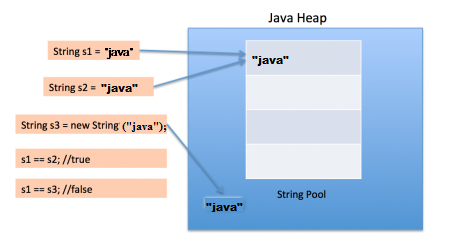
<https://www.studytonight.com/java/string-handling-in-java.php>

* String is a class in java.
* String class is encapsulated under java.lang package.
* every string that you create is actually an object of type **String**.
* string objects are immutable that means once a string object is created it cannot be altered.
* An object whose state cannot be changed after it is created is known as an Immutable object. String, Integer, Byte, Short, Float, Double and all other wrapper classes objects are immutable.

String Literal

String literal is a simple string enclosed in double quotes " ". A string literal is treated as a String object.

Each time you create a String literal, the JVM checks the string pool first. If the string literal already exists in the pool, a reference to the pool instance is returned. If string does not exist in the pool, a new string object is created, and is placed in the pool. String objects are stored in a special memory area known as **string constant pool** inside the heap memory.

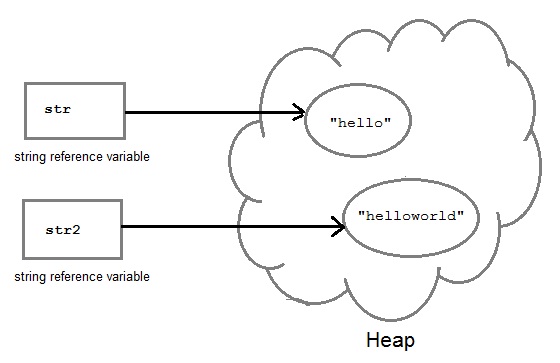


String with new Keyword

Always new object is created and it is stored on heap.

But if we change the new string, its reference gets modified.

str2=str2.concat("world");



#### String Comparison

String comparison can be done in 3 ways.

1. Using **equals()** method
2. Using == operator
3. By **CompareTo()** method

1)Equal

It compares the content of the strings. It will return **true** if string matches, else returns **false**.

String s = "Hell";

String s1 = "Hello";

String s2 = "Hello";

s1.equals(s2); //true

s.equals(s1) ; //false

String s3=new String(“Hello”);

S3.equals(s1);//true

Reason: Equals method checks hashcode. Ideally for every object new hashcode is generated by JVM. You can check it by System.***out***.println(System.*identityHashCode*(str1));

But in String class hashcode method is overloaded . If String content is same it will return same hascode because equals method internally use hashcode.

System.***out***.println(str1.hashCode());

System.***out***.println(str3.hashCode()); It will give same hashcode

2)==

== operator compares two object references to check whether they refer to same instance. This also, will return **true** on successful match.

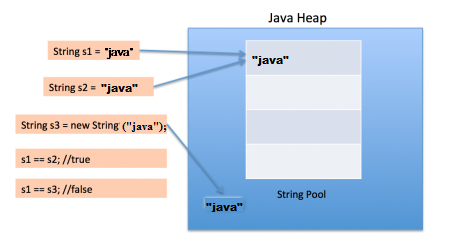
String s1 = "Java";

String s2 = "Java";

String s3 = new string ("Java");

test(s1 == s2) //true

test(s1 == s3) //false



#### 3) By compareTo() method

compareTo() method compares values and returns an int which tells if the string compared is less than, equal to or greater than the other string. It compares the String based on natural ordering i.e alphabetically. Its general syntax is,

String s1 = "Abhi";

String s2 = "Viraaj";

String s3 = "Abhi";

s1.compareTo(S2); //return -1 because s1 < s2

s1.compareTo(S3); //return 0 because s1 == s3

s2.compareTo(s1); //return 1 because s2 > s1

Function:

1)valueOf() function is used to convert **primitive data types** into Strings.

public class Example{

public static void main(String args[]){

int num = 35;

String s1 = String.valueOf(num); //converting int to String

System.out.println(s1+"IAmAString");

}

}

2)

#### trim()

This method returns a string from which any leading and trailing whitespaces has been removed.

String str = " hello ";

System.out.println(str.trim());

STRING BUFFER

 As we know that String objects are immutable, so if we do a lot of changes with **String** objects, we will end up with a lot of memory leak.

**StringBuffer** class is used when we have to make lot of modifications to our string.

* StringBuffer() creates an empty string buffer and reserves room for 16 characters.
* stringBuffer(int size) creates an empty string and takes an integer argument to set capacity of the buffer.
* class Test {
* public static void main(String args[])
* {
* String str = "study";
* str.concat("tonight");
* System.out.println(str); // Output: study
* StringBuffer strB = new StringBuffer("study");
* strB.append("tonight");
* System.out.println(strB); // Output: studytonight
* }
* }

Output is such because String objects are immutable objects. Hence, if we concatenate on the same String object, it won't be altered**(Output: study)**. But StringBuffer creates mutable objects. Hence, it can be altered**(Output: studytonight)**

### StringBuilder class

StringBuilder is identical to StringBuffer except for one important difference that it is not synchronized, which means it is not thread safe. Its because StringBuilder methods are not synchronised.

StringBuilder operates faster.

----------------------------------------------------------------------------------------------------------------------------------

// Main.java

public class Main

{

public static void main(String args[])

{

String s1 = "abc";

String s2 = s1;

s1 += "d";

System.out.println(s1 + " " + s2 + " " + (s1 == s2));

StringBuffer sb1 = new StringBuffer("abc");

StringBuffer sb2 = sb1;

sb1.append("d");

System.out.println(sb1 + " " + sb2 + " " + (sb1 == sb2));

}

} //end class

abcd abc false

abcd abcd true

stack Heap

S1

S2

In String class when literal is created it is created on String constant pool.

When you use + operator it internally call String buffer append method so new object is created on heap.

REGEX

/d= digit

/D=Nondigit

/w=word

/W=NonWord

/s=space

/S=non space

\*=0 or more

+=1 or more

?=0 or 1

String str="123o";

**if**(str.matches("[0-9]+"))

{

System.***out***.println("String contains only digits");

}

**else**{

System.***out***.println("string contains other character as well");

}

}