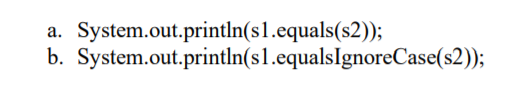
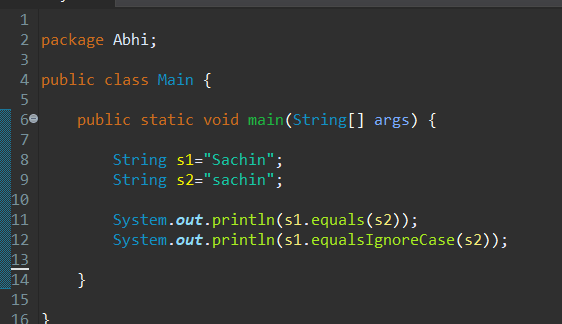
**String Class**

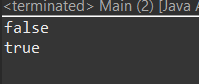
**Lab Assignment 4.1**

S. Abhishek

AM. EN. U4 CSE1947

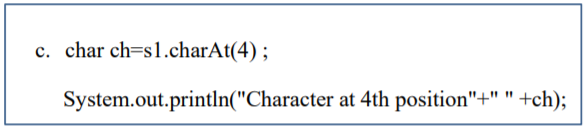


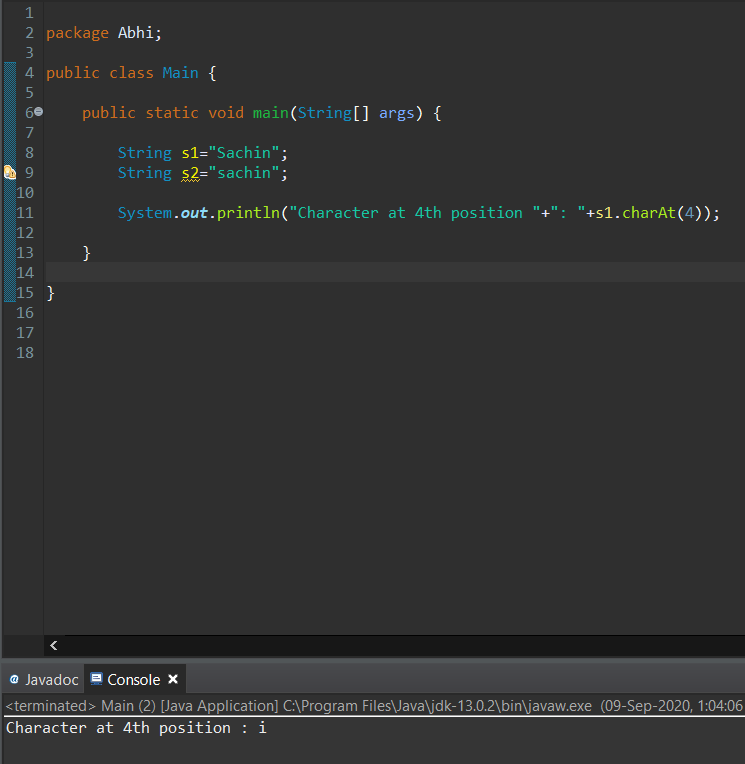


****

**A -** Equals() 🡪 It check’s whether the strings are equal or not and case sensitive. Return True if the two are same. Else return False.(Boolean)

**B -** EqualsIgnoreCase() 🡪 It check’s whether the strings are equal or not and case insensitive. Return True if the two are same. Else return False. (Boolean)

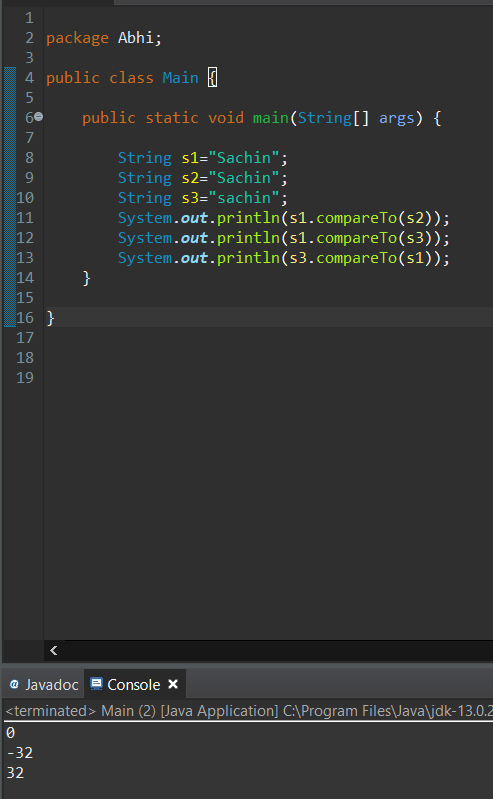
****

****

* Return’s the character at the given position.

s1.charAt(4) ; 🡪 Returns Character at the index 4.

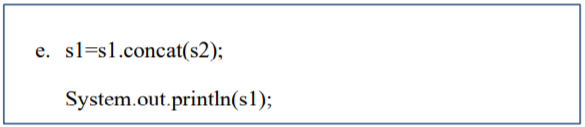


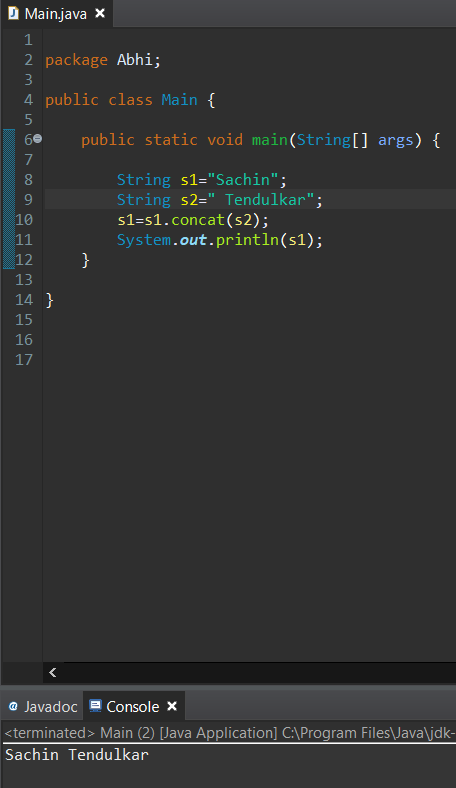


Returns 0 if s1 = = s2.

Returns 1 if s1 > s2.

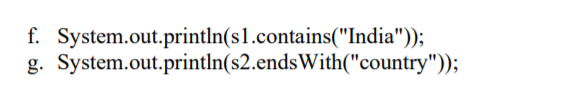
Returns -1 if s1 < s2.





It concatenates two String.

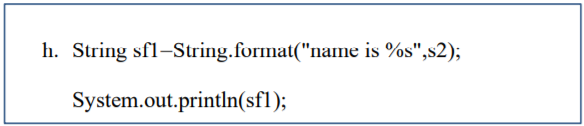
“Sachin” + “ Tendulkar” = “Sachin Tendulkar”.

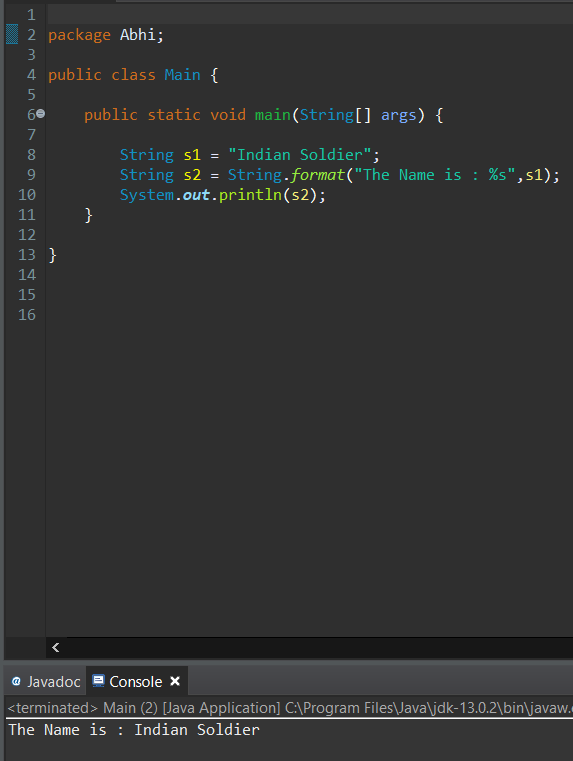




**F -** Returns True if it contains the Substring. Else Return False.

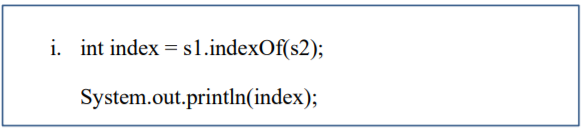
**G -** Returns True if it ends with the Substring. Else Return False.

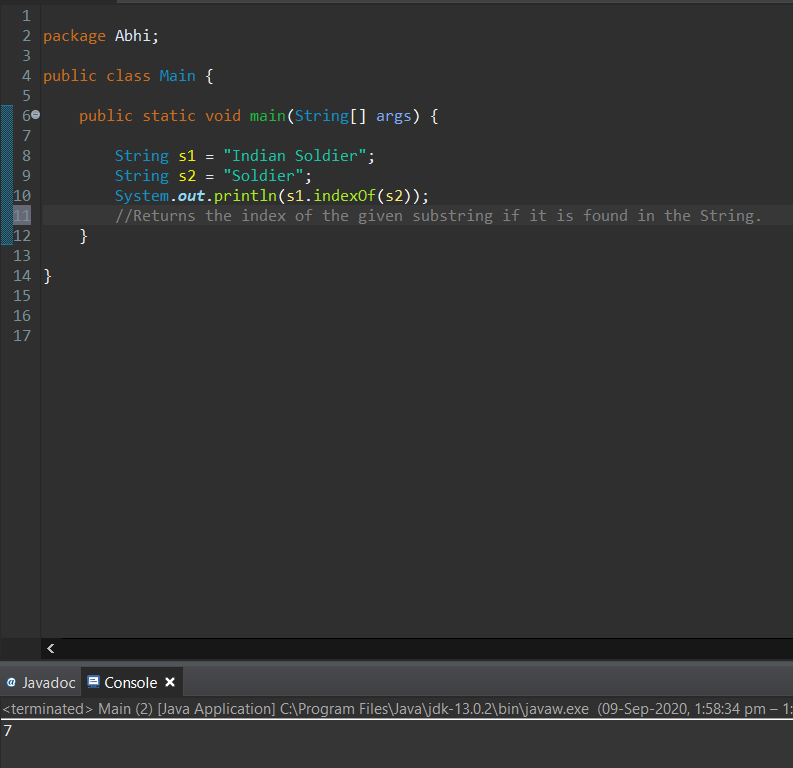




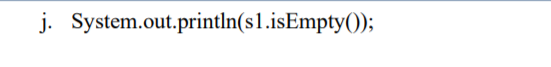
The **java string format()** method returns a formatted string using the given **locale**, specified **format string** and **arguments**.

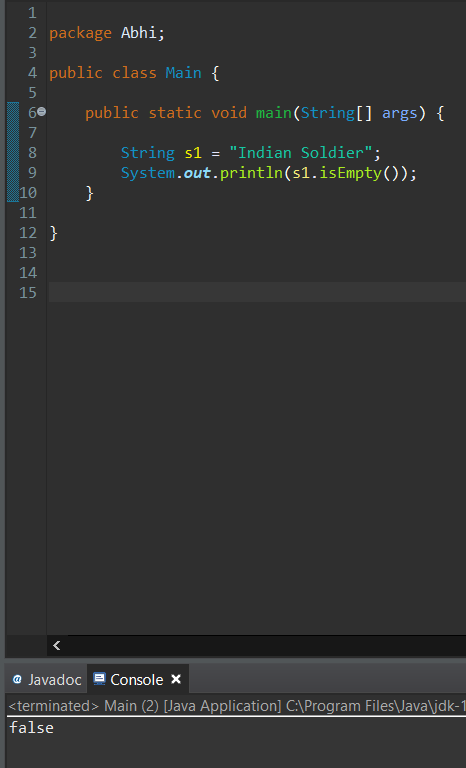
We can also concatenate the strings using this method and at the same time, we can format the output concatenated string.



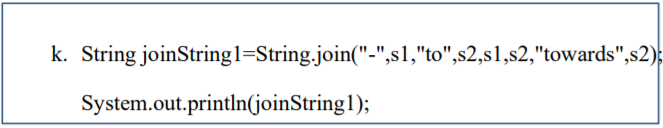


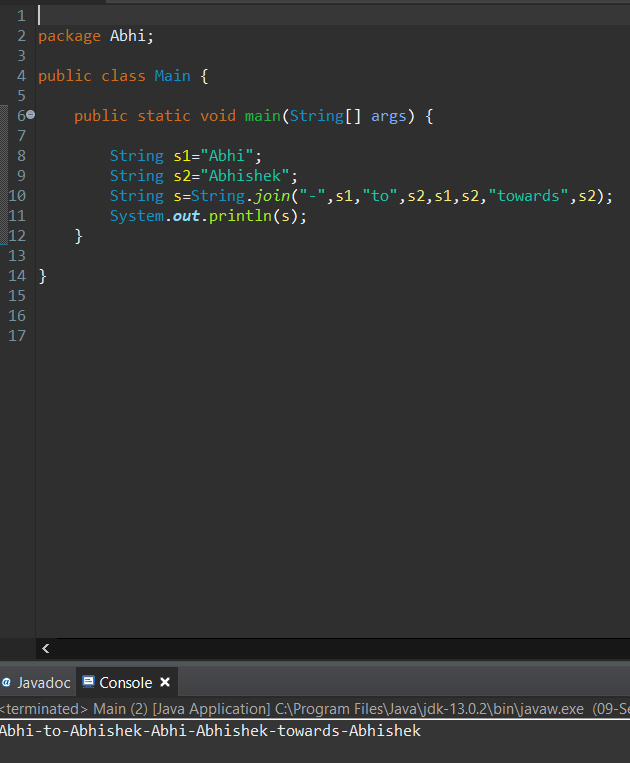
* This method **returns**the **index** within this string of the **first** occurrence of the specified character if it is found.
* Else Returns -1, if the character does not occur in the String.





Returns True if the String is Empty and False if it is not empty. It’s a Boolean.

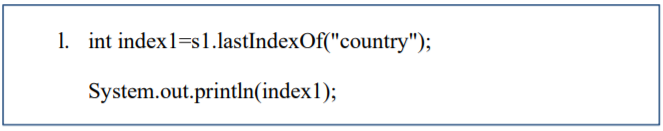


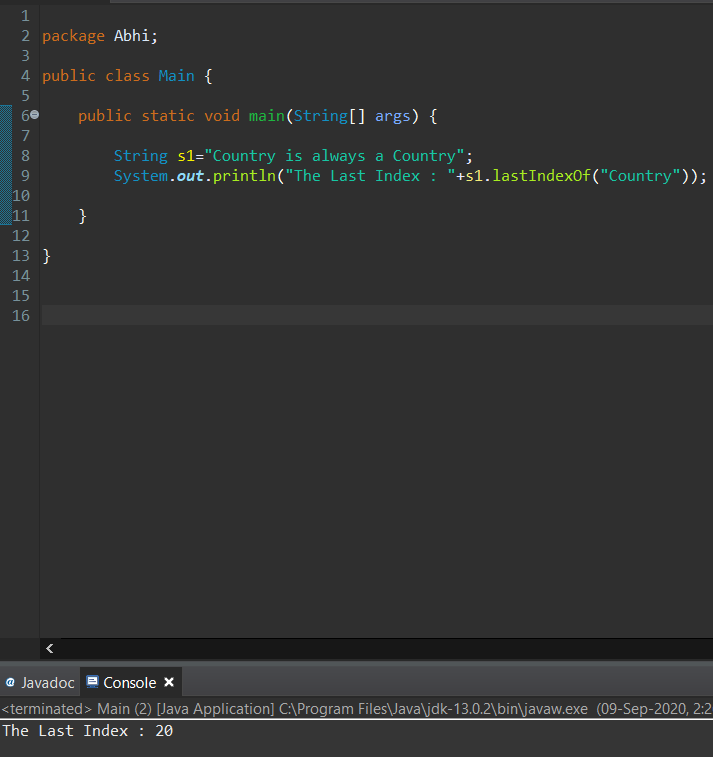


The **String.join()** method concatenates the given elements with the delimiter and returns the concatenated string.

Note that if an element is null, then null is added.

Here the Delimiter is provided as a first argument and it is ‘-‘.

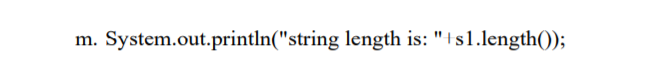


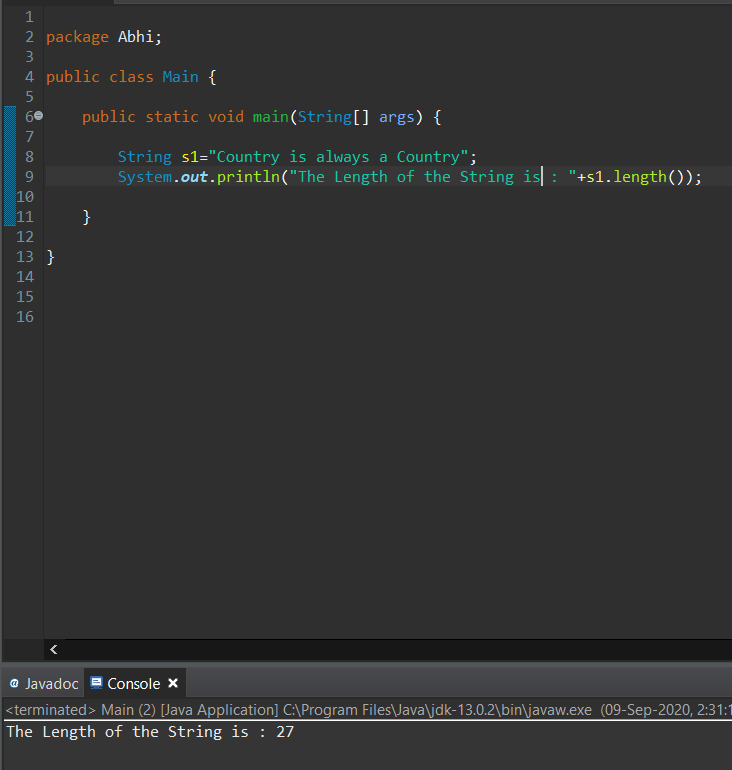


Here it returns the last index of the given substring.

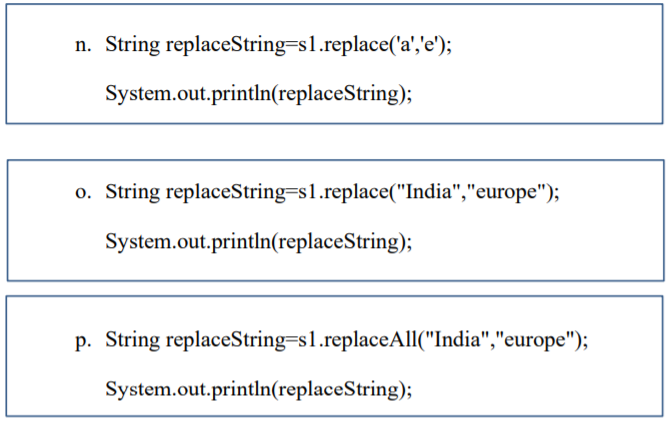
We can either pass the string directly as an argument or as other String.

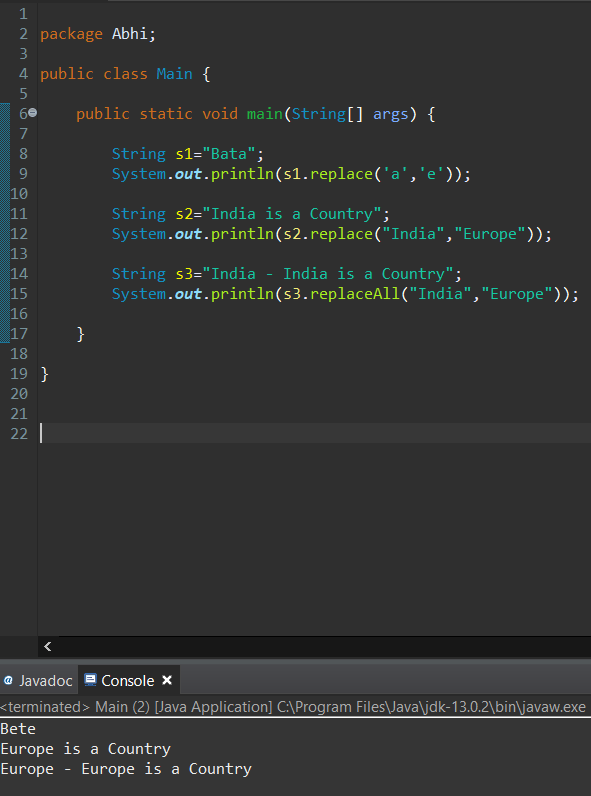
It returns the index if it is found. Else it returns -1 if the substring is not found.





It returns the length or the Size of the Given String.

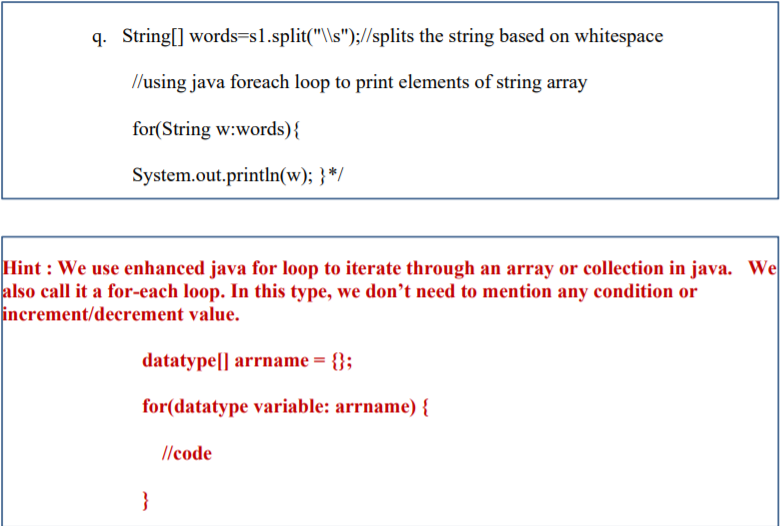




**N -** The replace() method replaces all the occurrences of old char with new char.

**O -** replaceAll() method replaces all the occurrences of old string with the new string.

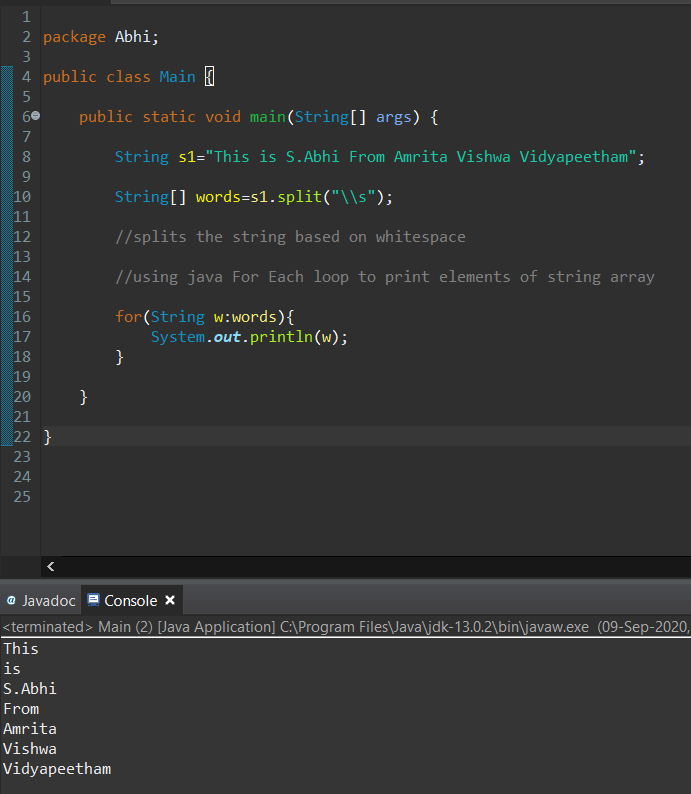
**P -** replace() works with replacing chars and replaceAll() works with replacing part of strings.



The split() method is used to split the String if it finds the given delimiter.

Here in this code [\\s](file:///\\s) is given which is basically a White space.

So when iterating the string through the For loop it separates and prints the output if it encounters [\\s](file:///\\s) (White Space).



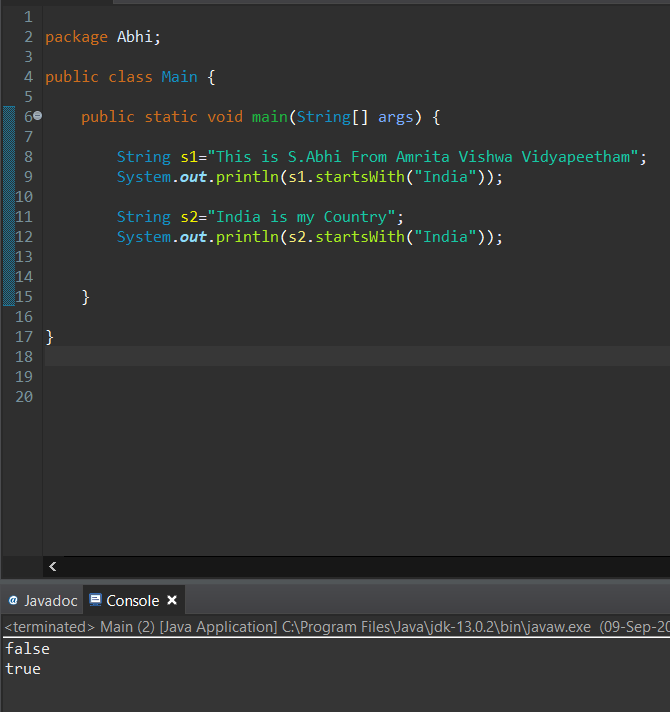


[Java String](https://howtodoinjava.com/java-string/) **startsWith()** method is used to check the prefix of string.

It verifies if given string starts with argument string or not.

**startsWith()** method is overloaded method and has two forms:

1. boolean startsWith(String str) – Returns true if the str is a prefix of the String Else Returns False
2. boolean startsWith(String str, int From-Index) – Returns true if the String begins with str starting from the specified index from-Index. Else Returns False

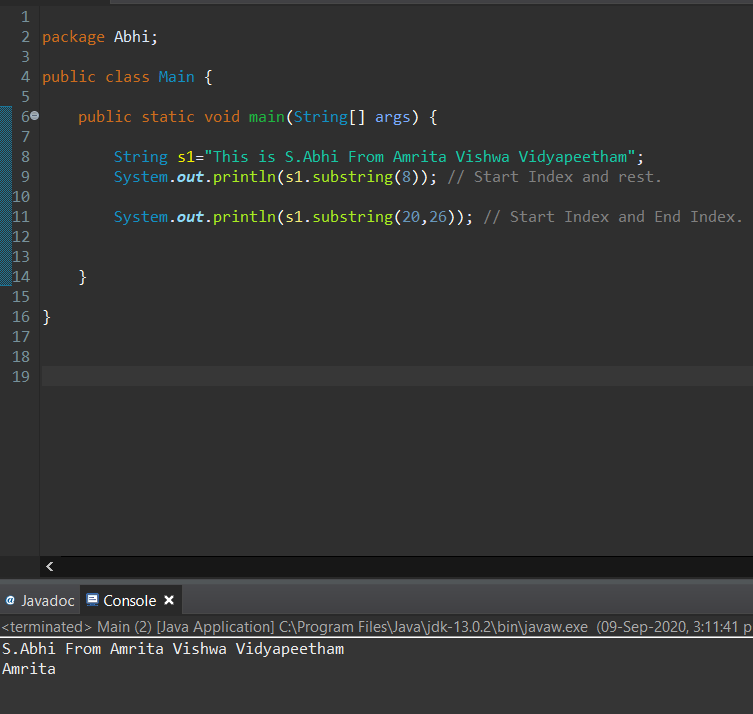


Here s1 does not start with “India”. So it returns False.

S2 starts with “India” and hence it returns True.



Here the First example is with only Start index and the second one is with both start and end.



Java [String](https://beginnersbook.com/2013/12/java-strings/) substring() method is used to get the substring of a given string based on the passed indexes.

**There are two ways we can use the substring() method:**

**When we pass only the starting index:**

String substring(int beginIndex)

Returns the substring starting from the specified index i.e beginIndex and extends to the character present at the end of the string.

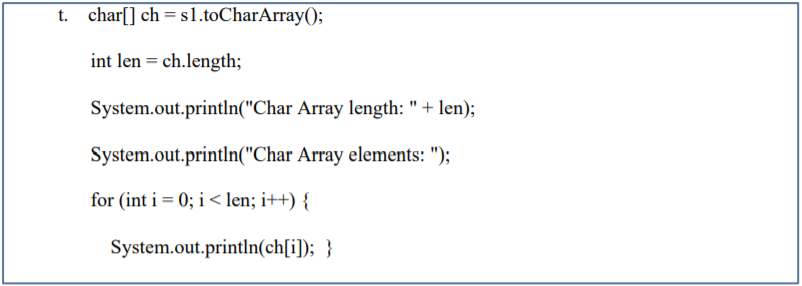
**When we pass both the indexes, starting index and end index:**

String substring(int beginIndex, int endIndex)

Returns a new string that is a substring of this string.

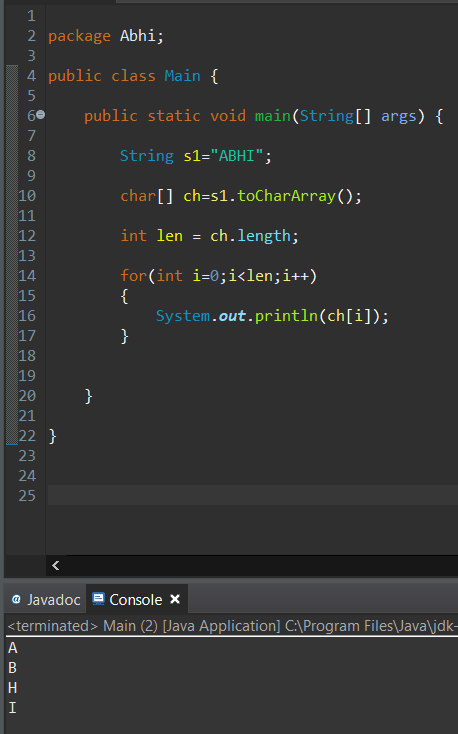
The substring begins at the specified beginIndex and extends to the character at index endIndex – 1.

Thus the length of the substring is endIndex-beginIndex.



The **java string toCharArray()** method converts this string into character array.

It returns a newly created character Array, its length is similar to this string and its contents are initialized with the characters of this string.



char[] ch = s1.toCharArray(); 🡪 **Converting the String to the** **Character Array.**

int len = ch.length; 🡪 **Finds the length of the Character Array.**

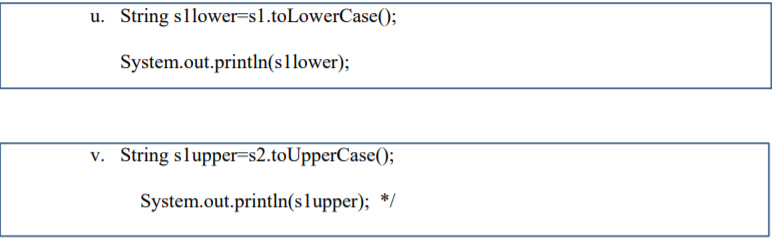
**Iterating through each Character:**

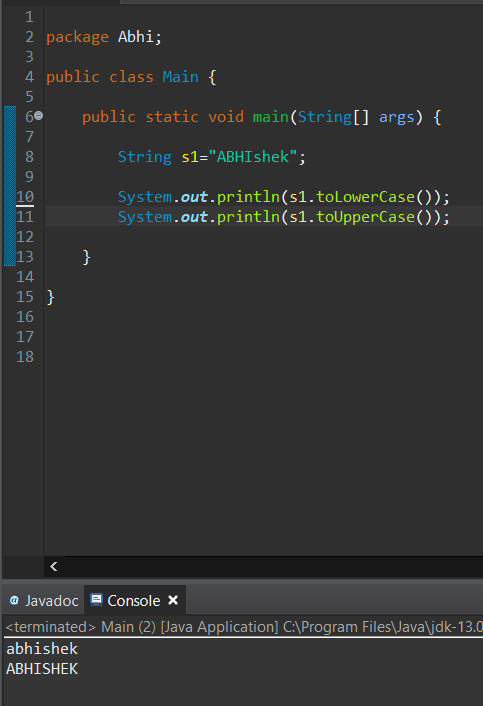
for(int i=0;i<len;i++)

{

System.***out***.println(ch[i]); 🡪 Printing Each Character.

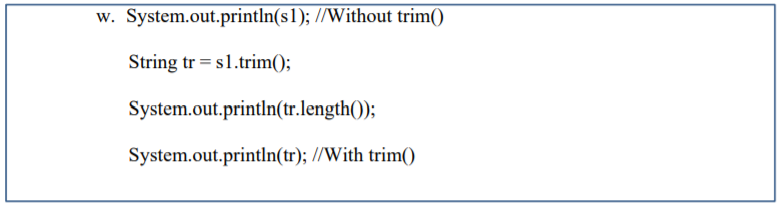
}





**U -** ToLowerCase() 🡪 Converts Upper case characters to Lower case.

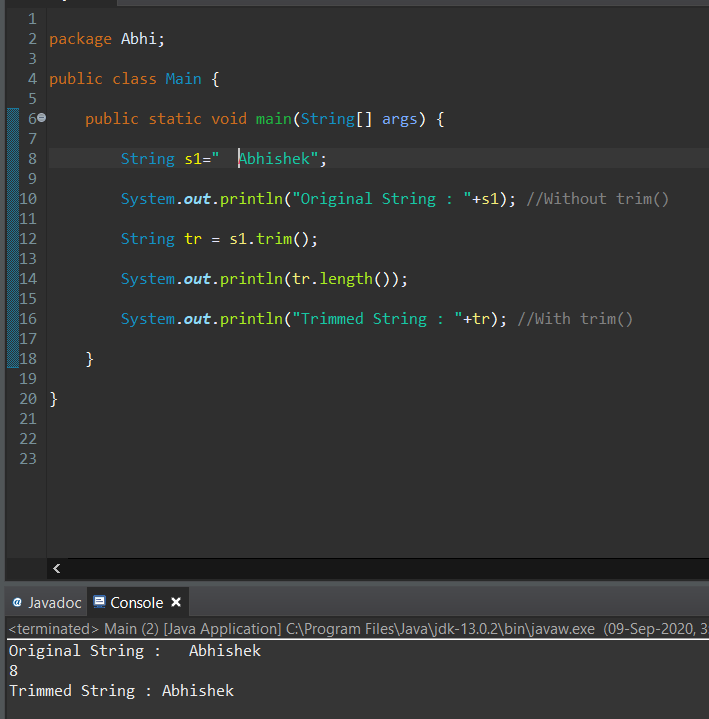
**V -** ToUpperCase() 🡪 Converts Lower case characters to Upper case.

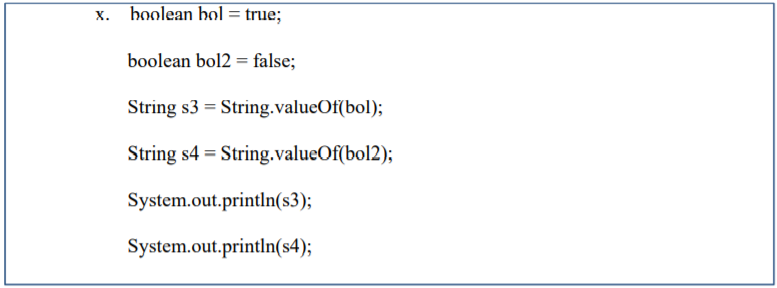


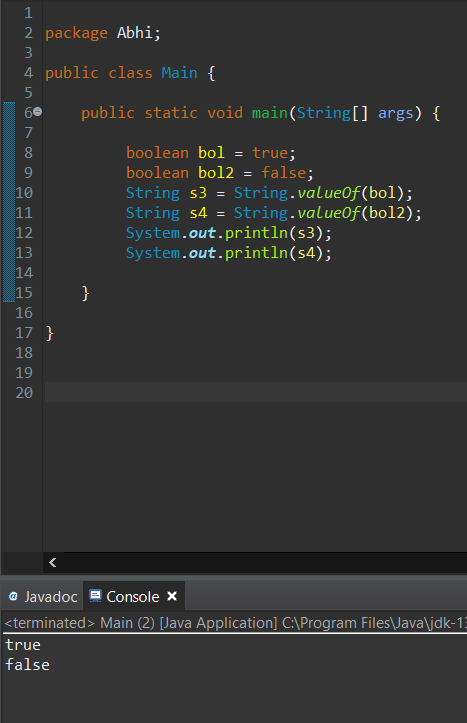
The trim() method removes whitespace from both ends of a string.

This method does not change the original string.

Removes White Space Infront of the Substring Abhishek.







This is a boolean version of overloaded valueOf() method.

It takes boolean value and returns a string.

**boolean** bol = **true**; 🡪 Assigning Bool Value True to the Boolean variable bol.

**boolean** bol2 = **false**; 🡪 Assigning Bool Value True to the Boolean variable bol2.

**String s1 = String.valueOf(bol);** 🡪 It takes boolean value of true and returns a string “true”.

**String s2 = String.valueOf(bol2);** 🡪 It takes boolean value of false and returns a string “false”.