Strings in Java

Characteristics of char variables:

- Only one character.

- Primitive.

- Single quotes: ‘a’

Characteristics of String variables:

- Zero or more characters.

- A String is an object.

- A String variable is therefore an object reference variable.

- Double quotes: “Aa”

- ‘A’ does not equal “A”

- A literal is a value of the type we’re talking about.

Integer literals: 3, 5, 10, 3285

Floating point literals: 3.14, 1.168

Boolean literals: Only two (true and false)

Character literals: ‘a’ , ‘&’ , ‘3’

String literals: “Computer Science”

- We want to be able to use Strings as variables. First we need to be able to declare our variables.

String Declaration:

String myString;

(Data type: String.)

- Variables aren’t much use until we give them value. One way to do that is to call a constructor, just like we did with our SavingsAccount variables.

Initialization:

myString = new String(“Roses are red”);

myString = “Roses are red”;

(These both work.)

- Another thing that is special about string objects is that they’re immutable. You cannot change them. But we can make the variable point at a new object. Remember that it is a reference variable, and assignment of one reference variable to another will align memory addresses.

- So we have this myString variable that is pointing at the actual object “Roses are red”. When we do the assignment of the myString gets “Violets are blue”, the original object hasn’t changed at all. We’ve just changed the value of the object reference variable myString so that it’s now pointing at the new value, the new object “Violets are blue.”

- We cannot change the String object itself.

- Strings are made up of characters. We can identify the individual characters by index. Notice that the indices begin at zero.

Text

Description automatically generated

- Note that spaces are also valid characters that have indices.

- Objects have methods that we can call so we can do STUFF.

char charAt( int index )

- This is a method that returns the character at the given index in the string.

int length ()

int strLen = myString.length();

- Returns the length of the string. There are no parameters because it’s just working on the string itself.

In the case of “Roses are red” above, the String length is 13 indices.

String to UpperCase()

String upperStr = myString.toUpperCase();

- Returns a copy of the string with all letters uppercase. Notice that this is a copy of the original string. We cannot modify the original string. So the things that are doing modifications to the values of the string are returning a new string with those new values.

String toLowerCase()

- Does the exact same thing as UpperCase().

int indexOf(String searchString)

Int searchStart = myString.indexOf(“are”);

- indexOf is nice if we want to figure out where something is in a string. This first indexOf method takes a string and returns the index of the beginning of the first occurrence of searchString in the string. Returns -1 if not found.

If we recall the “Roses are red” string, this method would output “6” because 6 is the index where “are” begins. The letter a is at index 6.

Int indexOf(String searchString, int start)

String string2 = “abracadabra”;

int strLoc = string2. indexOf(“ab”, 2 );

- This indexOf method will allow us to input an index starting point of the string. We specify start from this point.

Example:

a. b. r. a . c . a . d. a. b. r. a.

(0) (1) (2) (3) (4) (5) (6) (7) (8) (9) (10)

This will find the string “ab” after the start point of index 2.

So the output will be 7 because this is the occurrence of “ab” in terms of the indices after 2.

int indexOf( char searchChar)

int searchStart = myString.indexOf(“ “);

- So this will look for a specific character from the very beginning of the index, as we’ve seen in the previous method for strings.

- Returns the index of the beginning of the first occurrence of searchChar in the String; returns -1 if not found.

- This takes chars instead of whole Strings.

String substring( int startIndex, endIndex)

- Returns a copy of the portion of the string beginning at startIndex and ending with endIndex.

- Whatever is in between the startIndex and endIndex-1 we’ve provided will be returned as a copy. So it will not include the character at the endIndex.

Example:

Let’s return to “Roses are red”.

String secondWord = myString.substring( 6 , 9 );

Will return just “are”. No more and no less. It won’t include the space at index 9 because it only goes up until that point.

This is why the expected output is explicitly describe as

endIndex -1.

String substring( int startIndex)

String thirdWord = myString.substring(10);

- Returns a copy of the remainder of the string beginning at startIndex.

- Starting at index 10 it will return the third word “red” for the string “Roses are red”.

b. i. g. w. h. i. t. e. h. o. u. s. e.

(0). (1). (2). (3). (4). (5). (6). (7). (8). (9). (10). (11). (12). (13). (14).

I have a string variable named str1 that has the value "big white house". Which of the following statements will correctly store the value "white" in str2?

1. String str2 = str1.substring(3,9);
2. String str2 = str1.substring(4,9);
3. String str2 = str1.substring(5,9);

It’s named “str2” because we’re creating a copy of the string with the substring method. We can’t assign it to “str1” because a string cannot be changed once it has been declared.

[Check Java String API.](https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/lang/String.html)