str1.equalsIgnoreCase(str3); // false
```

6. Splitting a String:
```java
String sentence = "This is a sentence.";
String[] words = sentence.split(" ");
```

7. Extracting Substrings:
```java
String message = "Hello, World!";
String subMessage = message.substring(7); // "World!"
String subMessage2 = message.substring(7, 12); // "World"
```

8. Changing Case:
```java
String word = "Hello";
String upperCase = word.toUpperCase(); // "HELLO"
String lowerCase = word.toLowerCase(); // "hello"
```

By: Sanjay Madaan
https://in.linkedin.com/in/sanjay-madaan-9569452043

Here are a few more Java programs involving string manipulation:

1. Program to count the occurrences of a specific character in a string:
```java
String str = "Hello, World!";
char target = 'o';
int count = 0;

for (int i = 0; i < str.length(); i++) {
    if (str.charAt(i) == target) {
        count++;
    }
}

System.out.println("Occurrences of '" + target + "': " + count);
```

2. Program to reverse a string:
```java
String str = "Hello, World!";
String reversed = "";

for (int i = str.length() - 1; i >= 0; i--) {
    reversed += str.charAt(i);
}

System.out.println("Reversed string: " + reversed);
```

3. Program to check if a string is a palindrome:
```java
String str = "level";
boolean isPalindrome = true;

for (int i = 0; i < str.length() / 2; i++) {
    if (str.charAt(i) != str.charAt(str.length() - 1 - i)) {
        isPalindrome = false;
        break;
    }
}
```

By: Sanjay Madaan
https://in.linkedin.com/in/sanjay-madaan-9569452043

```
if (isPalindrome) {
    System.out.println("The string is a palindrome.");
} else {
    System.out.println("The string is not a palindrome.");
}
```

4. Program to remove all whitespace from a string:
```java
String str = "  Hello,   World!  ";
String trimmed = str.replaceAll("\\s", "");

System.out.println("Trimmed string: " + trimmed);
```

5. Program to check if two strings are anagrams of each other:
```java
String str1 = "listen";
String str2 = "silent";
boolean areAnagrams = true;

if (str1.length() != str2.length()) {
    areAnagrams = false;
} else {
    char[] charArray1 = str1.toCharArray();
    char[] charArray2 = str2.toCharArray();
    Arrays.sort(charArray1);
    Arrays.sort(charArray2);
    areAnagrams = Arrays.equals(charArray1, charArray2);
}

if (areAnagrams) {
    System.out.println("The strings are anagrams.");
} else {
    System.out.println("The strings are not anagrams.");
}
```
Certainly! Here are a few more Java programs involving string manipulation:

6. Program to find the index of a substring within a string:
```java
String str = "Hello, World!";
String substring = "World";
int index = str.indexOf(substring);

if (index != -1) {
    System.out.println("Substring found at index: " + index);
} else {
    System.out.println("Substring not found.");
}
```

7. Program to convert a string to an array of characters:
```java
String str = "Hello, World!";
char[] charArray = str.toCharArray();

System.out.println("Character Array:");
for (char ch : charArray) {
    System.out.print(ch + " ");
}
```

8. Program to check if a string starts with a specific prefix:
```java
String str = "Hello, World!";
String prefix = "Hello";
boolean startsWithPrefix = str.startsWith(prefix);

if (startsWithPrefix) {
    System.out.println("The string starts with the prefix.");
} else {
    System.out.println("The string does not start with the prefix.");
}
```

9. Program to replace a specific substring within a string:
```java
String str = "Hello, World!";
```

```
String oldSubstring = "World";
String newSubstring = "Universe";
String replacedString = str.replace(oldSubstring, newSubstring);

System.out.println("Replaced string: " + replacedString);
```

10. Program to convert a string to uppercase or lowercase using methods:
```java
String str = "Hello, World!";
String uppercase = str.toUpperCase();
String lowercase = str.toLowerCase();

System.out.println("Uppercase: " + uppercase);
System.out.println("Lowercase: " + lowercase);
```

By: Sanjay Madaan
https://in.linkedin.com/in/sanjay-madaan-9569452043

Some more Examples:

```
package com.java.basic;

public class StringDemo1 {
public static void main(String[] args) {

String str1 = new String("THis Is a JavA SESsIon");

System.out.println("Length = " + str1.length());
//index -> 0 to length-1
System.out.println("Charater at index 5 = "+ str1.charAt(5));
System.out.println("Index of s = " + str1.indexOf("s"));
System.out.println("Last Index of s = " + str1.lastIndexOf("s"));
System.out.println("Index of s = " + str1.indexOf("#"));

System.out.println("\nIndex of sIon = " + str1.indexOf("sIon"));
System.out.println("Last Index of s IS = " + str1.lastIndexOf("s Is"));
System.out.println("Index of sIon# = " + str1.indexOf("sIon#"));

System.out.println("\nStarts with (this)= " + str1.startsWith("this"));
System.out.println("Starts with (THis)= " + str1.startsWith("THis"));
System.out.println("Ends with (ion)= " + str1.endsWith("ion"));
System.out.println("Ends with (Ion)= " + str1.endsWith("Ion"));

System.out.println("\nLowercase = " + str1.toLowerCase());
System.out.println("Uppercase = " + str1.toUpperCase());

System.out.print("\nToggle::");
String togglestr1 = new String("");
for(int i=0; i<str1.length(); i++) {
        if(Character.isLowerCase(str1.charAt(i))) {
                togglestr1 = togglestr1 + Character.toUpperCase(str1.charAt(i));
        }else {
                togglestr1 = togglestr1 + Character.toLowerCase(str1.charAt(i));
        }
}
System.out.println(togglestr1);
System.out.println("Original str:: " + str1);

System.out.println("\nAfter replace S with Q : "
                                        + str1.replace('S', 'P'));
System.out.println("\nSubstring from index 6 = "
                                        + str1.substring(6));
System.out.println("Substring from index 6 to 13 = "
                + str1.substring(6,14));
```

By: Sanjay Madaan
https://in.linkedin.com/in/sanjay-madaan-9569452043

```
String strarr[] = str1.split(" ");
System.out.println("\nLength of strarr = " + strarr.length);
System.out.println("String after Split::");
for(String str : strarr) {
	System.out.println(str);
}

System.out.println("\n==========");
String str2 = new String("    String      ");
System.out.println("Length of str2 = " + str2.length());
str2 = str2.trim();
System.out.println("(after trim)Length of str2 = " + str2.length());


StringBuilder sb = new StringBuilder(str2);
sb.reverse();
String revstr2 = sb.toString();
System.out.println("\nOriginal str2   = " + str2);
System.out.println("Reverse of str2 = " + revstr2);

str2 = str2.concat(" Session");
System.out.println("\nUpdated str2   = " + str2);

str2 = "This is a ".concat(str2);
System.out.println("\nUpdated str2   = " + str2);

str2 = str2.replace("S", "");
System.out.println("\nString after replace = " + str2);



	}
}
```

==========

```
package com.java.basic;
//Duplicate character in a given string
//Count the number of duplicate character in a given string
public class StringDemo2 {
public static void main(String[] args) {
	String str1 = new String("This is a String Session");
	//a to z -> 26 characters
	int charcount[] = new int[26]; // 0 to 25  -> 97-97 to 122-97
	// a to z -> 97 to 122
```

By: Sanjay Madaan
https://in.linkedin.com/in/sanjay-madaan-9569452043

```
		// A to Z -> 65 to 90
		str1 = str1.toLowerCase();

		for(int i=0; i<str1.length(); i++) {
			if(Character.isAlphabetic(str1.charAt(i))) {
				int index = str1.charAt(i) - 97 ;
				charcount[index]++;
			}else {
				System.out.println(str1.charAt(i) + " is not alphabet");
			}
		}

		int count = 0;
		for(int i=0; i<charcount.length; i++) {
			if(charcount[i]>1) {
				System.out.println("duplicate character in string = "
								+ (char) (i+97));
				count++;
			}
		}
		System.out.println("duplicate character count:" + count);
		}
}

====================================

package com.java.basic;

public class StringDemo4 {
public static void main(String[] args) {
		String str= new String("Hii! How are you my friend, where do you live?");
		String strarr[] = str.split(" ");
		int countwords = strarr.length;
		  System.out.println("Words in given string:"+ countwords);

		  System.out.println("==================");
		  String str1 = new String("This finds the substring from the string");
		  String substr = "the";
		  System.out.println("Str1 contains substr ? = "
							+ str1.contains(substr));

		  System.out.println("==================");
		  int countsubstr = 0;
		String strarr1[] =  str1.split(" ");
		for(int i=0; i<strarr1.length; i++) {
			if(strarr1[i].equals(substr)) {
```

```
                        countsubstr++;
                    }
            }
            System.out.println("Occurance of substr in str1 = " + countsubstr);
}
}
```

==========

```
package com.java.basic;

import java.util.Arrays;

public class StringDemo5 {
public static void main(String[] args) {
            String[]arr1 = {"Hello","World"};
            String[]arr2 = {"Welcome", "To!"};

            String[] mergedArray = mergeArrays(arr1,arr2);

System.out.println("Merged Array :" + Arrays.toString(mergedArray));
}
public static String[] mergeArrays(String[] arr1,String[] arr2)
{
int length1 = arr1.length;
int length2 = arr2.length;
String[] mergedArray = new String[length1+ length2];

System.arraycopy(arr1, 0, mergedArray, 0, length1);
System.arraycopy(arr2, 0, mergedArray, length1, length2);
return mergedArray;

}
}
```

==========

```
package com.java.basic;

public class StringDemo6 {
public static void main(String[] args) {
            String inputString = "Hello world,Hello world,Hello world,"
                                                    + "Hello world,Hello world";
    String substring = "world";
    int occurrences = countSubstringOccurrences(inputString, substring);
    System.out.println("Number of occurrences: " + occurrences);
```

By: Sanjay Madaan
https://in.linkedin.com/in/sanjay-madaan-9569452043

```
    String str11= new String("J a  va    Str   ing");
                System.out.println("Before removing spaces = " + str11 );
                System.out.println("Length before = " +str11.length());
                System.out.println();
        String str= str11.replaceAll("\\s", "");
        System.out.println("After removing spaces = " +str);
        System.out.println("Length after = " +str.length());
  }

  public static int countSubstringOccurrences(String inputString, String substring) {
    int count = 0;
    String strarr[] = inputString.split("[ ,]");
    for(String str : strarr) {
          if(str.equals(substring)) {
                count++;
          }
    }

    return count;
  }
}
```

By: Sanjay Madaan
https://in.linkedin.com/in/sanjay-madaan-9569452043