String Concatenation by concat()

The String concat() method concatenates the specified string to the end of current string.

Syntax: **public** String concat(String another)

Let's see the example of String concat() method.

Example9

```
class Demo
{
    public static void main(String[] args)
    {
        String s1 = "java";
        String s2 = "python";
        String s3 = s1.concat(s2);
        System.out.println(s3);

    }
}
```

Output: **javapython**

Although it is important to note that Strings are immutable. Now lets see what does this mean.

Immutable String in java

In java, **string objects are immutable**. Immutable simply means unmodifiable or unchangeable.

Once string object is created its data or state can't be changed but a new string object is created.

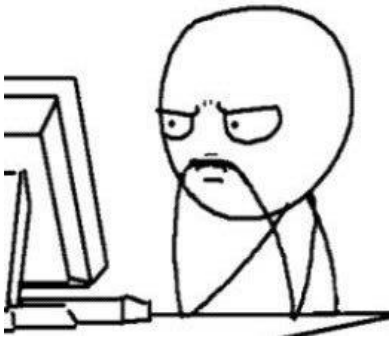
Let's try to understand the immutability concept by the example given below:

Example10

```
class Demo
{
    public static void main(String[] args)
    {
        String s1 = "java";
        String s2 = "python";
        System.out.println(s1);
        System.out.println(s2);
        s1.concat(s2);
        System.out.println(s1);
        System.out.println(s2);
    }
}
```

Output:

```
java
python
java
python
```



In java whenever an immutable string is attempted to be modified then that string will not be modified, instead another string is created in non-constant pool with new reference. And as of now no object is assigned to store the new reference, which is why we cannot see the concatenated string.

So now let's see how to get the concatenated string…

```
class Demo
{
    public static void main(String[] args)
    {
        String s1 = "java";
        String s2 = "python";
        System.out.println(s1);
        System.out.println(s2);
        s1 = s1.concat(s2);
        System.out.println(s1);
        System.out.println(s2);
    }
}
```

Output:

```
java
python
javapython
python
```

Previously we had a reference for the concatenated string but we did not assign it to anything, whereas here we are assigning it back to s1 so that now s1 is pointing to the new concatenated string.

Why string objects are immutable in java?

Because java uses the concept of string literal. Suppose there are 5 reference variables, all refer to one object "java". If one reference variable changes the value of the object, it will be affected to all the reference variables. That is why string objects are immutable in java.

Going further.. If you think we forgot about the third way of comparing strings that is with the help of *compareTo()* then certainly we haven't….So let's see how to do it.

The String compareTo() method compares values lexicographically and returns an integer value that describes if first string is less than, equal to or greater than second string.

Suppose s1 and s2 are two string variables. If:

- **s1 == s2** :0
- **s1 > s2** :positive value
- **s1 < s2** :negative value

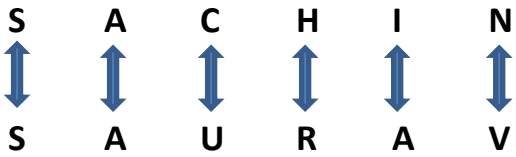
Confused??? Let's understand with example.

Case1:

S1 = “SACHIN”

S2 =“SAURAV”

Comparison happens as shown below



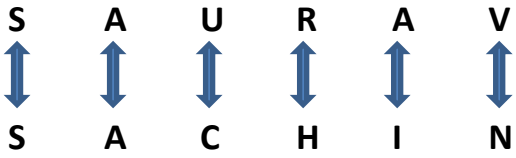
Each character is compared in **lexical order**. In the above example S,A are same in both string but when C and U are compared we know that **C comes before U** therefore **s1 is considered lesser than s2**. Which will give **negative number** when printed on screen.

Case2:

S1= “SAURAV”

S2= “SACHIN”

Comparison happens as shown below



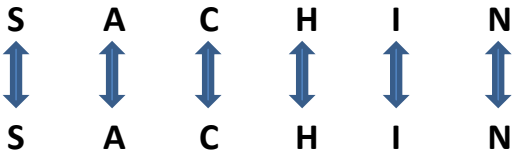
In the above example S,A are same in both string but when U and C are compared we know that **U comes after C** therefore **s1 is considered greater than s2**. Which will give **positive number** when printed on screen.

Case3:

S1 = “SACHIN”

S2 =“SACHIN”

Comparison happens as shown below



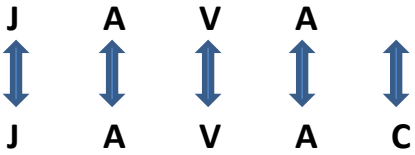
Above we see that all the **characters are same** in both the strings. Therefore when compared character by character there is **no character greater or smaller** than the other which means it will give the result as 0 when printed on the screen.

Case4:

S1= "JAVA"

S2= "JAVAC"

Let's now see what happens when the size of the string is different.



We can see that both the strings contain "JAVA" but **the second string has extra character C**. Therefore **second string is considered to be greater than the first**. And when printed on screen we will get **negative number**.

There are many such string methods, let's have a look at them…

contains() :

The **java string contains()** method searches the sequence of characters in this string. It returns *true* if sequence of char values are found in this string otherwise returns *false*.

Syntax: string_name.contains("sequence");

Returns: true if sequence of char value exists, otherwise **false**.

Throws a **NullPointerException** if the sequence is null.

endsWith() :

The **java string endsWith()** method checks if this string ends with given suffix. It returns true if this string ends with given suffix else returns false.

Syntax: string_name.endsWith("sequence or character");

Returns: true or false.

indexOf() :

The **java string indexOf()** method returns index of given character value or substring. If it is not found, it returns -1. The index counter starts from zero.

There are 4 types of indexOf method in java. The signature of indexOf methods are given below:

No.	Method	Description
1	int indexOf(int ch)	Returns index position for the given char value
2	int indexOf(int ch,int fromIndex)	Returns index position for the given char value and from index
3	int indexOf(String substring)	Returns index position for the given substring
4	int indexOf(String substring, int fromIndex)	Returns index position for the given substring and from index

Parameters:

- **ch**: char value i.e. a single character e.g. 'a'
- **fromIndex**: index position from where index of the char value or substring is retured
- **substring**: substring to be searched in this string

Returns: index of the string

charAt() :

The **java string charAt()** method returns *a char value at the given index number*.

The index number starts from 0 and goes to n-1, where n is length of the string. It returns **StringIndexOutOfBoundsException** if given index number is greater than or equal to this string length or a negative number.

Returns : char value

startsWith() :

The **java string startsWith()** method checks if this string starts with given prefix. It returns true if this string starts with given prefix else returns false.

Syntax :
public boolean startsWith(String prefix)
public boolean startsWith(String prefix, int offset)

parameters : prefix : Sequence of character

Returns: true or false

substring() :

The **java string substring()** method returns a part of the string.

We pass begin index and end index number position in the java substring method where start index is inclusive and end index is exclusive. In other words, start index starts from 0 whereas end index starts from 1.

There are two types of substring methods in java string.

Syntax:

1. **public** String substring(**int** startIndex)
2. **public** String substring(**int** startIndex, **int** endIndex)

parameters:

- **startIndex** : starting index is inclusive
- **endIndex** : ending index is exclusive

Returns: specified string

Throws: StringIndexOutOfBoundsException if start index is negative value or end index is lower than starting index.

toLowerCase() :

The **java string toLowerCase()** method returns the string in lowercase letter. In other words, it converts all characters of the string into lower case letter.

The toLowerCase() method works same as toLowerCase(Locale.getDefault()) method. It internally uses the default locale.

Syntax:

- **public** String toLowerCase()
- **public** String toLowerCase(Locale locale)

The second method variant of toLowerCase(), converts all the characters into lowercase using the rules of given Locale.

Returns: string in lowercase

toUpperCase() :

The **java string toUpperCase()** method returns the string in uppercase letter. In other words, it converts all characters of the string into upper case letter.

The toUpperCase() method works same as toUpperCase(Locale.getDefault()) method. It internally uses the default locale.

Syntax:

- **public** String toUpperCase()
- **public** String toUpperCase(Locale locale)

The second method variant of toUpperCase(), converts all the characters into uppercase using the rules of given Locale.

Returns: string in uppercase

There are definitely many other methods which you will be exploring once you start practicing.