Strings in Java

Difficulty Level : Easy Last Updated : 13 Jul, 2021

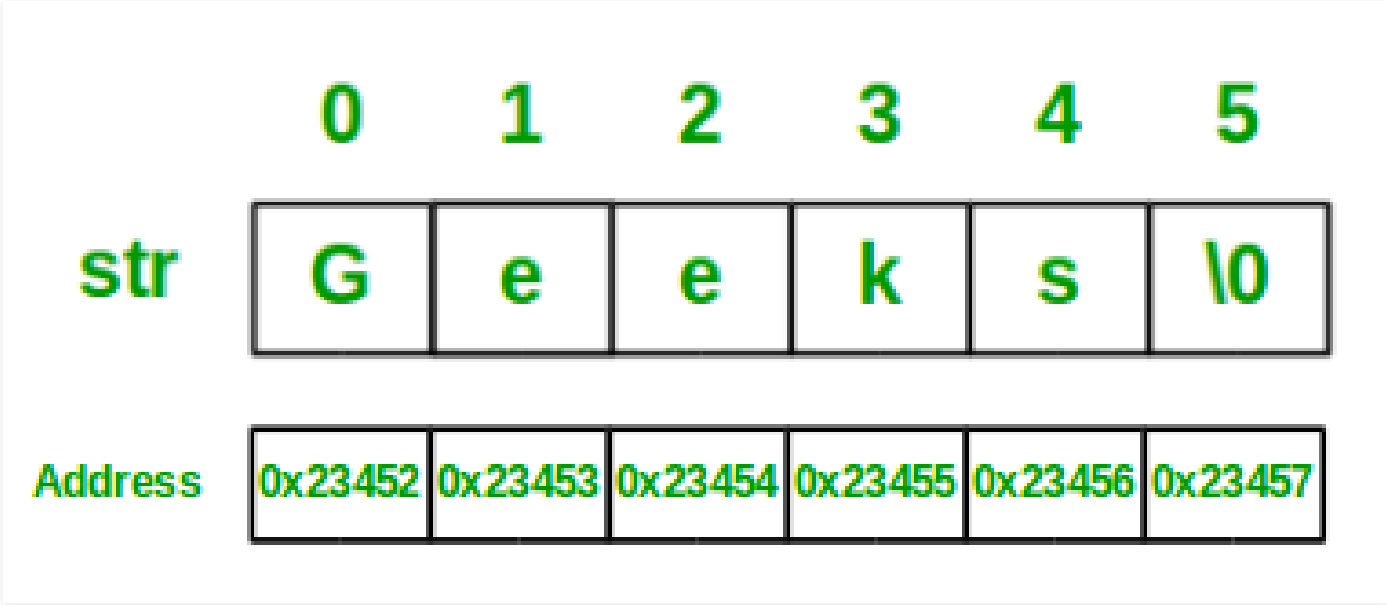
Strings in Java are Objects that are backed internally by a char array. Since arrays are immutable(cannot grow), Strings are immutable as well. Whenever a change to a String is made, an entirely new String is created.

Syntax:

```
<String_Type> <string_variable> = "<sequence_of_string>";
```

Example:

```
String str = "Geeks";
```



Memory allotment of String

Whenever a String Object is created as a literal, the object will be created in String constant pool. This allows JVM to optimize the initialization of String literal.

For example:

```
String str = "Geeks";
```

The string can also be declared using **new** operator i.e. dynamically allocated. In case of String are dynamically allocated they are assigned a new memory location in heap. This

string will not be added to String constant pool.

For example:

```
String str = new String("Geeks");
```

If you want to store this string in the constant pool then you will need to “intern” it.

For example:

```
String internedString = str.intern();
// this will add the string to string constant pool.
```

It is preferred to use String literals as it allows JVM to optimize memory allocation.

An example that shows how to declare String

```
// Java code to illustrate String
import java.io.*;
import java.lang.*;

class Test {
    public static void main(String[] args)
    {
        // Declare String without using new operator
        String s = "GeeksforGeeks";

        // Prints the String.
        System.out.println("String s = " + s);

        // Declare String using new operator
        String s1 = new String("GeeksforGeeks");

        // Prints the String.
        System.out.println("String s1 = " + s1);
    }
}
```

Output:

```
String s = GeeksforGeeks
String s1 = GeeksforGeeks
```

Interfaces and Classes in Strings in Java

- CharBuffer: This class implements the CharSequence interface. This class is used to allow character buffers to be used in place of CharSequences. An example of such usage is the regular-expression package java.util.regex.

- String: String is a sequence of characters. In java, objects of String are immutable which means a constant and cannot be changed once created.

Creating a String

- There are two ways to create a string in Java:
  - ***String literal***

```
String s = “GeeksforGeeks”;
```

- **Using *new* keyword**

```
String s = new String (“GeeksforGeeks”);
```

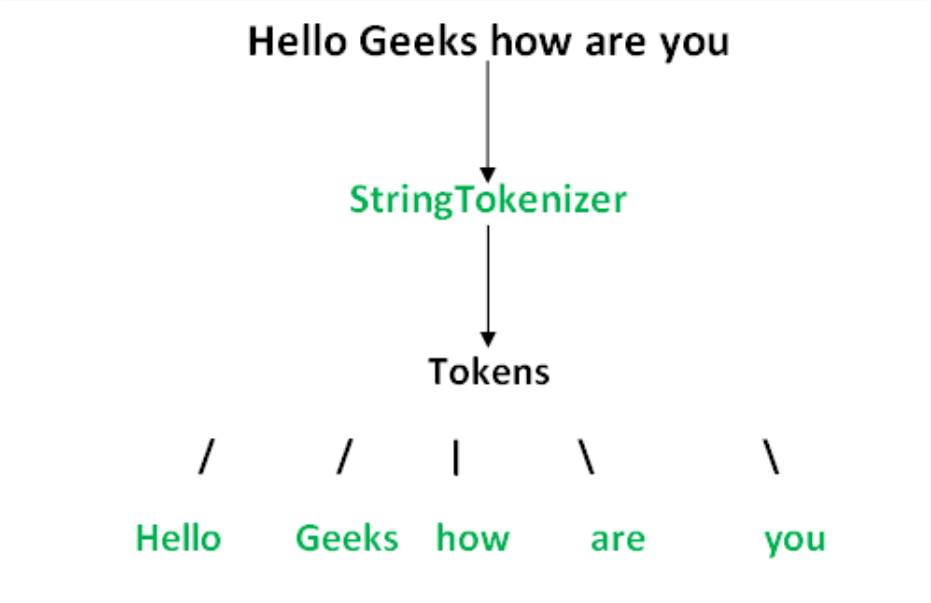
- StringBuffer:
  StringBuffer is a peer class of **String** that provides much of the functionality of strings. The string represents fixed-length, immutable character sequences while StringBuffer represents growable and writable character sequences.
  Syntax:

```
StringBuffer s = new StringBuffer("GeeksforGeeks");
```

- StringBuilder:
  The **StringBuilder** in Java represents a mutable sequence of characters. Since the String Class in Java creates an immutable sequence of characters, the StringBuilder class provides an alternate to String Class, as it creates a mutable sequence of characters.
  Syntax:

```
StringBuilder str = new StringBuilder();
str.append("GFG");
```

- StringTokenizer:
  StringTokenizer class in Java is used to break a string into tokens.
  Example:



- A StringTokenizer object internally maintains a current position within the string to be tokenized. Some operations advance this current position past the characters processed. A token is returned by taking a substring of the string that was used to create the StringTokenizer object.

- StringJoiner:
  StringJoiner is a class in *java.util* package which is used to construct a sequence of characters(strings) separated by a delimiter and optionally starting with a supplied prefix and ending with a supplied suffix. Though this can also be with the help of StringBuilder class to append delimiter after each string, StringJoiner provides an easy way to do that without much code to write.
  Syntax:

  public StringJoiner(CharSequence delimiter)

Above we saw we can create string by String Literal.

For ex- // String s="Welcome";

Here the JVM checks the String Constant Pool. If the string does not exist, then a new string instance is created and placed in a pool. If the string exists, then it will not create a new object. Rather, it will return the reference to the same instance. The cache which stores these string instances is known as the String Constant pool or String Pool. In earlier versions of Java up to JDK 6 String pool was located inside PermGen(Permanent Generation) space. But in JDK 7 it is moved to the main heap area.

Why did the String pool move from PermGen to the **normal heap area?**

PermGen space is limited, the default size is just 64 MB. it was a problem with creating and storing too many string objects in PermGen space. That's why the String pool was moved to a larger heap area. To make Java more memory efficient, the concept of string literal is used. By the use of the 'new' keyword, The JVM will create a new string object in the normal heap area even if the same string object is present in the string pool.

For ex-

String a=new String("Bhubaneswar")

Let's have a look at the concept with a java program and visualize the actual JVM memory structure:

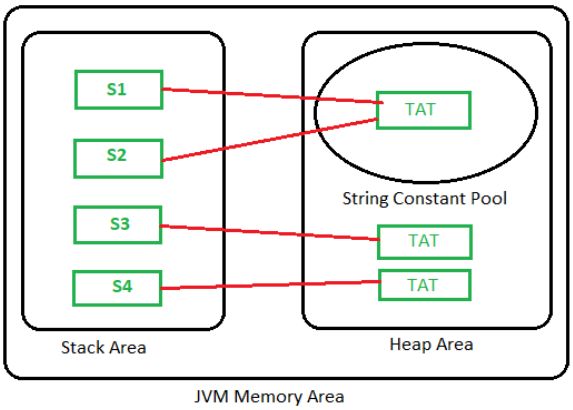
Program:

```
class StringStorage {
    public static void main(String args[])
    {
        String s1 = "TAT";
        String s2 = "TAT";
        String s3 = new String("TAT");
        String s4 = new String("TAT");
        System.out.println(s1);
        System.out.println(s2);
        System.out.println(s3);
```

```
        System.out.println(s4);
    }
}
```

Output

```
TAT
TAT
TAT
TAT
```



Note: All objects in Java are stored in a heap. The reference variable is to the object stored in the stack area or they can be contained in other objects which puts them in the heap area also.