**10 Things Every Java Programmer Should Know about String**

String in Java is very special class and most frequently used class as well. There are lot many things to learn about String in Java than any other class, and having a good knowledge of different String functionalities makes you to use it properly. Given heavy use of Java String in almost any kind of project, it become even more important to know subtle detail about String. Though I have shared lot of String related article already here in **Javarevisited**, this is an effort to bring some of String feature together. In this tutorial we will see some important points about Java String, which is worth remembering. You can also refer my earlier post [10 advanced Java String questions](http://javarevisited.blogspot.com/2012/10/10-java-string-interview-question-answers-top.html) to know more about String.   
  
Though I tried to cover lot of things, there are definitely few things, which I might have missed; please let me know if you have any question or doubt on java.lang.String functionality and I will try to address them here.

**1) Strings are not null terminated in Java.**

Unlike C and C++, String in Java doesn't terminate with null character. Instead String are Object in Java and backed by character array. You can get the character array used to represent String in Java by calling toCharArray() method of java.lang.String class of JDK.

**2) Strings are immutable and final in Java**

Strings are immutable in Java it means once created you cannot modify content of String. If you modify it by using toLowerCase(), toUpperCase() or any other method, It always result in new String. Since String is final there is no way anyone can extend String or override any of String functionality. Now if you are puzzled [why String is immutable or final in Java](http://javarevisited.blogspot.com/2010/10/why-string-is-immutable-in-java.html). checkout the link.

**3) Strings are maintained in String Pool**

As I Said earlier String is special class in Java and all String literal e.g. "abc" (anything which is inside double quotes are String literal in Java) are maintained in a separate String pool, special memory location inside Java memory, more precisely inside [PermGen Space](http://javarevisited.blogspot.com/2012/01/tomcat-javalangoutofmemoryerror-permgen.html). Any time you create a new String object using String literal, JVM first checks String pool and if an object with similar content available, than it returns that and doesn't create a new object. JVM doesn't perform String pool check if you create object using new operator.

You may face subtle issues if you are not aware of this String behaviour , here is an example

String name = "Scala"; //1st String object

String name\_1 = "Scala"; //same object referenced by name variable

String name\_2 = **new** String("Scala") //different String object

//this will return true

**if**(name==name\_1){

System.out.println("both name and name\_1 is pointing to same string object");

}

//this will return false

**if**(name==name\_2){

System.out.println("both name and name\_2 is pointing to same string object");

}

if you compare name and name\_1 using equality operator "==" it will return true because both are pointing to same object. While name==name\_2 will return false because they are pointing to different string object. It's worth remembering that [equality "==" operator compares object memory location](http://javarevisited.blogspot.sg/2012/12/difference-between-equals-method-and-equality-operator-java.html) and not characters of String. By default Java puts all string literal into string pool, but you can also put any string into pool by calling intern() method of java.lang.String class, like string created using new() operator.

**4) Use Equals methods for comparing String in Java**

String class overrides equals method and provides a content equality, which is based on characters, case and order. So if you want to compare two String object, to check whether they are same or not, always use equals() method instead of equality operator. Like in earlier example if we use [equals method](http://javarevisited.blogspot.com/2011/02/how-to-write-equals-method-in-java.html) to compare objects, they will be equal to each other because they all contains same contents. Here is example of comparing String using equals method.

String name = "Java"; //1st String object

String name\_1 = "Java"; //same object referenced by name variable

String name\_2 = **new** String("Java") //different String object

**if**(name.equals(name\_1)){

System.out.println("name and name\_1 are equal String by equals method");

}

//this will return false

**if**(name==name\_2){

System.out.println("name\_1 and name\_2 are equal String by equals method");

}

You can also check my earlier post [difference between equals() method and == operator](http://javarevisited.blogspot.com/2012/12/difference-between-equals-method-and-equality-operator-java.html) for more detail discussion on consequences of comparing two string using == operator in Java.

**5) Use indexOf() and lastIndexOf() or matches(String regex) method to search inside String**

String class in Java provides convenient method to see if a character or sub-string or a pattern exists in current String object. You can use indexOf() which will return position of character or String, if that exist in current String object or -1 if character doesn't exists in String. lastIndexOf is similar but it searches from end. String.match(String regex) is even more powerful, which allows you to search for a [regular expression pattern](http://javarevisited.blogspot.com/2012/10/regular-expression-example-in-java-to-check-String-number.html) inside String. here is examples of indexOf, lastIndexOf and matches method from java.lang.String class.

String str = "Java is best programming language";

**if**(str.indexOf("Java") != -**1**){

     System.out.println("String contains Java at index :" + str.indexOf("Java"));

}

**if**(str.matches("J.\*")){

     System.out.println("String Starts with J");

}

str ="Do you like Java ME or Java EE";

**if**(str.lastIndexOf("Java") != -**1**){

      System.out.println("String contains Java lastly at: " + str.lastIndexOf("Java"));

}

As expected indexOf will return 0 because characters in String are indexed from zero. lastIndexOf returns index of second “Java”, which starts at 23 and matches will return true because J.\* pattern is any String starting with character J followed by any character because of dot(.) and any number of time due to asterick (\*).

Remember matches() is tricky and some time non-intuitive. If you just put "Java" in matches it will return false because String is not equals to "Java" i.e. in case of plain text it behaves like equals method. See [here](http://java67.blogspot.sg/2012/09/java-string-matches-example-regular-expression.html) for more examples of String matches() method.

Apart from indexOf(), lastIndexOf() and matches(String regex) String also has methods like startsWith() and endsWidth(), which can be used to check an String if it starting or ending with certain character or String.

**6) Use SubString to get part of String in Java**

Java String provides another useful method called substring(), which can be used to get parts of String. basically you specify start and end index and substring() method returns character from that range. Index starts from 0 and goes till String.length()-1. By the way String.length() returns you number of characters in String, including white spaces like tab, space. One point which is worth remembering here is that substring is also backed up by character array, which is used by original String. This can be dangerous if original string object is very large and substring is very small, because even a small fraction can hold reference of complete array and prevents it from being garbage collected even if there is no other reference for that particular String. Read [How Substring works in Java](http://javarevisited.blogspot.com/2011/10/how-substring-in-java-works.html) for more details. Here is an example of using SubString in Java:

String str = "Java is best programming language";

//this will return part of String str from index 0 to 12

String subString = str.substring(**0**,**12**);

System.out.println("Substring: " + subString);

**7) "+" is overloaded for String concatenation**

*Java doesn't support Operator overloading* but String is special and + operator can be used to concatenate two Strings. It can even used to convert int, char, long or double to convert into String by simply concatenating with empty string "". internally + is implemented using StringBuffer prior to Java 5 and StringBuilder from Java 5 onwards. This also brings point of using StringBuffer or StringBuilder for manipulating String. Since both represent mutable object they can be used to reduce string garbage created because of temporary String. Read more about [StringBuffer vs StringBuilder](http://javarevisited.blogspot.com/2011/07/string-vs-stringbuffer-vs-stringbuilder.html) here.

**8) Use trim() to remove white spaces from String**

String in Java provides trim() method to remove white space from both end of String. If trim() removes white spaces it returns a new String otherwise it returns same String. Along with trim() String also provides replace() and replaceAll() method for replacing characters from String. replaceAll method even support regular expression. Read more about How to replace String in Java [here](http://javarevisited.blogspot.com/2011/12/java-string-replace-example-tutorial.html).

**9) Use split() for splitting String using Regular expression**

String in Java is feature rich. it has methods like split(regex) which can take any String in form of regular expression and split the String based on that. particularly useful if you dealing with comma separated file (CSV) and wanted to have individual part in a String array. There are other methods also available related to splitting String, see this [Java tutorial to split string](http://javarevisited.blogspot.com/2011/09/string-split-example-in-java-tutorial.html) for more details.

**10) Don't store sensitive data in String**

String pose security threat if used for storing sensitive data like passwords, SSN or any other sensitive information. Since String is immutable in Java there is no way you can erase contents of String and since they are kept in String pool (in case of String literal) they stay longer on Java heap ,which exposes risk of being seen by anyone who has access to Java memory, like reading from memory dump. Instead char[] should be used to store password or sensitive information. See [Why char[] is more secure than String for storing passwords in Java](http://javarevisited.blogspot.com.br/2012/03/why-character-array-is-better-than.html) for more details.  
  
  
**11) Character Encoding and String**  
Apart from all these 10 facts about String in Java, the most critical thing to know is *what encoding your String is using*. It does not make sense to have a String without knowing what encoding it uses. There is no way to interpret an String if you don't know the encoding it used. You can not assume that "plain" text is ASCII. If you have a String, in memory or stored in file, you must know what encoding it is in, or you cannot display it correctly. By default Java uses platform encoding i.e. character encoding of your server, and believe me this can cause huge trouble if you are handling Unicode data, especially if you are [converting byte array to XML String](http://javarevisited.blogspot.sg/2013/03/convert-and-print-byte-array-to-hex-string-java-example-tutorial.html). I have faced instances where our program fail to interpret Strings from European language e.g. German, French etc. because our server was not using Unicode encodings like UTF-8 or UTF-16. Thankfully, Java allows you to specify default character encoding for your application using system property file.encoding. See [here](http://javarevisited.blogspot.com/2012/01/get-set-default-character-encoding.html) to read more about character encoding in Java

That's all about String in Java. As I have said String is very special in Java, sometime even refer has God class. It has some unique feature like immutability, concatenation support, caching etc, and to become a serious Java programmer, detailed knowledge of String is quite important. Last but not the least don't forget about [character encoding](http://javarevisited.blogspot.com/2012/01/get-set-default-character-encoding.html) while converting a byte array into String in Java. Good knowledge of java.lang.String is must for good Java developers.

**References**

* http://javarevisited.blogspot.lt/2013/07/java-string-tutorial-and-examples-beginners-programming.html