Strings

Strings, which are widely used in Java programming, are a sequence of characters. In the Java programming language, strings are objects.

The Java platform provides the String class to create and manipulate strings.

Java provides the String classfrom the java.lang package to create and manipulate strings. String has its own methods for working with strings. The String class is not a primitive data type.

A string is a sequence of characters. Every character in a string has a specific position in the string, and the position of the first character starts at index 0. The length of a string is the number of characters in it. For example:

String	"Sunny Day"								
Character in the string	'S'	'u'	'n'	'n'	'y'	' '	'D'	'a'	'y'
Position of the character in the string	0	1	2	3	4	5	6	7	8

The length of the string **"Sunny Day"** is 9.

CREATING AN STRING

A String can be constructed by either of the following:

• directly assigning a string literal to a String object; or

• using the new keyword and String constructor to create a String object. The following statements are the most direct ways to create a string in Java:

string strGreeting = “Sunny Day”;

String strName;

strName = “Erick Lopez”;

In the example statements, “Sunny Day” and “Erick Lopez” are both string literals (note: string literals in Java are series of characters that will appear in output exactly as entered. String literals are constants.). Whenever the JVM encounters a string literal in a program, it creates a String object with its value.

The following statement creates a `String` object in Java using the new keyword and the `String` constructor:

```
String strGreeting = new String("Hello World!");
```

The string literals with the same contents share the same memory location, while the constructed strings using the new keyword are stored in different memory locations.

Consider the following `String` declarations:

```
String str1 = "Computer";
String str2 = "Computer";
String str3 = new String("Computer");
String str4 = new String("Computer");
```

The `String` method, `equals()`, is used to compare the content of two (2) strings. The relational operator (==) is used to compare the references of two (2) objects. The following example statements compare the string literals and `String` objects:

```
str1 == str2      // returns true, same memory location and same content
str1 == str3      // returns false, different memory locations
str1.equals(str3) // returns true, same contents
str3 == str4      // returns false, different memory locations
str3.equals(str4) // returns true, same contents
```

Unlike primitive data types, the `String` class is immutable; once it is created, its contents cannot be changed. For example, consider the statements:

```
int num = 0;
num++;
```

The variable, num, changed its value from `0` to `1`, but in a `String` variable, its content cannot be changed by a method.

STRING METHODS

Method	Description	Example for `String str = "Java";`
`charAt(index)`	Returns the character of a string based on the specified index	`str.charAt(2); //returns char 'v'`
`compareTo(string)`	Compares a string with another string. The method returns 0 if the string is equal to the other string. A value less than 0 is returned if the string is less than the other string (less characters) and a value greater than 0 if the string is greater than the other string (more characters).	`str.compareTo("Java's"); /*returns an integer value, the operation's result is a negative integer*/`
`concat(string)`	Returns a new string concatenated with the value of the parameter	`str.concat(" program"); /*returns a new string "Java program"*/`
`equals(string)`	Returns `true` if this string and the specified string are equal; otherwise, returns `false`	`str.equals("Java"); /*returns boolean value true*/`
`equalsIgnoreCase(string)`	Returns `true` if this string and the specified string are equal, considering the uppercase and lowercase versions of a letter to be the same; otherwise, returns `false`.	`str.equals("java"); /*returns boolean value true*/`
`indexOf(string)`	Returns the index of the first occurrence of the specified string within this string. If not found, the method returns -1.	`str.indexOf("a"); //returns int value 1`
`lastIndexOf(string)`	Returns the index of the last occurrence of the specified string within this string. If not found, the method returns -1.	`str.lastIndexOf("a"); //returns int value 3`
`length()`	Returns the length of the string in `int` type	`str.length(); //returns 4`
`toLowerCase()`	Returns a new string having the same characters as this string, but with any uppercase letters converted to lowercase. The content of this `String` object is unchanged.	`str.toLowerCase(); /*returns a new string "java", but the content of variable str is unchanged*/`
`toUpperCase()`	Returns a new string having the same characters as this string, but with any uppercase letters converted to uppercase. The content of this `String` object is unchanged.	`str.toUpperCase(); //returns a new string "JAVA"`
`replace(old_Char, new_Char)`	Returns a new string having the same characters as this string, but with each occurrence of specified *old_char* replaced by *new_char*. The content of this `String` object is unchanged.	`str.replace('a', 'o'); //returns a new string "Jovo"`
`substring(start_index)`	Returns a new string having the same characters as the substring begins at specified start index through to the end of	`str.substring(2); //returns a new string "va"`

Method	Description	Example for `String str = "Java";`
	the string. The content of this `String` object is unchanged.	
`substring(start, end)`	Returns a new string having the same characters as the substring that begins at specified index *start* through to but not including the character at index *end*. The content of this `String` object is unchanged.	`str.substring(1, 3);` `//returns a new string "av"`
`trim()`	Returns a new string having the same characters as this string, but with leading and trailing whitespace removed.	`String str = " Java Java ";` `str.trim(); /*returns a new string "Java Java"*/`

Note: You can use a variable to specify the parameters of a method. For example:

```
String str1 = "Java", str2 = " program";
System.out.println(str1.concat(str2));
```