**java.lang.String**

* String is an object, which is backend by char array.
* String is immutable and final.

Immutable means once it is instantiated its state cannot be changed. When it comes to String, the content of that String cannot be changed.

And final means you cannot extend that class. Means, you cannot extend String class.

**String Pool:**

In Java Strings are stored in a very separate area called String Pool in the PermGen i.e., “method area”. So, that means each String you creates is cached in String Pool.

Let’s take an example:

String s1 = “Hello”; //Cached in the String Pool

String s2 = “Hello”; // now s2 will also point to the same String Object in the String pool.

When s2 = “Hello” is assigned, JVM will check in String Pool whether any String is having “Hello”. If it is available, then JVM will return the reference of s1 to s2.

**String s3 = new String(“Hello”); //Which creates a new memory space. i.e., it is not interned.**

To refer to same memory which s1 points, use intern() method.

**Ex: String s3 = new String(“Hello”).intern(); //Now it is interned**

This whole process is called interning.

From Java 1.7, String is stored in the **main Heap Area.**

**What is the need of creating a String Pool?**

String being the most widely used Objects or Class; it makes sense to cache them, so as to optimize the memory.

Ex: Suppose we have batch file processing application, which fairly runs for long time. Suppose the input for this batch is a file with 100 thousand records. 50% of 100 thousand records contain common name “Kumar”. We have 50 thousand Strings “Kumar” to deal with. Suppose application goes and creates new Object for “Kumar”, we would end up with 50 thousand objects in memory.

To avoid this, the moment they encounter String literal its cache in the pool. Next time when String literal comes, the reference of cached instance is used.

That’s how instance pooling or interning helps in optimizing memory.

**Why String has to be immutable?**

1. Suppose we have a String “s1” pointing to “Hello” from previous example.

Now we want to modify “s1” as below:

**s1.toUpperCase();**

If String is mutable, we are changing the value if String s1.

With changing of “s1”, the other Strings, which are referring to “s1”, will also get impacted. Which causes to different issues.

This is main reason for making a String class immutable.

1. It is thread safe by making it immutable.
2. String caches their hash code internally. The moment you create a String with certain literal, its hash code is computed based on its characters and cached forever. No need to compute the hash code while fetching the values always.

Count the Occurrences of a substring program

**Memory Leak up until JDK 7.0:**

String s1 = “Hello Sravan Hello Hello”;

Strings2 = s1.subString(0,4);

**//java 1.7 substring source of memory leak**

Because, when you substring the original String s1, java is used to create a new String, but that new String would eventually contain the same character array that actually s1 have. Which is causing the memory leak.

**StringBuffer/StringBuilder and String:**

StringBuffer and StringBuilder are mutable data structures, which are very similar to String.

When String has immutable for various reasons, they have also provided classes that are mutable which represents Strings.

**StringBuffer is some what legacy and is synchronized.** Every method in StringBuffer has a synchronized keyword on public API.

**StringBuilder is not synchronized.**

**+ operator vs StringBuilder.append() and concat:**

+operator:

Example: s1 = “This” +” is” +” Sravan Kumar”.

When using ‘+’ previously, a new String is created every time and memory is not optimized properly.

With latest versions in Java when doing ‘+’, internally a new StringBuilder is created and appending to the StringBuilder and finally returns .**toString**() value of StringBuilder.

**Concat:** we can concat only Strings. This is the major difference between StringBuilder.append(), + operator of String.

* Concat is efficient when dealing with small set of Strings.
* When you are dealing with large amount of Strings, always good to deal with StringBuilder APIs.

**Storing Passwords:**

It’s always better to store the password in char []. Reason is Strings are created in String Pool and they are maintained for certain period, they are cache and cached for certain period and they are not Garbage Collected. By accident your Password is clear text and stores that in String variable and put that in String Pool and if somebody gets dump of that memory, he can clearly read the password that was in Plain text, which is definitely a security check.

Instead if you use char [], and store the password in char [], as soon as you are done dealing with the password set the char [] back to null or set char [] to ‘0’, so that char [] value gets Garbage Collected.

**Reverse a String Example:**

There are 3 ways:

1. Using customized logic
2. Using Standard Java API
3. Using Recursion

**Difference between String, StringBuffer and StringBuilder:**

|  |  |  |
| --- | --- | --- |
| String | StringBuffer | StringBuilder |
| Immutable Object | Mutable Object | Mutable Object |
| Stored in Method Area  Constant String Pool | Stored in Heap memory | Stored in Heap memory |
| Not Legacy | Legacy | Not Legacy |
| Synchronized | Synchronized | Not Synchronized |
| Performance: Fast | Very Slow | Fast |

**StringTokenizer:**

It allows String into tokens.

Example:

|  |
| --- |
| **public** **class** StringTokenizerExample {    **public** **static** **void** main(String[] args) {    String str = "Hello Sravan!!! Hello";  StringTokenizer strToken = **new** StringTokenizer(str,"!");    System.***out***.println("Count: "+strToken.countTokens());  **while**(strToken.hasMoreElements()){  System.***out***.println("String Tokenizer: "+strToken.nextToken());  } |

**Java String Split:**

This method splits this string against given regular expression and returns a char array.

Signature:

There are two signature for split() method in java string:

Public String split(String regex);

Public String split(String regex, int limit);

|  |
| --- |
| String strSp = "Hello Sravan Hello";  String[] strSplit = strSp.split("\\|");  **for** (String string : strSp.split("\\s",0)) {  System.***out***.println(string);  } |

**String methods:**

* toUpperCase()
* toLowerCase()
* trim()
* startsWith()
* endsWith()
* charAt()
* length()
* intern()
* valueOf()
* replace()
* indexOf()