Billing info update failed.

PROGRAMMING LANGUAGES

Top 10 String Methods In Java

By **Samyak Jain** - July 06, 2021 - 14 min read

Computers can only comprehend binary numbers i.e. 0 and 1. Isn't that correct?

Humans, on the other hand, are mainly concerned with Strings.

Strings are widely used in almost all application codes. As a result, robust support for string composition and manipulation is necessary for a programming language to be effective both in terms of storage and time. And as the industry's favorite, Java does this very well.

Before we find out how let's discuss -- ***What exactly is a String?***
A string can be thought of as a stream of characters that are supported by the programming language you are using and represents textual data.

Exploring String Class in Java

In Java, Strings are an instance of the ***String class*** and Double quotes **“ ”** are used to represent Strings in Java.

Examples of Strings in Java

1. **“Project-based Learning”**
2. **“Java NOob 2 Expert!!”**
3. **“abc”**

How do we create strings in Java?

A String object can be created in one of two ways:

1. String Literal - Double quotes are used to construct string literals.
2. Using new: Using the keyword "new," a Java String is generated.

Let's look at each.

String Literal - Double quotes are used to construct string literals.

Example: ***String greetString = “Welcome”;***

This creates a String object in Java String Pool also referred to as String Constant Pool (SCP) which is a special place for all strings in Heap Memory.

Why is this special area in heap(SCP) required? Is it only for Strings?

Yes it is only for strings mainly for 2 reasons

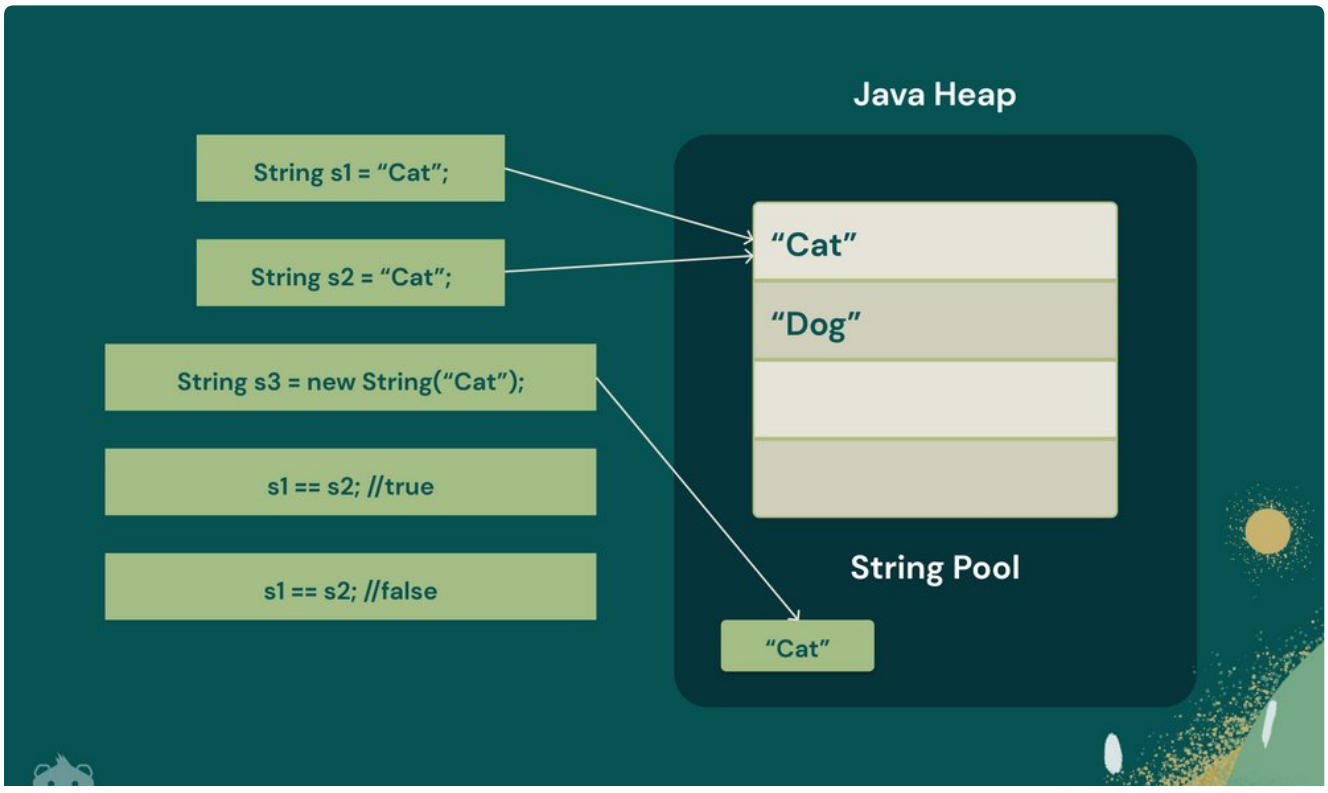
1. Most Used Objects
2. Immutability of Strings

By storing only one copy of each literal String in the pool, JVM will optimize the amount of memory allocated for strings. When we declare a String variable and assign it a value, the JVM scans the pool for a String of the same value. If it is located, the compiler would automatically return a reference to its memory address rather than allocating additional memory.

```
public class SCP {

    public static void main(String[] args) {
        String s1 = "Crio";
        String s2 = "Crio";
        String s3 = new String("Crio");

        System.out.println(s1==s2); // true
        System.out.println(s1==s3); // false
        System.out.println(s2==s3); // false
        System.out.println(s1.equals(s3)); // true
        System.out.println(s1.equals(s2)); // true
        System.out.println(s2.equals(s3)); // true
    }
}
```



2. Using new: Using the keyword "new," a Java String is generated.

Example: ***String greetString = new String("Welcome")***

This declaration creates a string object in heap memory and checks to see if it is in the string pool. If the string "Welcome" is not in the string pool, it will add it; otherwise, it will skip it and return a reference variable, which refers to the newly created heap object.

To read more about SCP glance through - ***https://www.baeldung.com/java-string-constant-pool-heap-stack***

How and why is it useful to use String functions in Java programming?

The String class contains over 60 methods that serve as basic utility methods to make the life of a developer dealing with strings smoother.

Note about Immutability Of Strings

Since the String class is immutable, all string objects generated with literals or new operators cannot be altered or updated. At the time of creation, the value of the immutable class object is set. Immutability of strings has numerous advantages, including improved performance due to the String Pool model, security, and thread safety.

Quick Exercise to demonstrate this concept is given below

```
class ImmutableString{
 public static void main(String args[]){
   String str = "Learning String";
   manipulateString(str);
   System.out.println(str);//will print Learning String because
strings are immutable objects
```

```
 }

 private static void manipulateString(String str) {
   str.concat(" By Doing");//concat() method appends the string at
the end
 }
}
```

You must use different classes, such as StringBuilder that are mutable counterparts of String if you wish to move the String to different methods for modification.

```
class ImmutableString{
 public static void main(String args[]){
   StringBuilder str = "Learning String";
   manipulateString(str);
   System.out.println(str);//will print Learning String By Doing
because stringbuilder object is mutable
 }

 private static void manipulateString(StringBuilder str) {
   str.concat(" By Doing");//concat() method appends the string at
the end
 }
}
```

What if you plan to write your own implementation of existing methods?

That's a perfect way to practice and improve your Java skills, but it's definitely not a good idea to use it in production.

The primary reason for this is that you might easily introduce bugs into your code. The Java Community is vast, and these implementations have been thoroughly reviewed and tested in stringent environments and are well known. Instead of reinventing the wheel, you should concentrate on your business logic. Another reason includes inbuilt documentation support, enhances the readability of code, and continued support from the community.

For example, to find out the number of characters contained in the string object, length() method can be used instead of iterating over all characters and increasing the count one at a time.

```
String creditCardNumber = “1234-5678-2345”
boolean isValidCreditCard = creditCardNumber.length() == 12 ? true : false;
```

Let’s dive deep into the most commonly used Java string methods and understand their working.

1. indexOf()

Used to find characters and substrings in a string. It returns the index of the first occurrence from the left of the passed string within this string; moreover, if we provide the fromIndex, it starts searching from that index.

Syntax of indexOf()

```
public int indexOf(int str, int fromIndex)
```

Parameter(s)

- ***str:*** Substring to be searched for
- ***fromIndex:*** Search begins from this index

Return Value

Index of the first occurrence of the specified substring, 0 if an empty string is passed, or -1 if there is no such occurrence.

Overloaded Variants

```
1. public int indexOf(int ch)
```

```
2. public int indexOf(int ch, int fromIndex)
```

```
3. public int indexOf(String str)
```

```
4. public int indexOf(String str,int fromIndex)
```

Real world application of indexOf()

Ever wondered how your favorite search feature in Code Editors works?

indexOf() can be used to implement this feature. How?

The idea is to iterate over all of the lines and invoke indexOf() with the target string, and if there are multiple occurrences of the target string in the same line, start from where the last found substring ends.

```
public static int findOccurances(String[] paragraph, String target) {

     int index = 0, totalOccurances = 0;

    for    (int    lineNumber    =    0;lineNumber    <    paragraph.length;
++lineNumber)
 {
   String line = paragraph[lineNumber];
   while (true) {
    index = line.indexOf(target, index);
    if (index != -1) {
    System.out.println("Found  at  line  number  "  +  lineNumber  +  "  at
position " + index);
    totalOccurances++;
    index += target.length();
  } else {
    break;
   }
  }
 }
 return totalOccurances;
}
```

Try it yourself

Write a function that uses ***indexOf()*** to find if a given character is a vowel or not.

Input - any character; Output - true if a given character is vowel else false

Don't forget to share your unique approach in the comments below.

---

2. toCharArray()

Used to form a new character array from this string. The newly allocated array's contents are initialized to contain the characters represented by this string, and its length is the length of this string.

Syntax of toCharArray()

```
public char[] toCharArray()
```

Parameter(s)

None

Return Value

char[] array is created in memory with content as characters of the string and its reference is returned

Real world application of toCharArray()

As we have discussed above, string objects are immutable.

During heavy string manipulation operations, many new string objects are made and this can be memory inefficient. A good idea is to convert it to char Array first, then perform manipulation operations and then convert it back to a string object.

Let’s take an example to see if a given string is palindrome or not.

```
String palindrome = "Dot saw I was Tod";
int len = palindrome.length();
char[] charArray = palindrome.toCharArray();

for (int j = 0; j < len; j++) {
    if (charArray[j] != charArray[len - 1 - j]) {
       System.out.println(“Not a palindrome”);
    }
}
System.out.println(“Valid palindrome”);
```

Try it yourself

Find all the indices of uppercase letters present in the string and convert them to their ASCII value.

Example - Input - asBcDeE Output - [2,4,6] modified string - as66c68e69

Let us know your most optimal approach in the comments below

---

3. charAt()

The character at the designated index (follows 0 based indexing) is extracted from the string. It's useful for checking a certain index without looping over the whole string.

Syntax of charAt()

```
public char charAt(int index)
```

Parameter(s)

- ***Index:*** index of character in string to be retrieved

Return Value

char value from String at index passed as argument

- Throws

IndexOutOfBoundsException - if the argument provided is negative or greater than the length of String.

Real world application of charAt()

Counting the frequency of a particular character - ***charAt()*** can be used to iterate the string and count frequency of a character.

```
String str = "I am upskilling by learning by doing";

int spaceCount = 0;

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

System.out.println(spaceCount); // 6
```

Try it yourself

Print the characters along with their frequency in the order of their occurrence.

Example - Input - “learnByDoing”

Output - [[l, 1], [e,1], [a,1], [r, 1], [n, 2], [B, 1], [y,1], [d,1], [o, 1], [i,1], [g,1]]

Let us know your most optimal approach in the comments below.

---

4. concat()

The passed string is appended to the end of the specified string. It's a great option for combining/merging strings.

Syntax of concat()

```
public String concat(String str)
```

Parameter(s)

- ***str***: string to be appended at end of a given string

Return Value

A new String object with a passed string appended at the end of the given string.

Real world application of concat()

You must have filled many forms online and encountered fields like Address Line 1, Address Line 2.

Do you think these are stored as it is in the database?

You might be wrong if you think so. When form data is processed it is combined to form a single field like a full address.

```
String addressLine1 = "4424  Black Oak Hollow Road ";
String addressLine2 = "San Francisco ";
String addressLine3 = "California 94108";


String fullAddress =
addressLine1.concat(addressLine2).concat(addressLine3);


System.out.println(fullAddress);
// 4424  Black Oak Hollow Road San Francisco California 94108
```

Try it yourself

While entering credit/debit card details you must have encountered multiple fields to fill in entering the card number. How will you combine them and verify with the server that your card number is valid?

Assume that server stores number in filed credit card number in the format 1234-567-1239

Let us know your most optimal approach in the comments below.

---

5. replace()

Used for replacing characters and substrings in a string.

Syntax of replace()

```
public String replace(char oldChar, char newChar)
```

Parameter(s)

- ***oldChar:*** Character that needs to be replaced
- ***newChar:*** Character that will replace the oldChar

Return Value

A new String object containing a string created by replacing all instances of oldChar with newChar.

Real world application of replace()

If you are a programmer, you might have encountered a scenario in which you needed to modify the name of all instances of your variable. How would you put this feature into action?

```
public static void replace(String[] paragraph, String oldChar,
String newChar ) {

     for     (int     lineNumber=0;lineNumber<     paragraph.length;
++lineNumber)
 {
   String line = paragraph[lineNumber];
   line = line.replace(oldChar, newChar);
 }
}
```

Try it yourself

Given an array of Strings. Remove all the whitespaces present in all the strings that are part of this array.

Example - Input - [[‘hey this is ’], [‘a example of’], [‘replace method ’]];

Output - ‘heythisisaexampleofreplacemethod’

Let us know your most optimal approach in the comments below.

---

6. substring()

Used to extract a portion of a string from a given string. It creates a new string object without altering the original string.

Syntax of substring()

```
public String substring(int beginIndex, int endIndex)
```

Parameter(s)

- **beginIndex:** Search begins from this index

- **endIndex:** Search end at endindex-1

Return Value

A new String object resulting from the part of the original string sliced out on the basis of begin and endIndex.

- Throws

IndexOutOfBoundsException - if beginIndex is negative or larger than the length of this String object.

Overloaded Variant

```
1. public String substring(int beginIndex)
```

Real world application of substring()

We can find out all the substrings of a given string using ***substring().*** Though this is not an optimal approach, it will help you through the learning of the substring method's working.

```
public void SubString(String str, int n)
{
  for (int i = 0; i < n; i++)
    for (int j = i+1; j <= n; j++)
      System.out.println(str.substring(i, j));
}
```

Try it yourself

Find out if the given string is palindrome or not by using ***substring().***

Share your solution with fellow readers in the comments section below.

7. split()

Used to separate the specified string depending on the regular expression. The limit statement is used to limit the number of strings returned after separating.

Syntax of split()

```
public String[] split(String regex, int limit)
```

Parameter(s)

- ***regex:*** Substring to be searched for
- ***limit:*** Search begins from this index

Return Value

An array containing each substring of this string is ended by a substring that matches the given expression or by the end of the string. The array's substrings are listed in the order in which they appear in this string.

- Throws

PatternSyntaxException - if the regular expression's syntax is invalid

Overloaded Variants

```
1. public String[] split(String regex)
```

Real world application of split()

We can count the number of words present in a paragraph by splitting the string by passing space as an argument.

```
String str = "This string contains five words";
String[] arrayOfWords = str.split(" ");
System.out.println(arrayOfWords.length);
```

Try it yourself

Given a date string in DD/MM/YYYY format, extract and print the year given in the string.

Example - Input - 26/09/2000 Output - 2000

---

8. compareTo()

As Java developers, we often need to sort items that are grouped together in a list or in a class, and compareTo() is used to compare two Strings alphabetically.

Syntax of compareTo()

```
public int compareTo(String anotherString)
```

Parameter(s)

- ***anotherString:*** String to be compared

Return Value

0 if the argument string is equal to given string;

Value < 0 if the given string is lexicographically less than the string argument;

Value > 0 if the given string is lexicographically greater than the string argument.

Real world application of compareTo()

Comparison-based sorting algorithms are the most used sorting algorithms.

In educational institutes mostly students are arranged based on increasing the

order of their firstName. If the firstName of 2 students is the same then decide on the basis of increasing order of their lastName.

```
class Student implements Comparable<Student> {
  public String firstName;
  public String lastName;
  public int age;

  public Student(String firstName) {
    this.firstName = firstName;
    this.age = age;
  }
  public int compareTo(Student student) {
    int comparison = firstName.compareTo(student.firstName);
    return comparison == 0
           ? lastName.compareTo(student.lastName)
           : comparison;
  }
}
public class SortStudents {
  public static void main(String[] args) {
      List<Student> students=new ArrayList<Student>();
      students.add(new Student("Jame", 19));
      students.add(new Student("Saurav", 20));
      students.add(new Student("Jacky", 20));

    Collections.sort(students);
  }
}
```

Try it yourself

Use ***compareTo()*** to find the length of a string.

Hint: Closely examine what compareTo() returns and how this can help us to find the length of the string.

Mention your approach in the comments section.

9. strip()

To eliminate all trailing and leading whitespaces from the specified string.

Syntax of strip()

```
public String strip()
```

Parameter(s) : None

Return Value

Returns a string whose value is this string, with all leading and trailing white space removed.

Real world application od strip()

When taking input from your presentation layer, you must ensure that trailing and leading whitespaces do not contaminate the actual string. This ensures the data is consistently stored in the database and processed in the backend.

split() can be used to ensure that our input is not polluted.

```
public static void login() {
      // Get input from frontend
      String username = getUserNameFromUI(); // "  crio.do "
      String password = getPasswordFromUI(); // " abc"

      String cleanUsername = username.strip(); // "crio.do"
      String cleanPassword = password.strip(); // "abc"

      Account account = new Account(cleanUsername, cleanPassword);
      // save credentials to database
      db.save(account);
}
```

Try it yourself

What happens if we don't delete the whitespaces? Does it have an effect on

our business operations, or is it just a way to keep things clean? Find out by taking any leading spaces in a string and comparing it to the original string with no leading spaces.

Mention your observations in the comment section below.

---

10. valueOf()

Used to return string representation of the passed argument. ***valueOf()*** has a plethora of overloaded variants that aid in the conversion of almost any primitive form to string.

Syntax of calueOf()

```
public static String valueOf(char[] data)
```

Parameter(s)

- ***data :*** Character Array to be converted to string

Return Value

Returns a new String object that contains data as the string representation of the passed argument

Overloaded Variants

`public static String valueOf(boolean b)`
`public static String valueOf(char c)`
`public static String valueOf(int i)`
`public static String valueOf(long l)`
`public static String valueOf(float f)`

`public static String valueOf(double d)`
`public static String valueOf(char[] data)`
`public static String valueOf(char[] data, int offset, int count)`
`public static String valueOf(Object obj)`

Real world application of valueOf()

We can form a single String combining all the elements of char Array using ***valueOf()***

```
char array[] = {'a', 'b', 'c', 'd', 'e', 'f', 'g'};
System.out.println(String.valueOf(array)) ;// abcdefg
```

Try it yourself

Use ***valueOf()*** to convert an object to its String representation. What did you observe? Find out the reason for this weird output and let us know in the comments below.

Recap

Here's a quick summary of the Java string methods we conquered in the article.

Java Method	Syntax	Returns	Usage
indexOf()	`indexOf(String str, int fromIndex)`	`int`	For finding the index of the first occurrence of a character or a string in the given string.
toCharArray()	`toCharArray()`	`char[]`	To form a new character array from this string
charAt()	`charAt(int index)`	`char`	To get character at the specified index

concat()	concat(String str)	String	To append the passed string to the end of the given string.
replace()	replace(char oldChar, char newChar)	String	To replace all the occurrences of the given character/String from given String
substring()	substring(int beginIndex, int endIndex)	String	To get a part of a string from the given string.
split()	split(String regex,int limit)	String[]	For splitting the given string based on the given regular Expression.
compareTo()	compareTo(String anotherString)	int	To compare two Strings Lexicographically.
strip()	strip()	String	To remove all trailing and leading whitespaces from the given string.
valueOf()	valueOf(char[] data)	String	To return String representation of the passed argument.

List of other useful string methods for further learning

- contains(CharSequence s)
- isEmpty()
- join()
- repeat()
- startsWith() / endsWith()
- toLowerCase() / toUpperCase()
- indent()

Final thoughts

Congratulations!! on getting your Java strings game stronger. When dealing with strings in Java, you will undoubtedly feel more secure and at ease. But don't stop there; look at other methods and see if you can understand and

implement them using the ***Learn By Doing*** process.

Now is the time to spread the word and allow your mates to catch up to you.

Share this article:

Written by Samyak Jain

MORE POST BY SAMYAK JAIN →

You might also like

PROGRAMMING LANGUAGES

Inheritance in C++ Simplified for Programmers

October 29, 2021 - 7 min read

PROGRAMMING LANGUAGES

Deep Dive into Abstraction in C++ with Easy Examples

October 14, 2021 - 9 min read

PROGRAMMING LANGUAGES

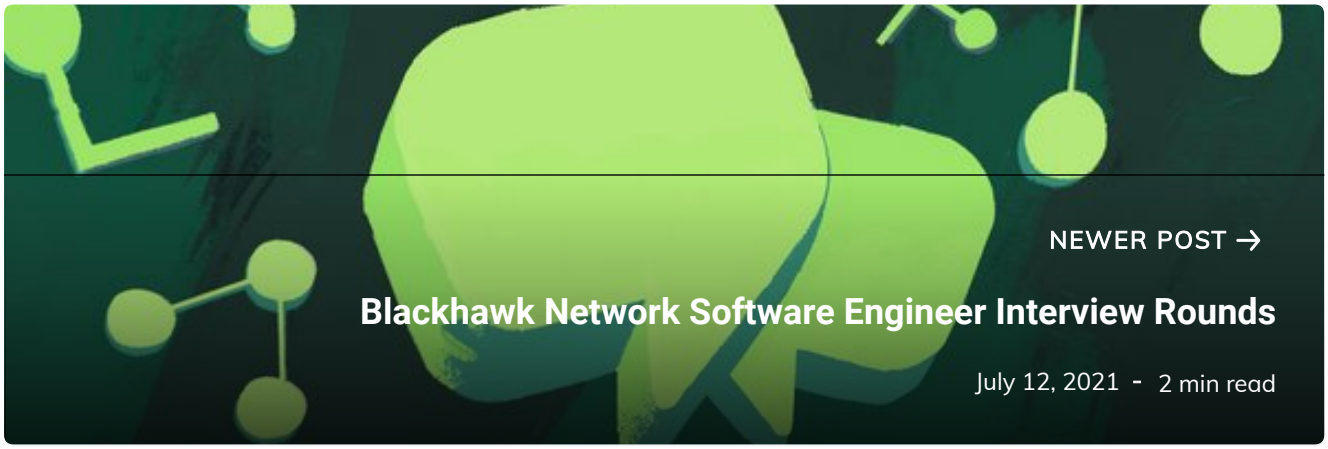
Learn Encapsulation in C++ with simple examples

September 29, 2021 - 7 min read

PROGRAMMING LANGUAGES

15 Insanely Useful String Functions in Python

July 02, 2021 - 17 min read



LATEST POSTS

PROGRAMMING LANGUAGES

Inheritance in C++ Simplified for Programmers

October 29, 2021 - 7 min read

MINI PROJECTS

A Comprehensive List Of Web Development Projects

October 22, 2021 - 15 min read

PROGRAMMING LANGUAGES

Deep Dive into Abstraction in C++ with Easy Examples

October 14, 2021 - 9 min read

DATABASES

Virtual Memory (Memory Leaks)

October 08, 2021 - 10 min read

PROGRAMMING LANGUAGES

Learn Encapsulation in C++ with simple examples

September 29, 2021 - 7 min read

CATEGORIES

CAREER GUIDANCE CRIO COMMUNITY DATA STRUCTURES AND ALGORITHMS
DATABASES DEVELOPER ESSENTIALS IN THE NEWS MINI PROJECTS
PROGRAMMING LANGUAGES WEB DEVELOPMENT WHY CRIO

STAY CONNECTED

NAVIGATION

Mini Projects

Data Structures and Algorithms

Career Guidance

Developer Essentials

Subscribe to newsletter

Stay up to date! Get all the latest posts delivered straight to your inbox.

Your email address

Subscribe

© 2021 Crio Blog - All rights reserved.