String Literal Vs String Object in Java

String str=”maruthi”;

This is string literal. When you declare string like this, you are actually calling intern() method on String. This method **references internal pool**of string objects. If there already exists a string value “maruthi”, then str will reference of that string and no new String object will be created.

String str1=new String(“hi hello”);

This is string object. In this method JVM is forced to create a new string reference, even if “GeeksForGeeks” is in the reference pool.

Therefore, if we compare performance of string literal and string object, string object will always take more time to execute than string literal because it will construct a new string every time it is executed.

Difference between string as an object and string as an literal.

**package** practice;

**import** java.util.Scanner;

**public** **class** Testscanner {

**public** **static** **void** main(String args[])

{

String s="manu";

String ss="manu";

String s1=**new** String("maruthi");

String s2=**new** String("maruti");

**if**(s==ss)

System.***out***.println("match");

System.***out***.println(s1.contentEquals(s2));

}

}

Output:

match

false

charset UTF-8,UTF-16?

UTF-8, UTF-16 and UTF-32 character encoding is how many bytes it require to represent a character in memory.

UTF-8 uses minimum one byte, while UTF-16 uses minimum 2 bytes. BTW, if character's code point is greater than 127, maximum value of byte then UTF-8 may take 2, 3 o 4 bytes but UTF-16 will only take either two or four bytes.

On the other hand, UTF-32 is fixed width encoding scheme and always uses 4 bytes to encode a Unicode code point.

There are two things, which are important to [convert bytes to characters](http://javarevisited.blogspot.sg/2014/08/2-examples-to-convert-byte-array-to-String-in-Java.html), a **character set** and an **encoding**. A character set is nothing but list of characters, where each symbol or character is mapped to a numeric value, also known as code points.

*UTF-16, UTF-32 and UTF-8 are encoding schemes*, which describe *how these values (code points) are mapped to bytes* (using different bit values as a basis; e.g. 16-bit for UTF-16, 32 bits for UTF-32 and 8-bit for UTF-8). UTF stands for Unicode Transformation, which defines an algorithm to map every Unicode code point to a unique byte sequence.  
  
 For example, for character A, which is Latin Capital A, Unicode code point is U+0041, UTF-8 encoded bytes are 41, UTF-16 encoding is 0041 and Java char literal is '\u0041'.

---replaceALL?-----

**package** practice;

**import** java.util.Scanner;

**public** **class** Testscanner {

**public** **static** **void** main(String args[])

{

String s1="hi..helloo.....this is java";

String repstr=s1.replaceAll("a","b");

String repcha=s1.replace('a','e');

String repstr1=s1.replaceAll("is","pp");

System.***out***.println(repstr);

System.***out***.println(repcha);

System.***out***.println(repstr1);

}

}

Output:

hi..helloo.....this is jbvb

hi..helloo.....this is jeve

hi..helloo.....thpp pp java

----------Intern()  
The **java string intern()** method returns the interned string. It can be used to return string from memory, if it is created by new keyword. It creates exact copy of heap string object in string constant pool.

Example:

**package** practice;

**import** java.util.Scanner;

**public** **class** Testscanner {

**public** **static** **void** main(String args[])

{

/\*String s1="hi..helloo.....this is java";

String repstr=s1.replaceAll("a","b");

String repcha=s1.replace('a','e');

String repstr1=s1.replaceAll("is","pp");

System.out.println(repstr);

System.out.println(repcha);

System.out.println(repstr1);\*/

String s1="hi";

String s2=**new** String("hello");

String s3=s1.intern();

System.***out***.println(s1==s2);

System.***out***.println(s1==s3);

System.***out***.println(s2==s3);

}

}

Output:

False—reference variables pointing to different instances

True—references variables pointing to same instances

False

-------Purpose of Match() in java

There are **three** variants of matches() method.

1. **String matches() :**This method tells whether or not this string matches the given regular expression. An invocation of this method of the form**str.matches(regex)** yields exactly the same result as the expression **Pattern.matches(regex, str).**
2. **String regionMatches() (with ignoreCase) :**This method has two variants which can be used to test if two string regions are equal.
3. **3. String regionMatches() :**This method has two variants which can be used to test if two string regions are equal.