*“Nothing succeeds like success.”*

*― Alexandre Dumas, Ange Pitou*

Dear readers, hope you are not bored yet. Are you? But this boredom will not last long, because we will now learn a new data structure in Java, known as ***Strings***.

Please watch this introductory video on [Strings](https://www.pepcoding.com/resources/online-java-foundation/string,-string-builder-and-arraylist/strings-intro-official/video) in Java, to learn about them before reading this article. This article will sum up all the points discussed in the video and, we will try to develop some more insights about Strings in Java.

* ***Definition***:
  + In Java, string is an *object* that represents an array or a sequence of characters.
  + A character (char) is a primitive-data type, whereas string in java is actually a non-primitive data type, because it refers to an object. The String object has methods that are used to perform certain operations on strings.
  + Name of the data type used to declare a string is ‘String’ and not ‘string’.
* ***Declaration***: We can declare strings using 2 methods:

1. Using *String Literal*: String literal is created by using double quotes. Example).

String s = "pepcoding";

1. Using ‘*new*’ keyword:

String s = new String("pepcoding");

**Note:** We will discuss how string is stored in memory after the next video on [String Interning and Immutability.](https://www.pepcoding.com/resources/online-java-foundation/string,-string-builder-and-arraylist/string-interning-immutability/video) Until then, you can stop worrying about memory management in Strings concept.

* **String Input and Output**

We can take string as input using two methods:

1. scn.next(): It will read the input until space (‘ ‘) or end-of-line has occurred.
2. scn.nextLine(): It will read the input until end-of-line has occurred.

Since, scn.nextLine() ignores spaces, a sentence with many words and spaces in between can be taken input in a single string object using it.

***Exercise***: Predict the output of the following 4 cases:

// Case 1

// Input:

// Pepcoding Rocks

String s1 = scn.next();

String s2 = scn.next();

System.out.println(s1);

System.out.println(s2);

// Case 2

// Input:

// Pepcoding

// Rocks

String s1 = scn.next();

String s2 = scn.next();

System.out.println(s1);

System.out.println(s2);

// // Case 3

// // Input:

// // Pepcoding Rocks

String s1 = scn.nextLine();

String s2 = scn.next();

System.out.println(s1);

System.out.println(s2);

// Case 4:

// Input:

// Pepcoding Rocks

// And Shines

String s1 = scn.nextLine();

String s2 = scn.nextLine();

System.out.println(s1);

System.out.println(s2);

Answers:

*Case 1:*

pepcoding

Rocks

*Case 2:*

Pepcoding

Rocks

Case 3:

***Exception (No Such Element).*** But why? See s1 takes the input string from the entire line, hence it will store the entire ‘Pepcoding Rocks’ and there is no input left for s2 (in the next line). Hence a run-time error/exception is thrown.

Case 4:

Pepcoding Rocks

And Shines

* **In-Built Methods (Functions) in Strings (Provided by Java)**

1. *int* ***length****()*

It returns the length of the string. For eg) For string str = “pepcoding”, str.length() will return 9.

Note, that length is a method/ member function in strings, whereas it was a data member in case of arrays. Hence ‘()’ are a must to invoke the function call.

1. *char* ***charAt****(int index)*

It takes an integer input which represents an index of the string (whose method charAt is called) and returns the character present at that index.

For eg) str.charAt(5) will return the character at index = 5, (in 0-based indexing).

String str = "pepcoding";

char ch = str.charAt(5);

system.out.println(ch);

The code above will print character at index = 5, i.e. ‘d’.

***Note***: Index passed should be a valid index (from 0 to length of string - 1), otherwise it will give IndexOutOfBound Exception.

**Exercise**: Given a string, print all it’s characters in separate lines.

We will loop through for index = 0 to str.length() - 1, and print all characters using charAt method.

String str = "pepcoding";

for(int i=0; i<str.length()-1; i++) {

char ch = str.charAt(i);

system.out.println(ch);

}

***Important Note***: We cannot set/modify any character of a string by modifying str.charAt(index). (It will give an error that *str.charAt(index)* is not a variable but a value).

This is because **STRINGS ARE IMMUTABLE IN JAVA**. I don’t want you to think much about ‘**WHY’** aboutthis currently, we will have a complete video lecture and an article to explain immutability of strings in Java.

1. *String str.****substring****(int start\_index, int end\_index)*

A substring of a string is a contiguous section of characters from the string. If we want to extract characters starting from a given index and ending at an index (ending index should be after the starting index), then we can take help of this inbuilt method.

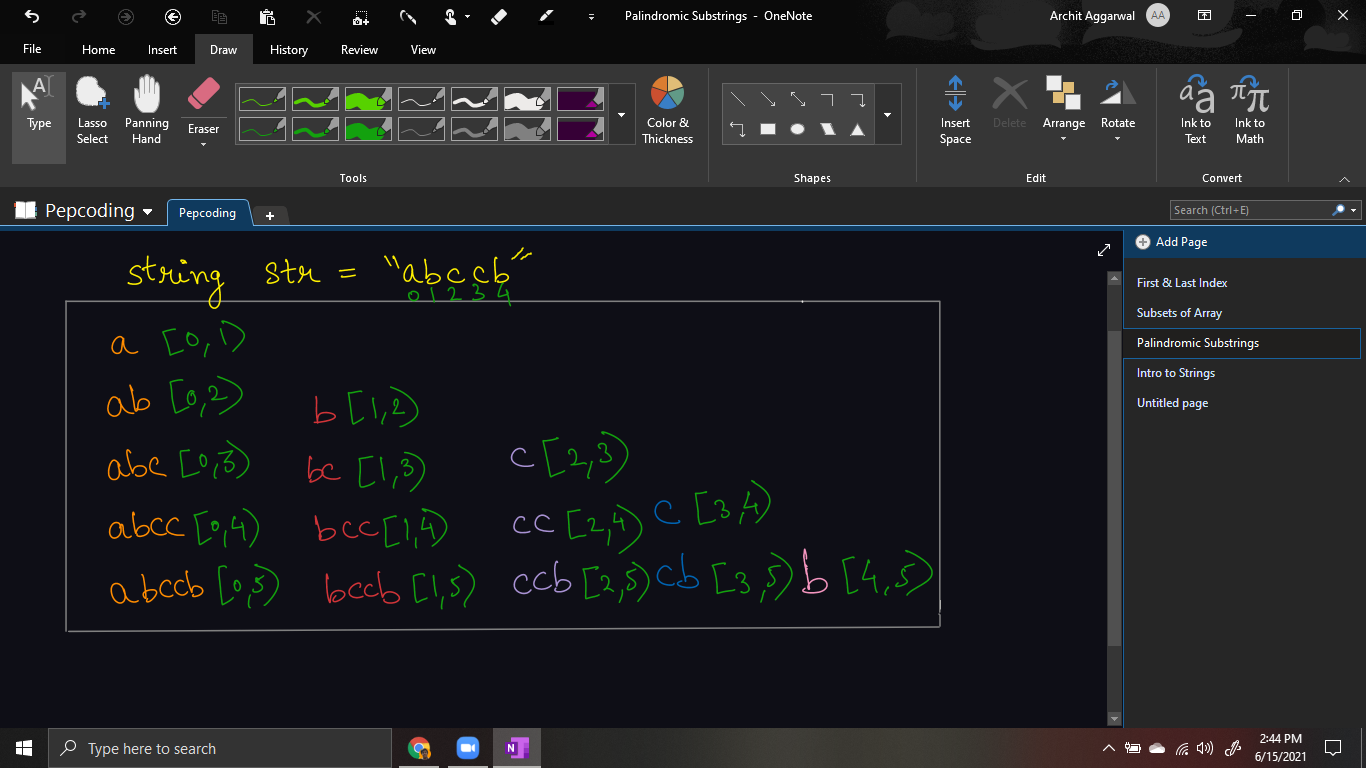
This method will return a substring of string, from the start\_index till end\_index - 1. (i.e. it will *not include the end\_index*). Hence, if we want to get a substring from 0th index to 5th index, we must call str.substring(0, 6) instead of str.substring(0, 5).

***Note***: If starting index will be the same as ending index, then since substring() method does not include ending index in substring, it will return an empty string “”.

***Note***: If the ending index is taken less than the starting index, then it does not represent a valid substring, (For eg, “ba” is not a substring of “abc”.) Hence, it will give OutOfBoundException.

***Note***: If we will not pass the ending index, and make a call to substring() method with only 1 parameter (starting index), then it will return the substring starting from start\_index and upto the string’s length, hence end\_index will get default value as as str.length().

**Exercise**: Generate all substrings of a string str, using nested loops and substring() method.



Generating all substrings is discussed in detail in the next problem: ‘[Print All Palindromic Substrings](https://www.pepcoding.com/resources/online-java-foundation/string,-string-builder-and-arraylist/print-all-palindromic-substrings-official/ojquestion)’ in detail, however, you can get a feel about it, and try to understand the code provided.

String str = "abccb";

for(int i=0; i<str.length()-1; i++){

for(int j=i+1; j<=str.length(); j++){

System.out.println(str.substring(i, j));

}

}

1. *Concatenate Two Strings*

We can concatenate two strings using the ‘+’ operator.

For example)

String a = "Hello";

String b = "Pepcoders";

String c = a + " " + b;

System.out.println(c);

It will give the result as “Hello Pepcoders”, as strings a, “ “, and b will be concatenated and stored in c.

Interesting part is, we can even concatenate an integer to the string.

For example)

String a = "Hello";

String c = a + " " + 30;

System.out.println(c);

It will store “Hello 30” in string c, and print Hello 30.

**Exercise:** Guess the output.

System.out.println("Hello" + 10 + 20);

System.out.println(10 + 20 + "Hello");

In Java, expressions are read from left to right (for +, -, \*, / operators etc.), i.e. + is said to have left to right associativity.

Hence in the first statement, first “Hello” and 10 are concatenated which results in “Hello10”. Now “Hello10” and 20 are concatenated which will print “Hello1020”

But in the second statement, first 10 and 20 are added, and then the result 30 is concatenated with “Hello”, which will print the result “30Hello”.

1. *String[]* ***split****(String regex)*

We can split our string into an array of strings, whenever we find a substring *regex* regular expression.

A simple example is, let’s suppose string str represents a long sentence with spaces in between. We want to store the words in the sentence in an array of strings. We can use this function to do so in a single line:

String str = "We are learning strings";

String words[] = str.split(" ");

for(int i=0; i<words.length; i++){

System.out.println(words[i]);

}

words is an array of strings which will contain each space-separated word of the string { “We”, “are”, “learning”, “strings”}.

***Note***: Since, words is an array of string (and not string), hence we have used words.length and not words.length().

We can even separate/split the string based on “,” or any other regular expression. But, please keep in mind, that parameter that split expects is a string and not a character.

* There are some more methods available with String, like *contains(), toLowerCase(), toUpperCase(), trim(),* etc. but we will learn about them as and when we encounter them.
* One important method remaining is ***equals***() and the difference between equals() method and == operator which we will try to cover in the immutability in string video and article.

Contributor : Archit Aggarwal