L29

Friday, January 28, 2022 1:18 AM

Saurav Hathi

https://www.youtube.com/channel/UCp6MFWao5vWRnyRCxBsKnfw

https://www.instagram.com/saurav_hathi/

Text processing Methods – indexOf(), lastIndexOf(), substring(), replace(), trim(), split()

Q1. Some times in text processing, we have to check if a string or a particular character is contained within another string and if so at what position. The following methods help in handling such cases.

The method public int indexOf(String str) returns the index of the first occurrence of the specified substring. Example:
"man and woman and child".indexOf("and") returns 4
as the index of the first occurrence of 'and' is 4 within the string.
If we want to search for the index of a substring after a certain position within a string, we should use the method public int indexOf(String str, int fromIndex). It returns the index of the first occurrence of the specified substring, starting the search at the specified index. Example:
"man and woman and child".indexOf("and", 5) returns 14
as the index of the first occurrence of 'and' after index 5 is 14.
If we want the index position of the last occurrence of a string within another string, the method lastIndexOf(String str) can be used. It returns the index of the last occurrence of the specified substring. Example:
"rats and cats and dogs".lastIndexOf("s") returns 21
as the index of the last occurrence of 's' is 21.
Suppose we want to search for the last occurrence of a string within another string, but we want the search to begin at a certain index and then go backwards. The method lastIndexOf(String str, int fromIndex) is useful for doing this.
An example will make this functionality clear.

"ac & dc & fc &".lastIndexOf("&", 10) returns 8,
which points to the second occurrence of '&' in the string.
The first index of '&' is 3, second index is 8, and third index is 13.
We have specified that the search should begin at index 10.
So the method starts searching backwards starting from 10.
It finds '&' at position 8 and returns it.
There are equivalent methods to the above ones for char data type. We can simply use char instead of String in the above methods.

Retype the code below which demonstrates how to use the above methods.

```java
package q11171;
public class StringMethods {
    public static void main(String[] args) {
        String str = "special char &";
        System.out.println(str.indexOf("char"));
        System.out.println(str.indexOf(' '));
        System.out.println(str.lastIndexOf(' '));
        System.out.println(str.lastIndexOf("a", 8));
        System.out.println(str.indexOf("sp", 4));
    }
}
```

Q2. Write a class StringIndexOf with a **main** method. The method receives one command line argument. Print the **index** of the **first occurrences** of the string **CA** in the argument.

For Example:
Cmd Args : TCA*CX*CX*CAT
1

```java
package q11172;

public class StringIndexOf{
    public static void main(String[] args){
        System.out.println(args[0].indexOf("CA"));
    }
}
```

Q3. Write a class StringIndexOf with a **main** method. The method receives one command line argument. Print the second occurrence of the string **CA** in the argument.

For Example:
Cmd Args : CALIFORNICA
9

```java
package q11173;

public class StringIndexOf{
    public static void main(String[] args){
        System.out.println(args[0].indexOf("CA",2));
    }
}
```

Q4. Write a class StringLastIndexOf with a **main** method. The method receives one command line argument. Print the **index of** the **last occurrence** of the character ***** in the argument.

For Example:
Cmd Args : TCA*TX*TX*
9

```java
package q11174;

public class StringLastIndexOf{
    public static void main(String[] args){
        System.out.println(args[0].lastIndexOf("*"));
    }
}
```

Q5. Write a class StringLastIndexOf with a **main** method. The method receives one command line argument. Print the index of the **last but one** occurrence of the string **TX** in the given argument.

For example:
Cmd Args : TX12TX3TX4
4
Hint: You may want to store the lastIndex of **TX** first into a variable. And later explore how to use the method String.lastIndexOf(String str, int fromLastIndex)

```java
package q11175;

public class StringLastIndexOf{
    public static void main(String[] args){
        int index = args[0].lastIndexOf("TX");
        System.out.println(args[0].lastIndexOf("TX",index-1));
    }
}
```

Q6. To extract a string from another string, we use public String substring(int beginIndex) method. It returns a substring that starts at the specified index and extends to the end of the string. Examples:

"toBeOrNot".substring(2) returns "BeOrNot"
"Summers".substring(3) returns "mers"
"emptiness".substring(9) returns "" (an empty string)

If **beginIndex** is negative or larger than the length of the string, the method throws IndexOutOfBoundsException.

If we know at what indices the substring should begin and end in another string, we can use the method public String substring(int beginIndex, int endIndex) to extract the substring. The substring begins at the specified **beginIndex** and extends up to the character at **endIndex** - 1. **Note:** The character at the **endIndex** is **not** included in the resulting substring. Examples:

"hamburger".substring(4, 8) returns "urge"
"smiles".substring(1, 5) returns "mile"

The method throws IndexOutOfBoundsException if the **beginIndex** is negative, or **endIndex** is larger than the length of this String object, or **beginIndex** is larger than **endIndex**.

Retype the code below that demonstrates the usage of these methods.

```java
package q11176;
public class SubString {
    public static void main(String[] args) {
        String str = "An offer that you cannot refuse";
        System.out.println(str.substring(9));
        System.out.println(str.substring(3, 8));
        System.out.println(str.substring(str.indexOf("you")));
        System.out.println(str.substring(str.indexOf("that"), str.indexOf("refuse")));
    }
}
```

Q7. Write a class SubstringText with a **main** method. The method receives one command line argument. Print a **substring** of the argument that starts from index **7**.

For Example:
Cmd Args : Synchronicity
nicity
Note: Assume that the first argument will always contain more than **7** characters.

```java
package q11177;

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

        System.out.println(args[0].substring(7));
    }
}
```

Q8. Write a class SubstringText with a **main** method. The method receives one command line argument. Print the **substring** of the argument that starts at index **5** and ends at index **12** (should include the character at index 12).

Assumptions:
1. The argument contains more than 12 characters

For Example:
Cmd Args : Merry-go-round
-go-roun

```java
package q11178;

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

        System.out.println(args[0].substring(5,13));
    }
}
```

Q9. If we want to replace every character in a string with another character, the method public String replace(char target, char replacement) can be used.

Example:
"An Apple".replace('A', '@') returns "@n @pple"
There is an equivalent method to the above one where we can use strings as arguments instead of chars.
Examples:
"An Apple".replace("An", "The") returns "The Apple"
Many times during text processing, we should make sure that a string has no preceding or trailing space characters. A typical scenario is, if we are reading a name from a file, it is possible that there are extra spaces after the name in the file. When we compare it with an expected value, the comparison fails because of the spaces.

For example, if we read "Giza " from file and compare it with "Giza" using "Giza ".equals("Giza"), the method returns false. Solution to this problem is to use the public String trim() method. This method returns a string whose value is this string, with any leading and trailing white spaces removed.
Examples:
" Galaxy ".trim() returns "Galaxy"
"   Galaxy".trim() returns "Galaxy"
"Galaxy   ".trim() returns "Galaxy"
"Ga la xy".trim() returns "Ga la xy"

Retype the program below which demonstrates the usage of the above methods.

```java
package q11179;
public class StringMethods {
    public static void main(String[] args) {
        String str = " Six seasons   ";
        System.out.println("length = " + str.length());
        System.out.println("length after trim = " + str.trim().length());
        System.out.println(str.replace(' ', '*'));
        System.out.println(str.trim().replace("Six", "Four"));
    }
}
```

Q10. Write a class StringTrim with a **main** method. The method receives one command line argument, remove any **leading** or **trailing** white spaces from the argument and print the argument.

For Example:
Cmd Args : blank
blank

```java
package q11180;
public class StringTrim {
    public static void main(String[] args) {

        System.out.println(args[0].trim());
    }
}
```

Q11. Create a class StringReplace with a **main** method. The method receives one command line argument , **replace** all the occurrences of character **c** with **k** and print the result.

For Example:
Cmd Args : Acceptance
Akkeptanke

```java
package q11181;

public class StringReplace {
    public static void main(String[] args) {
        System.out.println(args[0].replace('c', 'k'));
    }

}
```

Q12. Suppose we have a string that contains a list of comma separated items like, "Apples, Oranges, Peaches, Bananas". We want to split the string extract all the items into an array. String class provides a method for doing this, called public String[] split(String regex).

The above string can be split using split method like this :
"Apples, Oranges, Peaches, Bananas".split(",").
It returns an array containing the items in the list.
When we print the array, it prints the following:
Apples
Oranges
Peaches
Bananas
Note the extra space before each item (except Apples).
The argument for the split method should be a regular expression. You will learn about using regular expressions in Java in some other section.

Retype the code below that demonstrates the usage of split method.

```java
package q11182;
public class StringMethods {
    public static void main(String[] args) {
        String str = "http://docs.oracle.com/javase/7/docs/api/";
        String[] splits = str.split("/");
        for (String s : splits) {
            System.out.println(s);
        }
    }
}
```

Q13. Create a class StringSplit with a **main** method. The method receives two command line arguments. First argument is a **string**. Second argument is a **regular expression**.

A regular expression is a sequence of characters that define a search pattern.

Example:
+, -, *,]
Split the first argument using the regular expression and print the items. Make sure there are no leading or trailing spaces in the items.

For Example:
Cmd Args: Rama-Lakshmana-Bharatha-Satrugna -
Rama
Lakshmana
Bharatha
Satrugna

```java
package q11183;

public class StringSplit {
    public static void main(String[] args) {
        String[] splits = args[0].split(args[1]);
        for (String s : splits) {
            System.out.println(s.trim());
        }


    }

}
```

Q14. Program to check the given String is Palindrome or not
Create a class PalindromeOrNot with a **main** method. The method receives one command line argument. Check the given argument is palindrome or not.

For example:
Cmd Args : madam
The given string madam is a palindrome
Cmd Args : Godavari
The given string Godavari is not a palindrome

```java
package q11184;

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

            String str = args[0];
            String reverse = new StringBuffer(str).reverse().toString();

             if (str.equals(reverse)){

                 System.out.println("The given string "+str+" is a palindrome");

             }else{

                  System.out.println("The given string "+str+" is not a palindrome");

             }
      }
}
```