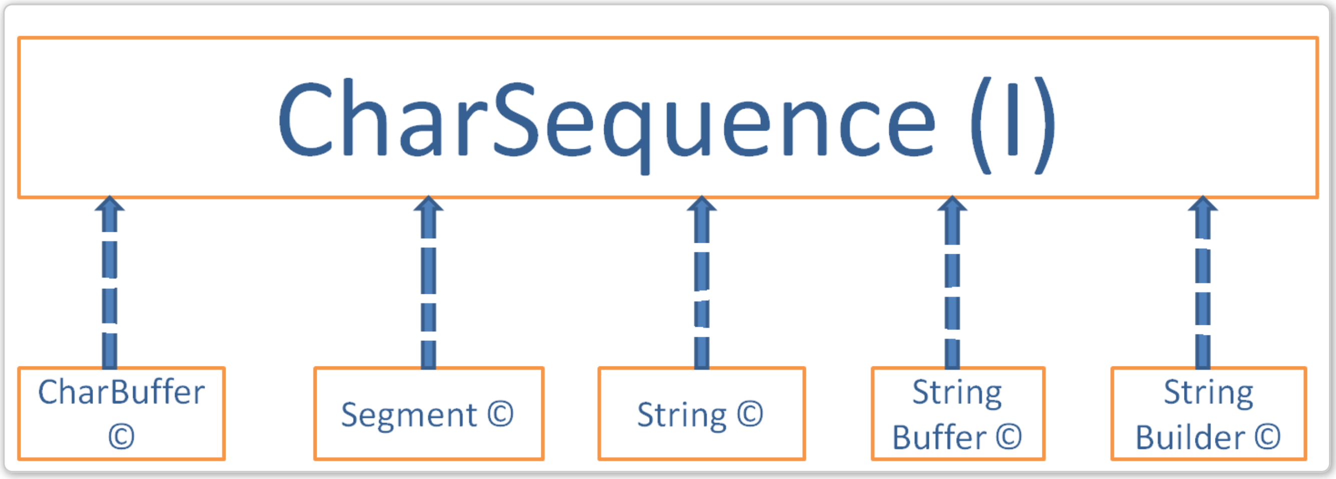
Strings in Java & Selenium

Table of content

String in Java/selenium

Strings are a sequence of characters enclosed within the double quotes which include alphabets, numbers, special characters and so on. Strings are objects in Java, which means strings are not a primitive data type.

The Architecture of Strings :



We can form strings using below classes; all three classes implement CharSequence interface and extends Object Class :

String

StringBuffer

StringBuilder

String Class in Java

String class implements the CharSequence interface and extends Object class, and object class is a topmost class in java.

A string is represented with "" (double quotes). The string is immutable, which means once we create a string object, we cannot modify the String object.

The class String includes methods for examining individual characters of the String, for comparing strings, for searching strings, for extracting substrings, and for creating a copy of a string with all characters translated to uppercase or lowercase.

We can create strings in two methods :

Double quotes method (most preferred method)

String object method

Customer Locators in Selenium

Double quotes method :

The string is created by enclosing the string inside double-quotes. Below are a few processes that happen when we use double quotes to create a string.

> JVM checks String Constant pool for any value which same as what we are trying to create.

> If yes, point that string object reference to the variable that we are trying to create.

> If no, JVM creates new string object in String constant pool

```
public static void main(String[] args) {
   // create chercher tech string
   String firstString = "CherCher Tech";
   System.out.println("String we created : "+firstString);
}
```

2. String object method :

We can also create a string by creating an object of String class. We have to call the String class constructor to create the string, and that creates the string in the non-constant pool in the String pool.

JVM will not check for the duplicate strings if we use String Constructor to create a string. So there can be duplicate strings in the non-constant pool.

```
public class StringExample2 {
   public static void main(String[] args) {
      // create the string

      String secondString = new String("Selenium-webdriver.com");
      // print the string
```

```
        System.out.println("String created is : "+secondString);
    }
}
```

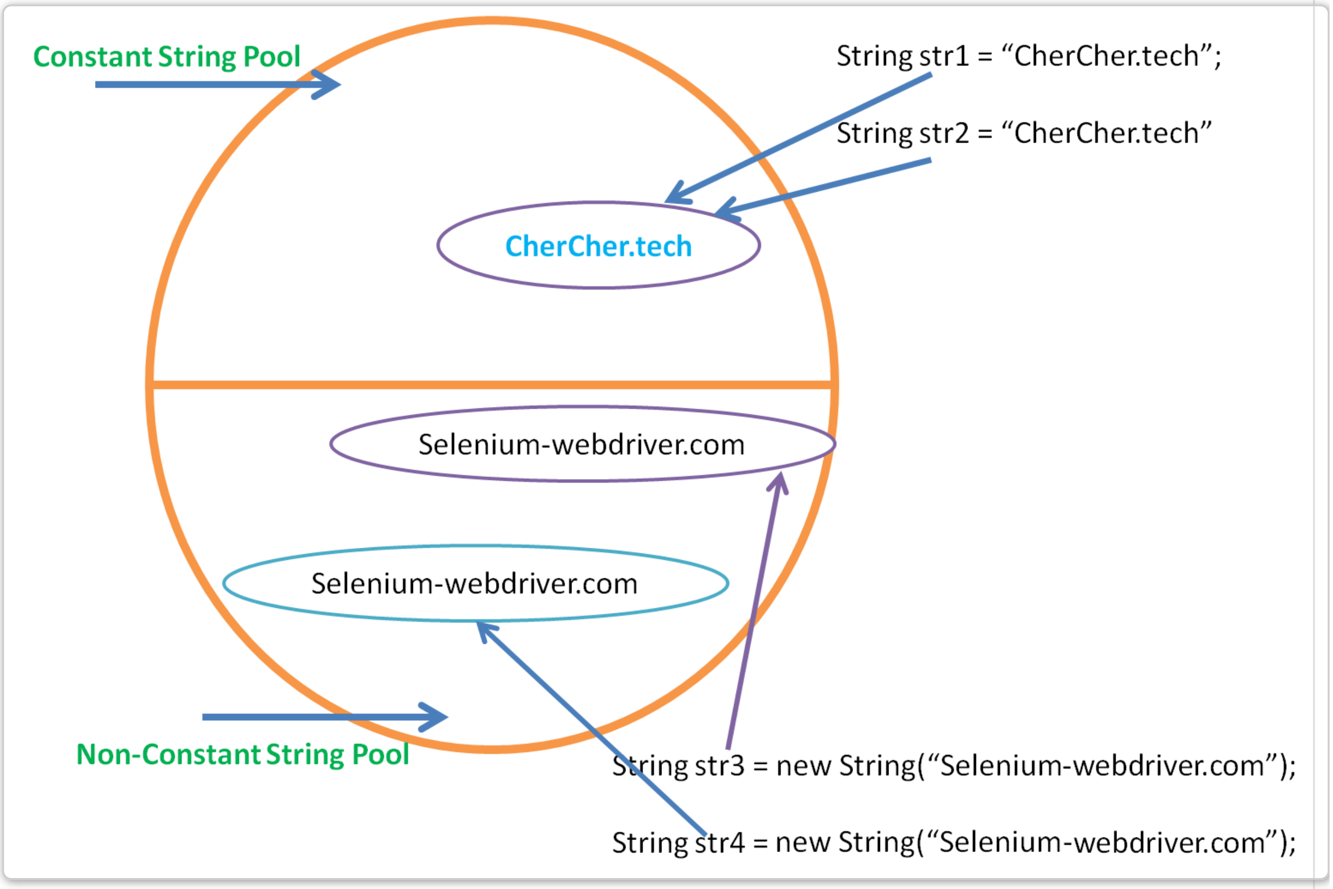
Parameterize cucumber BDD

What is String pool :

String Pool is a pool of string values that are available to other parts of the program. String pool can be divided into two parts

Constant String pool

Non-Constant String pool


Constant String Pool
String str1 = "CherCher.tech";
String str2 = "CherCher.tech"
CherCher.tech
Selenium-webdriver.com
Selenium-webdriver.com
Non-Constant String Pool
String str3 = new String("Selenium-webdriver.com");
String str4 = new String("Selenium-webdriver.com");

1. String Constant Pool :

String constant pool located inside a section of memory is called the heap. This is a part of memory that is used for run-time operations, working with classes and objects.

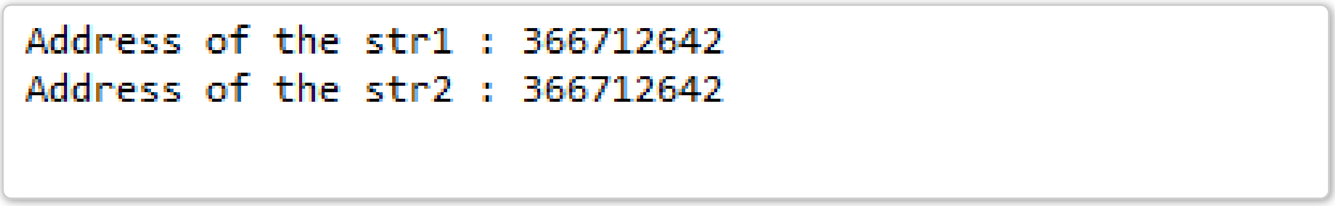
String constant pool does not allow users to create duplicate strings. If the user tries to create a duplicate string using " (double quotes), then the string pool returns the address of the existing string.

So at any given time, the Constant String pool will contain only unique values, and also users may not feel any difference with operations.

In the below example, we will create two strings, and we will verify the address of those strings.

```
public class StringExample3 {
   public static void main(String[] args) {
      String str1 = "checrcher tech";
      String str2 = "checrcher tech";
      // get the address of the strings
      System.out.println("Address of the str1 : "+System.identityHashCode(str1));
      System.out.println("Address of the str2 : "+System.identityHashCode(str2));
   }
}
```

Output :



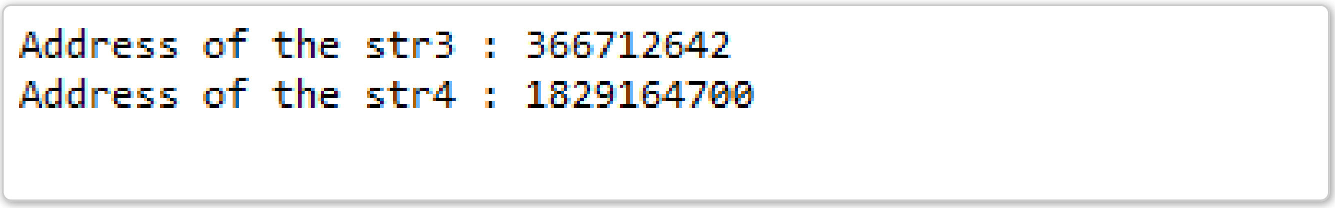
```
Address of the str1 : 366712642
Address of the str2 : 366712642
```

Non-Constant String Pool :

Non-constant string pool will allow the user to create duplicates. Below example print the address of the string created.

```
public class StringExample4 {
   public static void main(String[] args) {
      String str3 = new String("Selenium-webdriver.com");
      String str4 = new String("Selenium-webdriver.com");
      // get the address of the strings
      System.out.println("Address of the str3 : "+System.identityHashCode(str3));
      System.out.println("Address of the str4 : "+System.identityHashCode(str4));
   }
}
```

Output :

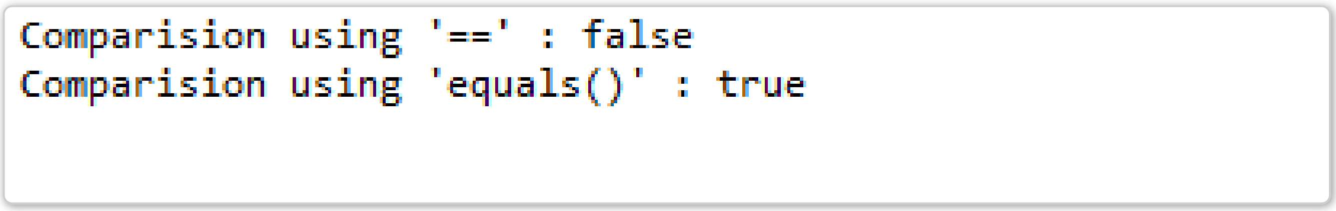


```
Address of the str3 : 366712642
Address of the str4 : 1829164700
```

Compare Strings : We cannot compare strings using '==' relational operator, as '==' operator tries to compare the address of the object. But most of the time, strings may not be stored in the same object so the address will be different for each string. We should use equals() present in the String object to compare two strings.

```
public static void main(String[] args) {
   String str1 = new String("CherCher tech");
   String str2 = "CherCher tech";
   System.out.println("Comparision using '==' : " + (str1 == str2));
   System.out.println("Comparision using 'equals()' : " +(str1.equals(str2)));
}
```

Output :



```
Comparision using '==' : false
Comparision using 'equals()' : true
```

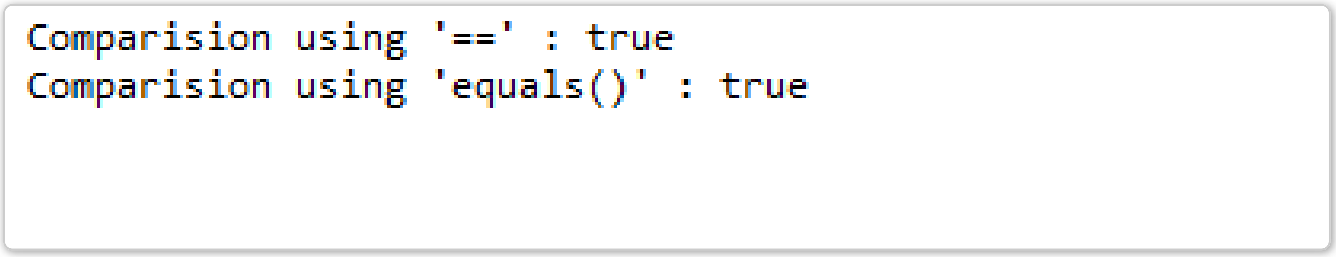
Really, Cannot we compare Strings using '==' : We can compare the strings using '==' sign as well, but with one condition, Strings must have created using "".

In this case also '==' tries to compare the address but as the constant pool will let only unique values to present, so the address will be the same when the content of the strings is the same.

```
public static void main(String[] args) {
   String s3 = "CherCher tech";
   String s4 = "CherCher tech";

   System.out.println("Comparision using '==' : " + (s3 == s4));
   System.out.println("Comparision using 'equals()' : " +(s3.equals(s4)));
}
```

Output :



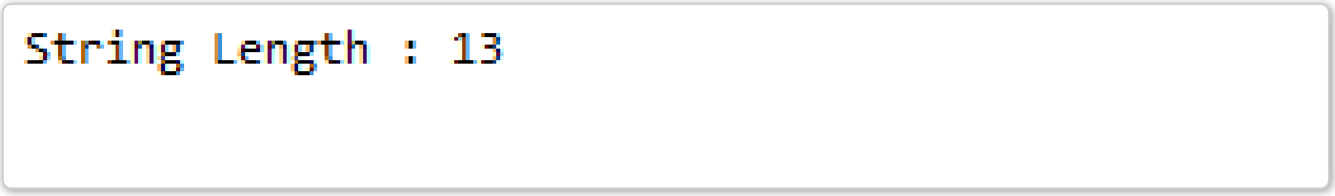
```
Comparision using '==' : true
Comparision using 'equals()' : true
```

Useful methods in String : There are so many methods present in String, but we will learn about the most used methods of String.

1. Length : length method return number of characters present in the string, it returns an integer value.

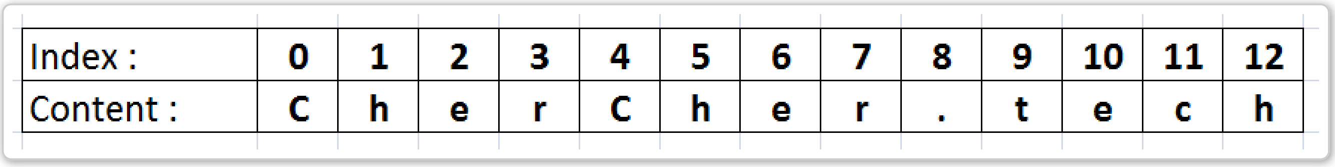
```
public static void main(String[] args) {
    String str = "CherCher.tech";
    System.out.println("String Length : "+str.length());
}
```

Output :



```
String Length : 13
```

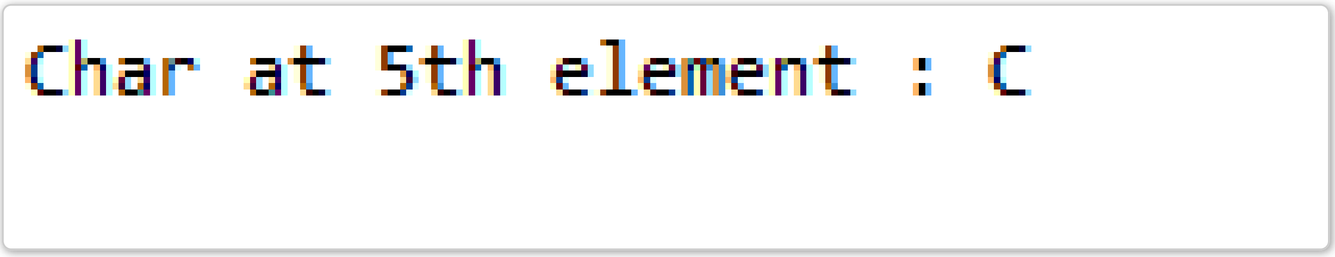
charAt : charAt method returns a character at a particular position; in java, the index starts from 0. If we want to get 5th element, then we have to query for 4th index



Index :	0	1	2	3	4	5	6	7	8	9	10	11	12
Content :	C	h	e	r	C	h	e	r	.	t	e	c	h

```
public static void main(String[] args) {
    String str = "CherCher.tech";
    // index starts from zero
    System.out.println("Char at 5th element : "+str.charAt(4));
}
```

Output :



```
Char at 5th element : C
```

3. **Concat :** concat is nothing but adding two or more strings, concat() method helps the user to add two strings into one string. This method will not disturb the existing strings, and it will create a new string with concatenation and returns it.

```
public static void main(String[] args) {
    String str = "CherCher";
    String str2 = "Tech";
    String abc = str.concat(str2);

    System.out.println("String  : "+str);
    System.out.println("String 2  : "+str2);
    System.out.println("String concat : "+abc);
}
```

Output :

```
String   : CherCher
String 2   : Tech
String concat : CherCherTech
```

4. **startsWith :** startsWith method verifies whether the String starts with given substring or not. This returns a boolean value, true if the String starts with given substring else false.

```
public static void main(String[] args) {
   String str = "CherCher.tech";
   System.out.println("Starts with Cher : " +str.startsWith("Cher"));
   System.out.println("Starts with ABC : " +str.startsWith("ABC"));
}
```

Output :

```
Starts with Cher : true
Starts with ABC : false
```

5. **endsWith :** endsWith method verifies whether the String ends with given substring or not, endsWith method also returns a boolean value.

6. **contains :** contains method verifies whether the String contains substring or not, the substring could be anywhere in the main string.

```
public static void main(String[] args) {
   String str = "CherCher.tech";
   System.out.println("contains te : " +str.contains("te"));
   System.out.println("contains xyz : " +str.contains("xyz"));
}
```

Output :

```
contains te : true
contains xyz : false
```

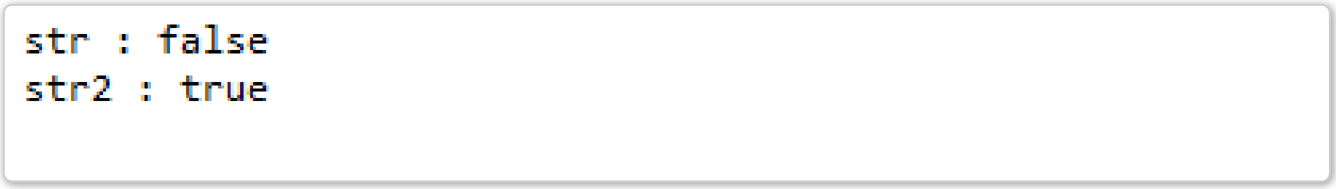
7. **equals :** equals() method compares the two string based on the content present in the string if the content matches the equals() method returns true else false.

```
public static void main(String[] args) {
   String str = "CherCher.tech";
   System.out.println("equals : " +str.equals("CherCher.tech"));
}
```

8. **isEmpty :** isEmpty method verifies whether given string is empty("") or not, empty means string with no value ("").

```
public static void main(String[] args) {
   String str = "CherCher.tech";
   String str2 = "";
   System.out.println("str : " +str.isEmpty());
   System.out.println("str2 : " +str2.isEmpty());
}
```

Output :



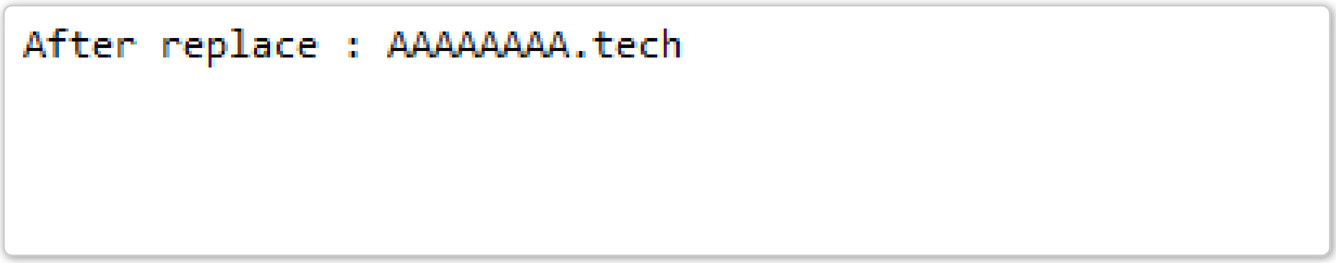
```
str : false
str2 : true
```

9. **replace :** replace method replaces the specified value with the given new value in all occurrences

```
public static void main(String[] args) {
   String str = "CherCher.tech";

   System.out.println("After replace : " +str.replace("Cher", "AAAA"));
}
```

Output :

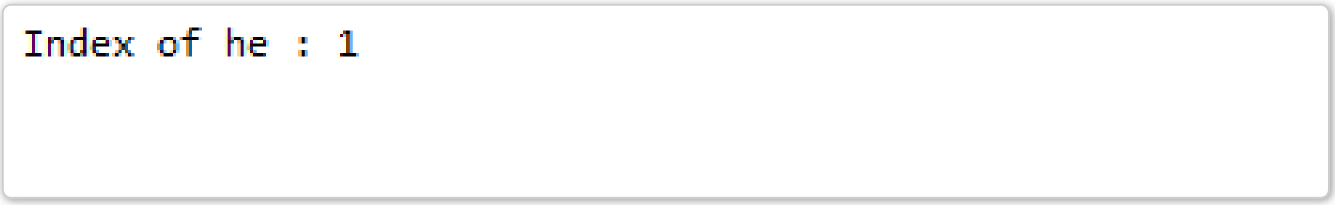


```
After replace : AAAAAAAA.tech
```

10. **indexOf :** this method returns the first index of the given substring or character.

```
public static void main(String[] args) {
   String str = "CherCher.tech";
   System.out.println("Index of he : " +str.indexOf("he"));
}
```

Output :

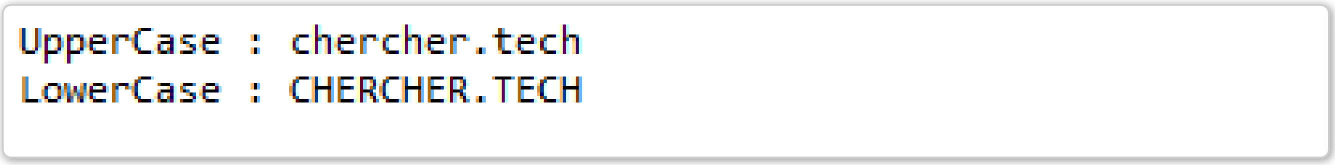


```
Index of he : 1
```

11. **toUpperCase /toLowerCase :** We can convert the string into UPPERCASE or to lowercase using string methods.

```
public static void main(String[] args) {
   String str = "CherCher.tech";
   System.out.println("UpperCase : " +str.toLowerCase());
   System.out.println("LowerCase : " +str.toUpperCase());
}
```

Output :



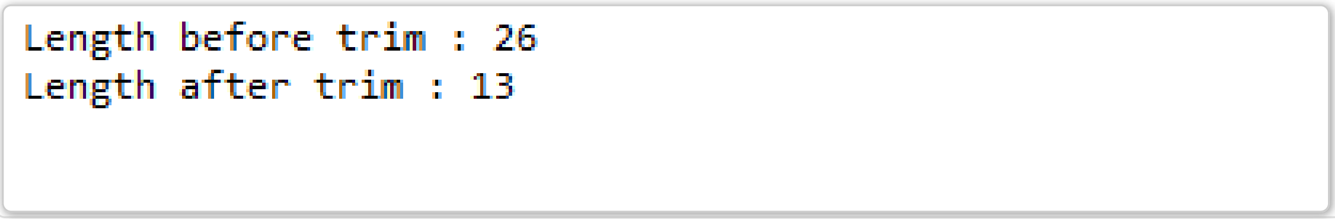
```
UpperCase : chercher.tech
LowerCase : CHERCHER.TECH
```

12. **trim() :** The trim method removes the spaces at the starting and end of the string and returns the string without spaces at the ends.

```
public static void main(String[] args) {

   String str = "       CherCher.tech       ";
   System.out.println("Length before trim : " +str.length());
   System.out.println("Length after trim : " +str.trim().length());
}
```

Output :



```
Length before trim : 26
Length after trim : 13
```