String

Strings in Java

String is a sequence of characters, digits and symbols. Strings are defined as an array of characters. The difference between a character array and a string is the string is terminated with a special character '\0'.

In java String is a class available in *java.lang* package. String variables declared as

object of String class.

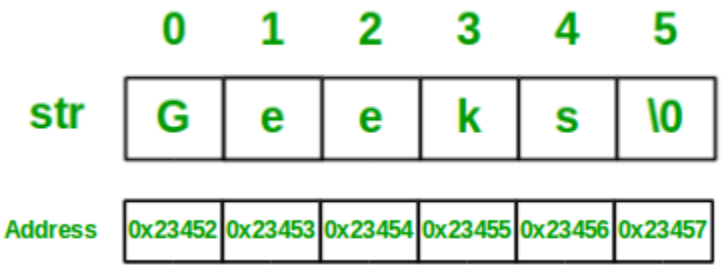
The basic syntax for declaring a string in **Java programming** language.

Syntax:

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

Example:

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

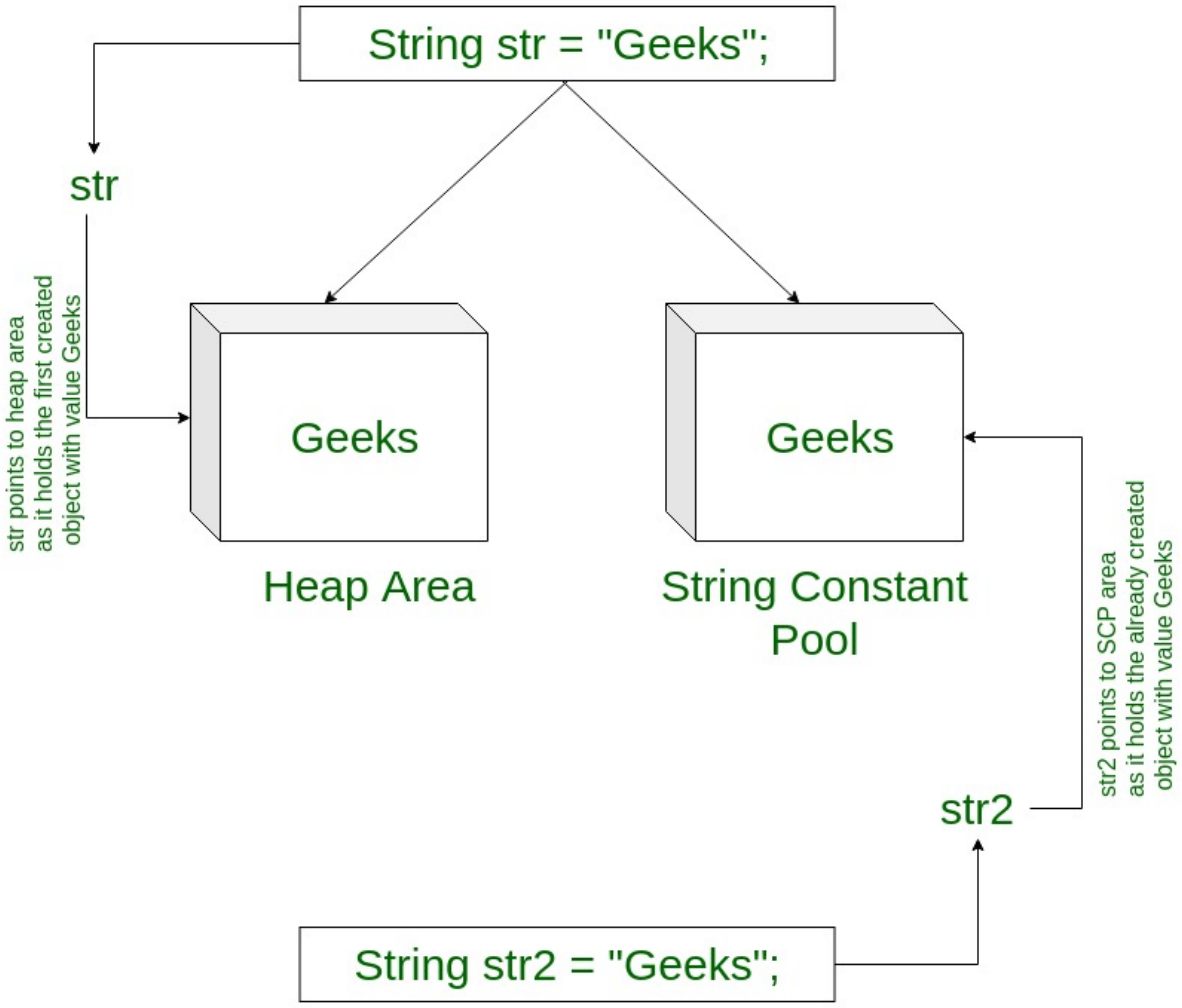


Memory allotment of String

Whenever a String Object is created, two objects will be created- one in the Heap Area and one in the String constant pool and the String object reference always points to heap area object.

For example:

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



How to create a string object?

There are two ways to create String object:

1. By string literal
2. By new keyword

1) String Literal

Java String literal is created by using double quotes. **For Example:**

1. String s="welcome";

Each time you create a string literal, the JVM checks the "string constant pool" first. If the string already exists in the pool, a reference to the pooled instance is returned. If the string doesn't exist in the pool, a new string instance is created and placed in the pool. **For example:**

1. String s1="Welcome";
2. String s2="Welcome"; *//It doesn't create a new instance*

In the above example, only one object will be created. Firstly, JVM will not find any string object with the value "Welcome" in string constant pool, that is why it will create a new object. After that it will find the string with the value "Welcome" in the pool, it will not create a new object but will return the reference to the same instance.

> ***Note: String objects are stored in a special memory area known as the "string constant pool".***

Why Java uses the concept of String literal? To make Java more memory efficient (because no new objects are created if it exists already in the string constant pool).

2) By new keyword

1. String s=**new** String("Welcome");//creates two objects and one reference variable

In such case, JVM will create a new string object in normal (non-pool) heap memory, and the literal "Welcome" will be placed in the string constant pool. The variable s will refer to the object in a heap (non-pool).

Java String Example 1

```
public class StringExample1
{
public static void main(String args[])  //static void main()
{
String s1="java";                                   //creating string by java string literal
char ch[]={'s','t','r','i','n','g','s'};
String s2=new String(ch);                  //converting char array to string
String s3=new String("example");       //creating java string by new keyword
System.out.println(s1);
System.out.println(s2);
System.out.println(s3);
}
}
```

Java String Example 2

```
import  java.util.*;
public class StringExample2
{
public static void main(String args[]) //static void main()
{
Scanner sn=new Scanner(System.in);
System.out.print(“Enter value = ”);
String s1=sn.next();    // receive single word or letter without whitespace
System.out.print(“Enter value = ”);
String s2=sn.nextLine();         //receive one or more words with or without whitespace
System.out.println(s1);
System.out.println(s2);
}
}
```

Java String Example 3

```
import  java.io.*;

public class StringExample3
{
public static void main(String args[]) throws IOException     //static void main()
{
InputStreamReader isr=new InputStreamReader (System.in);
BufferedReader br=new BufferedReader(isr);
System.out.print("Enter value = ");
String s1=br.readLine();          //receive one or more words with or without whitespace
System.out.print("Enter value = ");
String s2= br.readLine();         //receive one or more words with or without whitespace
System.out.println(s1);
System.out.println(s2);
}
}
```

Java String class methods

The java.lang.String class provides many useful methods to perform operations on sequence of

char values (String).

No.	Method	Description
1	char charAt(int index)	*returns char value for the particular index*
	Example: String str=”Welcome”; char x=str.charAt(1); **Value of x is ‘e’**	
2	int length()	*returns string length*
	Example: String str=”Welcome”; int x= str.length(); **Value of x is 7**	
3	static String format(String format, Object... args)	*returns a formatted string.*
4	static String format(Locale l, String format, Object... args)	*returns formatted string with given locale.*
5	String substring(int beginIndex)	*returns substring for given begin index.*
	Example: String str=”Welcome to School”; String s1=str.substring(3); **Value of s1 is “come to School”**	
6	String substring(int beginIndex, int endIndex)	*returns substring of calling string for given begin index and end index.*
	Example: String str=”Welcome to School”; String s1=str.substring(3,12);	

	Value of s1 is “come to S”	
7	boolean contains(CharSequence s)	*returns true or false after matching the sequence of char value.*
8	static String join(CharSequence delimiter, CharSequence... elements)	*returns a joined string.*
9	static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements)	*returns a joined string.*
10	boolean equals(String another)	*checks the equality of calling string with the specified string.*
	Example: String str1=”Welcome”; String str2=”Welcome”; boolean x=str1.equals(str2); **Value of x is true** String str1=”Welcome”; String str2=”WELCome”; boolean x=str1.equals(str2); **Value of x is false** String str1=”Welcome”; String str2=”Hello”; boolean x=str1.equals(str2); **Value of x is false**	
11	boolean isEmpty()	*checks if string is empty.*

12	String concat(String str)	*concatenates the calling and specified string.*
	Example: String str1=”Welcome”; String str2=”Hello”; String s1 =str1.concat(str2); **Value of s1 is “WelcomeHello”**	
13	String replace(char old, char new)	*replaces all occurrences of the specified char value.*
	Example: String str1=”Welcome to School”; String s1 x=str1.replace(‘o’,‘k’); **Value of s1 is “Welckme tk Schkkl”**	
14	String replace(String old, String new)	*replaces all occurrences of the specified String.*
	Example: String str1=”Right is Right but not always Right”; String s1 x=str1.replace(“Right”, “Wrong”); **Value of s1 is “Wrong is Wrong but not always Wrong”**	
15	String equalsIgnoreCase(String another)	*checks the equality of calling string with the specified string.. It doesn't check case.*
	Example: String str1=”Welcome”; String str2=”Welcome”; boolean x=str1.equalsIgnoreCase(str2);	

	Value of x is true String str1=”Welcome”; String str2=”WELCome”; boolean x=str1.equalsIgnoreCase(str2); **Value of x is true** String str1=”Welcome”; String str2=”Hello”; boolean x=str1.equalsIgnoreCase(str2); **Value of x is false**	
16	String[] split(String regex)	*returns a split string matching regex.*
17	String[] split(String regex, int limit)	*returns a split string matching regex and limit.*
18	String intern()	*returns an interned string.*
19	int indexOf(char ch)	*returns the specified char value index. If not found then return -1.*
20	int indexOf(char ch, int fromIndex)	*returns the specified char value index starting with given index.* *If not found then return -1.*
21	int indexOf(String substring)	*returns the specified substring index. If not found then return -1.*
22	int indexOf(String substring, int fromIndex)	*returns the specified substring index starting with given index.* *If not found then return -1.*

	Example: String str=”Right is Right but not always Right”; int x= str.indexOf(‘i’); **Value of x is 1** String str=”Right is Right but not always Right”; int x= str.indexOf(‘i’,2); **Value of x is 6** String str=”Right is Right but not always Right”; int x= str.indexOf(“gh”); **Value of x is 2** String str=”Right is Right but not always Right”; int x= str.indexOf(“gh”,4); **Value of x is 11**	
23	String toLowerCase()	*returns calling string in lowercase.*
	Example: String str=”WelComE”; String x= str.toLowerCase(); **Value of x is “welcome”**	
24	String toLowerCase(Locale l)	*returns a string in lowercase using specified locale.*
25	String toUpperCase()	*returns calling string in uppercase.*

	Example: String str=”WelComE”; String x= str.toUpperCase(); **Value of x is “WELCOME”**	
26	String toUpperCase(Locale l)	*returns a string in uppercase using specified locale.*
27	String trim()	*removes beginning and ending spaces of this string.*
	Example: String str=” Welcome to School ”; String x= str.trim(); **Value of x is “Welcome to School”**	
28	static String valueOf(int value)	*converts given type into string. It is an overloaded method.*
	Example: int x=25; double y=35.67; String str1=String.valueOf(x) ; String str2=String.valueOf(y) ; **Value of str1 is “25”** **Value of str2 is “35.67”**	
29	int lastIndexOf(char ch)	*returns the specified char value last index. If not found then return -1.*
30	int lastIndexOf(String substring)	*returns the specified substring last index. If not found then return -1.*

31	int lastIndexOf(char ch,int index)	*returns the specified char value last index up to given index. If not found then return -1.*
32	int lastIndexOf(String substring,int index)	*returns the specified substring last index up to given index. If not found then return -1.*
	Example: String str="Right is Right but not always Right"; int x= str.lastIndexOf('R'); **Value of x is 30** String str="Right is Right but not always Right"; int x= str.lastIndexOf("gh"); **Value of x is 32** String str="Right is Right but not always Right"; int x= str.lastIndexOf('R',15); **Value of x is 9** String str="Right is Right but not always Right"; int x= str.lastIndexOf("gh",15); **Value of x is 11**	
33	int compareTo(String)	*Compares calling and specified Strings are equal or greater or smaller. Equals then return zero(0), greater then return positive value otherwise negative value.*

	Example: String str1=”Welcome”; String str2=”Welcome”; int x=str1.compareTo(str2); **Value of x is 0 (zero)** String str1=”Welcome”; String str2=”WELCome”; int x=str1.compareTo(str2); **Value of x is 32 (positive value)** String str1=”Welcome”; String str2=”Zyx”; int x=str1.compareTo(str2); **Value of x is -3 (negative value)**	
34	int compareToIgnoreCase(String)	*Compares calling and specified Strings are equal or greater or smaller. Equals then return zero(0), greater then return positive value otherwise negative value. Ignore the cases of String(Lowercase or Uppercase)*
	Example: String str1=”Welcome”; String str2=”Welcome”; int x=str1.compareTo(str2); **Value of x is 0 (zero)**	

	String str1=”Welcome”; String str2=”WELCome”; int x=str1.compareTo(str2); **Value of x is 0 (zero)** String str1=”Welcome”; String str2=”Zyx”; int x=str1.compareTo(str2); **Value of x is -3 (negative value)** String str1=”Welcome”; String str2=”Hello”; int x=str1.compareTo(str2); **Value of x is 15 (positive value)**	
35	boolean endsWith(String)	*Checks the calling string has specified String as suffix or not.*
36	boolean startsWith(String)	*Checks the calling string has specified String as prefix or not.*
	Example: String str1=”Welcome to School”; boolean x=str1.endsWith(“ol”); **Value of x is true** String str1=”Welcome to School”; boolean x=str1.endsWith(“Sch”); **Value of x is false**	

	String str1=”Welcome to School”; boolean x=str1.startsWith(“We”); **Value of x is true** String str1=”Welcome to School”; boolean x=str1.startsWith(“come”); **Value of x is false**	
37	char [] toCharArray()	*returns an array of chars after converting a string into sequence of characters.*
	Example: String str1=”Welcome”; char x[]=str1.toCharArray(); **Value of x is {‘W’,‘e’,‘l’,‘c’,‘o’,‘m’,‘e’}**	
38	void getChars(int startindex,int endindex, char [], int arrayindex)	*Copies characters of calling into the specified char array from specified index*
	Example: String str=”Welcome”; char x[]=new char [10]; str.getChars(2,5,x,3); **Value of x is {‘’,‘’,‘’,‘l’,‘c’,‘o’,‘m’,‘’,‘’ ,‘’}**	

StringBuffer

StringBuufer class is mutable(changeable) class unlike the String class which is immutable. Both the capacity and character string of a StringBuffer can be changed dynamically.

A *StringIndexOutOfBoundException* is thrown if an index is not valid when using wrong index in StringBuffer manipulations.

StringBuffer Methods:

No	Method	Description
1	capacity()	*Returns the capacity of StringBuffer*
2	length()	*Returns the length(character count) of StringBuffer*
3	charAt(int index)	*The specified character of the sequence currently represented by the StringBuffer, as indicated by the index argument, is returned.*
4	setCharAt(int index, char ch)	*The character at the specified index of this StringBuffer is set to ch.*
5	toString()	*Converts to a string representing the Data in this StringBuffer.*
6	insert(int offset, char c)	*Insert the string representation of the char argument into this StringBuffer.*
7	delete(int start, int end)	*Removes the characters in a substring of this StringBuffer.*
8	replace(int start, int end, String str)	*Replaces the characters in a substring of this StringBuffer with characters in the specified String.*

9	reverse()	*The characters sequence contained in this StringBuffer is replaced by the reverse of the sequence.*
10	append(String str)	*Append the string to this StringBuffer.*
11	setLength(int newLength)	*Sets the length of this StringBuffer.*