Strings

String simply means sequence of characters.

Strings in java are objects that are backed internally by char arrays, since arrays are immutable(can not grow) strings are immutable too. Whenever a change to string is made , an entirely new string is created.

```
String str = "Coding";

String str = new String("Coding");
```

String Methods

Method	parameters	Description	Return type
charAt()	int index	Returns the char at specified index	char
codePointAt()	int index	Returns the unicode value of the char at specified index	int
compareTo()	String str	Compares two strings lexicographically	int
compareToIgnoreCase()	String str	Compares two strings lexicographically, ignoring case differences	int
concat()	String str	Appends str to the end of the another string	String

contains()	String str	Check whether the string contains specified sequence of characters	boolean
copyValueOf()	char[] chr	Returns a String that represents the characters of the character array	String
endsWith()	String str	Checks whether a string ends with the specified character(s)	boolean
equals()	String str	Compare two strings. Returns true if the strings are equal, and false if not	boolean
equalsIgnoreCase()	String str	Compares two strings, ignoring case considerations	boolean
format()	String format, String arg	Returns a formatted string using the specified locale, format string, and arguments	String
indexOf()	String str / char ch	Returns the position of the first found occurrence of specified characters in a string	int
isEmpty()	-	Checks whether a string is empty or not	boolean
lastIndexOf()	String str / char ch	Returns the position of the last found occurrence	int

		of specified characters in a string	
length()	-	Returns length of the string	int
replace()	char oldChar , char newChar	Searches a string for a specified value, and returns a new string where the specified values are replaced	String
split()	String str	Splits a string into an array of substrings	String[]
startsWith()	String str	Checks whether a string starts with specified characters	boolean
endsWith()	String str	Checks whether a string ends with specified characters	boolean
substring()	int startIndex / int startIndex , int endIndex	Returns a new string which is the substring of a specified string	String
toCharArray()	-	Converts this string to a new character array	char[]
toLoweCase()	-	Converts string to lowercase	String
toUpperCase()	-	Converts string to uppercase	String
toString()	-	Returns the value of String object	String

trim()	-	Removes the white space from both end of the string	String
valueOf()	-	Returns the string representation of the specified value	String

How are Strings stored ?

```
String str = "abc";
```

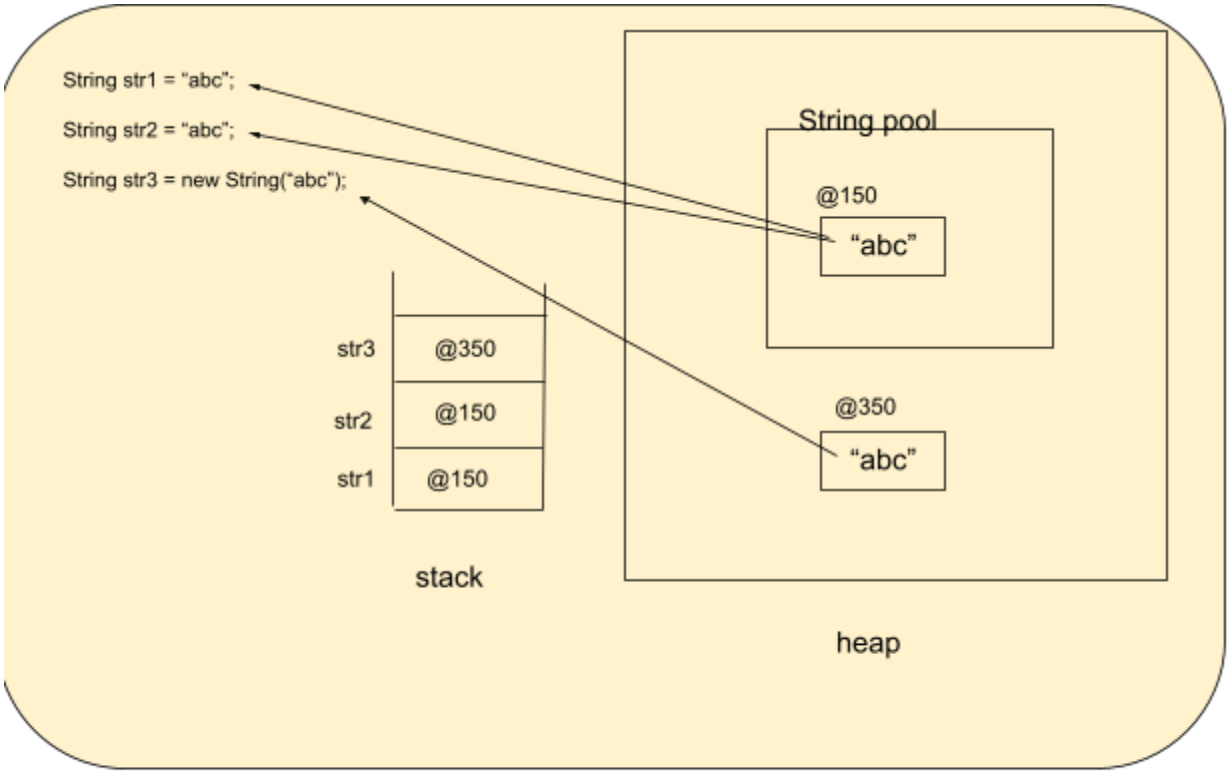
Java stores this string in a memory area called **string pool** which is basically a small area of heap memory only.

```
String str2 = new String("abc");
```

Java stores this string separately in heap as we have specified the new keyword that means we don't want to use the string pool and we want new storage space.

String pool

Before storing string in string pool java runtime machine checks whether the same string already exists in the string pool or not and if it does then it just simply returns the reference to that object.



String Immutability

Strings in java are immutable, that means once the string is created you can not change any character at any index or can not increase the size of the string so there is no such function like **str.setChar(i)** in java.

The reason for this is that java stores strings in the string pool and returns the reference to string variables so let us assume that java allows the string mutability then it would happen that one variable changes any character and the change is reflected in all the variables that were using the same reference which is not intended. That is why strings are immutable in java.

```
String str = "abc";
str = str + "def";   // this is correct in java ! , why?
```

Here java creates new string `"abcdef"` in string pool and changes the str reference to this new string

```
str = "xyz";
```

This is also right, here new string `"xyz"` is created and str reference is changed to new string

Comparing non-primitives

```
int[] arr = {1,2};
int[] arr2 = {1,2};
if (arr==arr2) {
    System.out.println("equal");
}else{
    System.out.println("not equal");
}

//output: not equal

System.out.println(arr + " "+ arr2);

//output: [I@58d25a40 [I@1b701da1
```

Here output is not equal because arr and arr2 contents are stored in different space in heap and their references are stored in arr and arr2 as variable so when we are doing arr == arr2 , we are actually comparing the value of variable not which is address , and which is also different for both arr and arr2.

```
String str = "abc";
String str2 = "abc";

if (str==str2) {
    System.out.println("equal");
}else{
    System.out.println("not equal");
}

//output: equal
```

This is because there is only one `"abc"` in the string pool and both str and str2 have reference to that so when str == str2 is performed it turns out to be true.

```
String str = "abc";
String str2 = new String("abc");

if (str==str2) {
    System.out.println("equal");
}else{
    System.out.println("not equal");
}

//output: not equal
```

This is because here str2 is not in string pool so the references are different so str == str2 turns out to be false

---

```
String str = "abc";
String str2 = new String("abc");

if (str.equals(str2)) {
    System.out.println("equal");
}else{
    System.out.println("not equal");
}

//output: equal
```

Because equals function does not compare references but contents of the str and str2.

String vs StringBuffer

```
StringBuffer str = new StringBuffer("hello");
str.setCharAt(0, 'H');
System.out.println(str);

//output: Hello
```

When you string is changing very often then it is wiser to use StringBuffer instead of String

StringBuffer Methods:

Methods	Action Performed
append()	Used to add text at the end of the existing text.
length()	The length of a StringBuffer can be found by the length( ) method

capacity()	the total allocated capacity can be found by the capacity( ) method
charAt()	
delete()	Deletes a sequence of characters from the invoking object
deleteCharAt()	Deletes the character at the index specified by loc
ensureCapacity()	Ensures capacity is at least equals to the given minimum.
insert()	Inserts text at the specified index position
length()	Returns length of the string
reverse()	Reverse the characters within a StringBuffer object
replace()	Replace one set of characters with another set inside a StringBuffer object