Java String Notes

Contents

[What is a String in Java? 1](#_Toc82936034)

[How to create String object? 1](#_Toc82936035)

[1) String Literal 1](#_Toc82936036)

[2) By new keyword 2](#_Toc82936037)

[Comparing Strings: 3](#_Toc82936038)

[1) By equals() method 3](#_Toc82936039)

[2) By == operator 4](#_Toc82936040)

[3) By compareTo() method: 4](#_Toc82936041)

[Replacing characters or substrings in a String 4](#_Toc82936042)

[Methods that are covered in chapter 4, Mathematical Functions, Characters, and Strings: 6](#_Toc82936043)

# What is a String in Java?

In Java, a String is basically an object that represents sequence of char values.

An array of characters works same as Java string. For example:

char[] ch = {'C','S','C', 'I', '2', '3', '0','2'};

String s = new String(ch);

is same as:

String s = "CSCI2302";

The Java.lang.String class implements *Serializable*, *Comparable* and *CharSequence* interfaces.

The Java String is immutable, meaning it cannot be changed but a new instance is created. For a mutable class, you can use StringBuffer and StringBuilder class.

# How to create String object?

There are two ways to create String object:

By String literal 🡪 how we did in CSCI 1302

By new keyword 🡪 how we are learning now

## 1) String Literal

Java String literal is created by using double quotes. For Example:

String s = "welcome";

Each time you create a String literal, the JVM checks the string constant pool first. If the String already exists in the pool, a reference to the pooled instance is returned. If String doesn't exist in the pool, a new string instance is created and placed in the pool. For example:

String s1 = "Welcome";

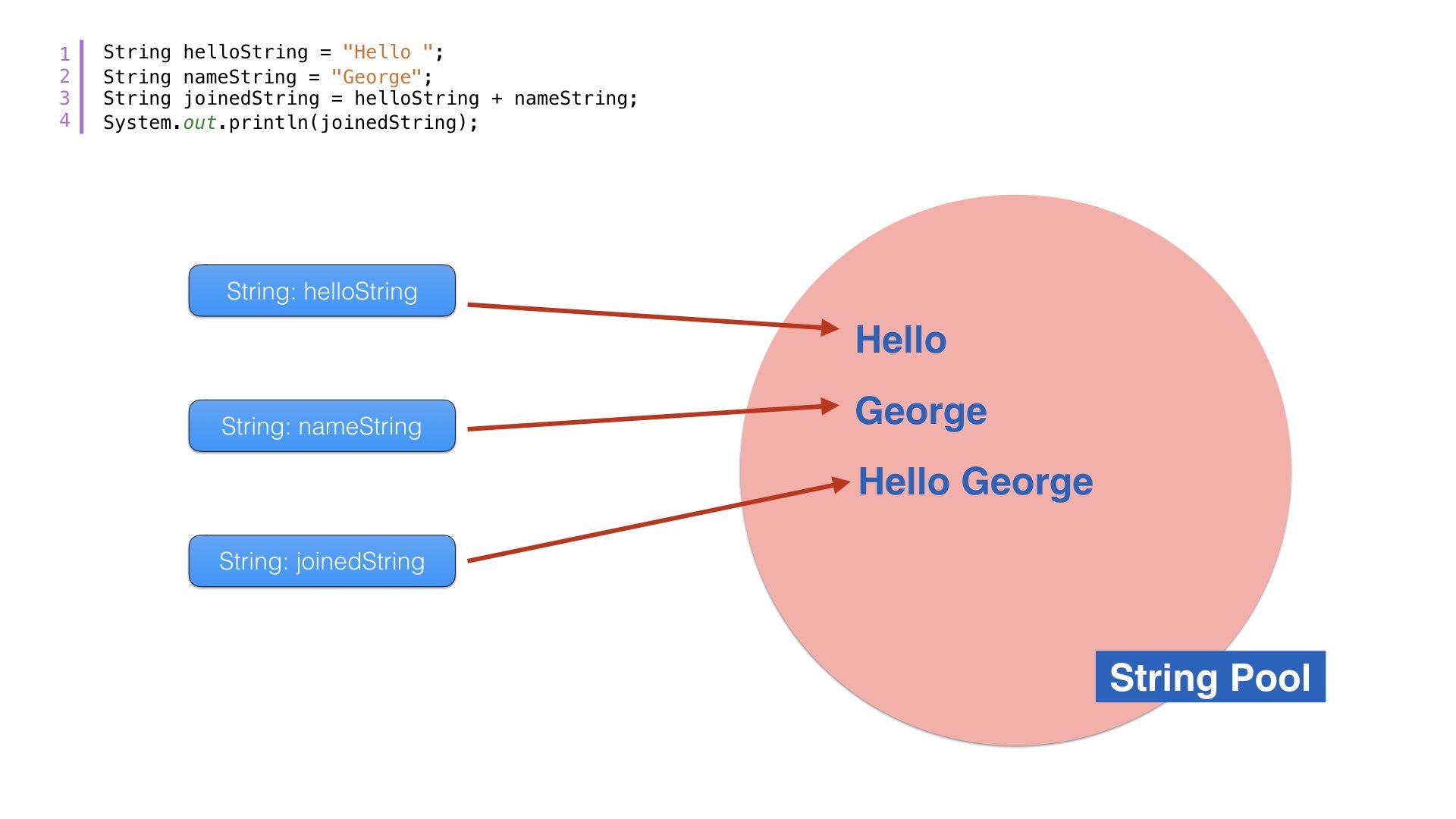
String s2 = "Welcome"; //will not create new instance

Welcome

s2

s1

In the above example only one object will be created. Firstly JVM will not find any string object with the value "Welcome" in String constant pool, so it will create a new object. After that, it will find the String with the value "Welcome" in the pool, it will **not** create new object but will return the reference to the same instance.



Note: String objects are stored in a special memory area known as string constant pool

**Why Java uses concept of string literal?**

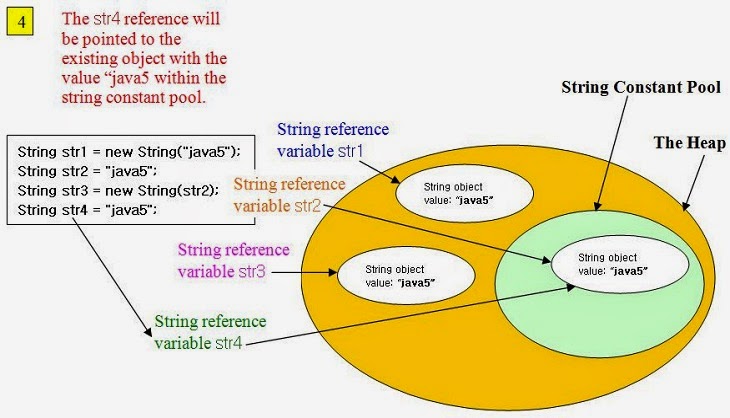
To make Java more memory efficient (because no new objects are created if it exists already in string constant pool).

## 2) By new keyword

String s = new String("Welcome");

//creates an object and one reference variable

In this case, JVM will create a new string object in normal (non pool) heap memory The variable s will refer to the object in heap (non pool).



# Comparing Strings:

There are three ways to compare String objects:

1. By equals() method
2. By = = operator
3. By compareTo() method

## 1) By equals() method

equals() method compares the original content of the String. It compares values of String for equality. String class provides two methods:

public boolean equals(Object another){} compares this string to the specified object.

public boolean equalsIgnoreCase(String another){} compares this String to another String, ignoring case.

String s1 = "CSCI2302";

String s2 = "CSCI2302";

String s3 = new String("CSCI2302");

String s4 = "CSCI1302";

System.out.println(s1.equals(s2));//true

System.out.println(s1.equals(s3));//true

System.out.println(s1.equals(s4));//false

//Example of equalsIgnoreCase(String) method

String s1 = "csci2302";

String s2 = "CSCI2302";

System.out.println(s1.equals(s2)); //false

System.out.println(s1.equalsIgnoreCase(s3)); //true

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## 2) By == operator

The = = operator compares the main memory – not the values of the reference objects.

String s1 = "CSCI2302";

String s2 = " CSCI2302";

String s3 = new String("CSCI2302");

System.out.println(s1==s2);//true (because both refer to same instance)

System.out.println(s1==s3);//false(because s3 refers to instance created in nonpool)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## 3) By compareTo() method:

compareTo() method compares values and returns an int which tells if the values compare less than, equal, or greater than.

Suppose s1 and s2 are two String variables. If:

• s1 == s2 :0

• s1 > s2 :positive value

• s1 < s2 :negative value

String s1 = "CSCI2302";

String s2 = "CSCI2302";

String s3 = "csci2302";

System.out.println(s1.compareTo(s2));//0

System.out.println(s1.compareTo(s3));//-32(because s1 > s3)

System.out.println(s3.compareTo(s1));//32(because s3 < s1 )

# Replacing characters or substrings in a String

Methods:

|  |  |
| --- | --- |
| [**String**](https://docs.oracle.com/javase/7/docs/api/java/lang/String.html) | [**replace**](https://docs.oracle.com/javase/7/docs/api/java/lang/String.html#replace(char,%20char))(char oldChar, char newChar)  Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar. |
| [**String**](https://docs.oracle.com/javase/7/docs/api/java/lang/String.html) | [**replace**](https://docs.oracle.com/javase/7/docs/api/java/lang/String.html#replace(java.lang.CharSequence,%20java.lang.CharSequence))(**[CharSequence](https://docs.oracle.com/javase/7/docs/api/java/lang/CharSequence.html" \o "interface in java.lang)** target, **[CharSequence](https://docs.oracle.com/javase/7/docs/api/java/lang/CharSequence.html" \o "interface in java.lang)** replacement)  Replaces each substring of this string that matches the literal target sequence with the specified literal replacement sequence. |
| [**String**](https://docs.oracle.com/javase/7/docs/api/java/lang/String.html) | [**replaceAll**](https://docs.oracle.com/javase/7/docs/api/java/lang/String.html#replaceAll(java.lang.String,%20java.lang.String))([**String**](https://docs.oracle.com/javase/7/docs/api/java/lang/String.html) regex, [**String**](https://docs.oracle.com/javase/7/docs/api/java/lang/String.html) replacement)  Replaces each substring of this string that matches the given [**regular expression**](https://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html#sum) with the given replacement. |
| [**String**](https://docs.oracle.com/javase/7/docs/api/java/lang/String.html) | [**replaceFirst**](https://docs.oracle.com/javase/7/docs/api/java/lang/String.html#replaceFirst(java.lang.String,%20java.lang.String))([**String**](https://docs.oracle.com/javase/7/docs/api/java/lang/String.html) regex, [**String**](https://docs.oracle.com/javase/7/docs/api/java/lang/String.html) replacement)  Replaces the first substring of this string that matches the given [**regular expression**](https://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html#sum) with the given replacement. |

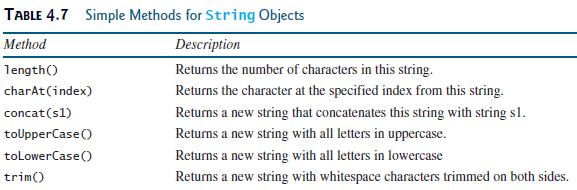
"Welcome".replace('e', 'A') returns a new string, WAlcomA.

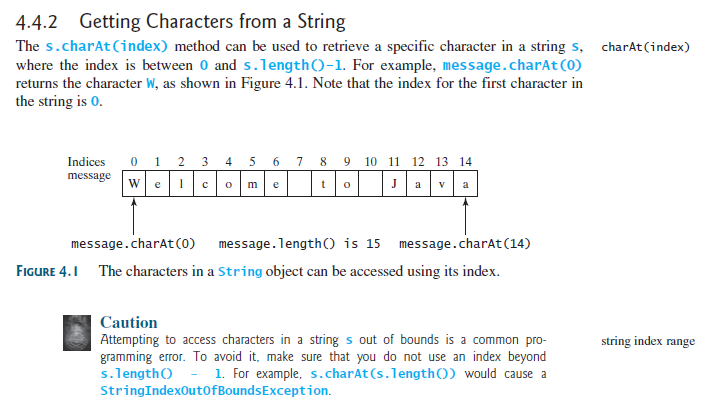
"Welcome".replaceFirst("e", "AB") returns a new string, WABlcome.

"Welcome".replace("e", "AB") returns a new string, WABlcomAB.

"Welcome".replace("el", "AB") returns a new string, WABcome.

# Methods that are covered in chapter 4, Mathematical Functions, Characters, and Strings:





You can use the concat method to concatenate two strings. The statement shown below, for example, concatenates strings s1 and s2 into s3:

String s3 = s1.concat(s2);

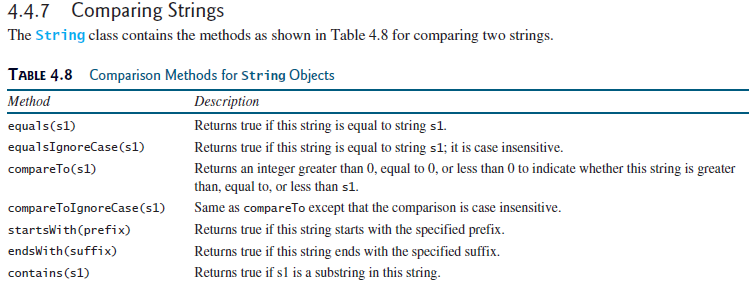
It is the same as doing: String s4 = s1 + s2;

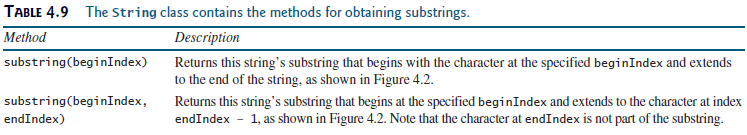
The + can be used in concatenation, as here, and in addition, as in int num = 2 + 2;

In order to be used in concatenation, at least one of the operands has to be a String.

When you are joining two or more words, you have to make sure that you are thinking of the space between the words. For instance, if you are joining the word hi and the word there, you want the new word to be hi there, not hithere.







Notice the beginIndex and endIndex values in the substring method; beginIndex values means starting at this spot, the endIndex value means up to – NOT including – that index spot.

