String

* String is a built-in class available in java that represents a sequence of characters.

* String is predefine class which is present in "java.lang" package.

* In java, all classes are also considered as data types. So we can take string as a data type also.

* A class is also called as user-defined data types.

* In java string is an -
  - object
  - data type
  - class
  - group of characters.

* Creating Strings:

There are two ways to create strings in java -

i) Using new keyword

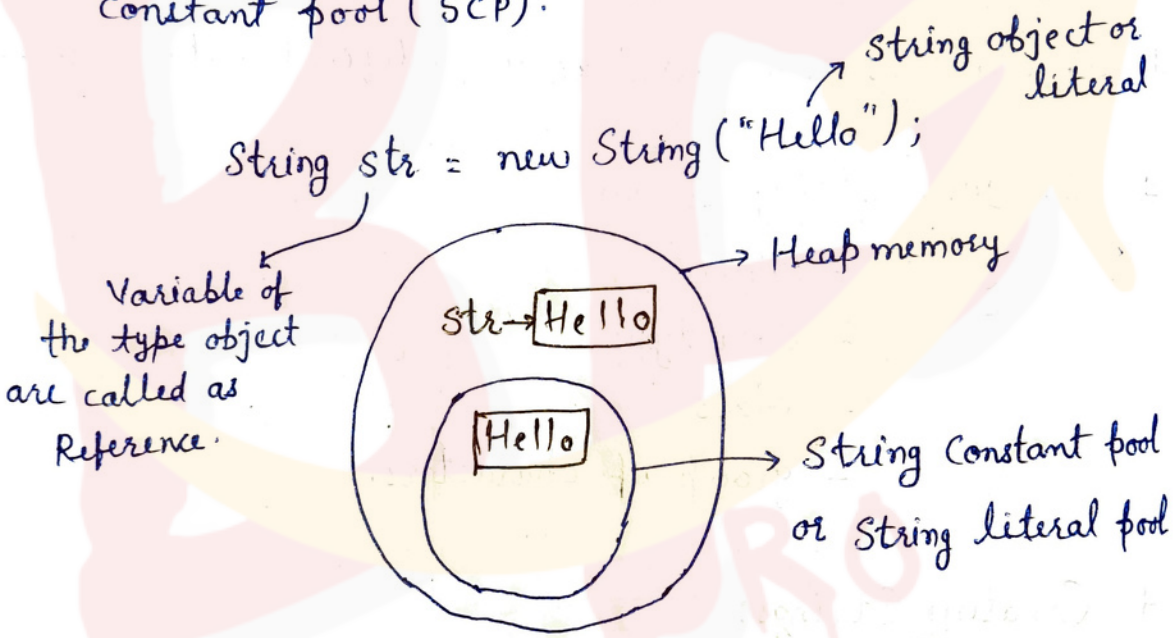
ii) Using string literal

i) Using new keyword :

We can create an object to string class by allocating memory using new operator.

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

→ Whenever 'new' keyword is applied the obj is created in heap memory and the memory occupied by the string literal is in String constant pool (SCP).



- two objects will get created
  1) One is under heap area
  2) another is under String Constant Pool(SCP)

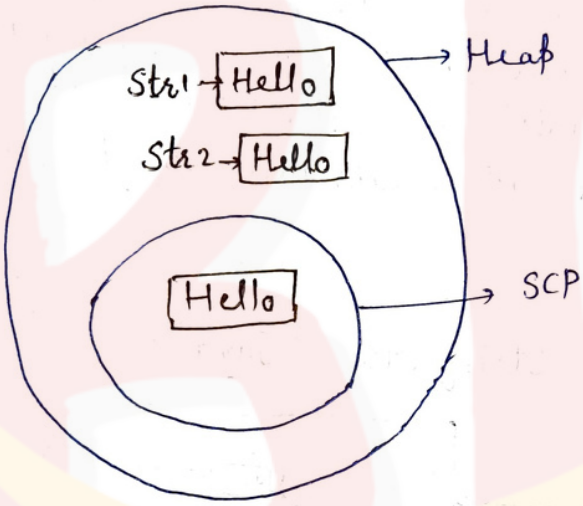
* String Constant Pool (SCP) -

SCP is special memory location present in heap area which is used to store string literal.

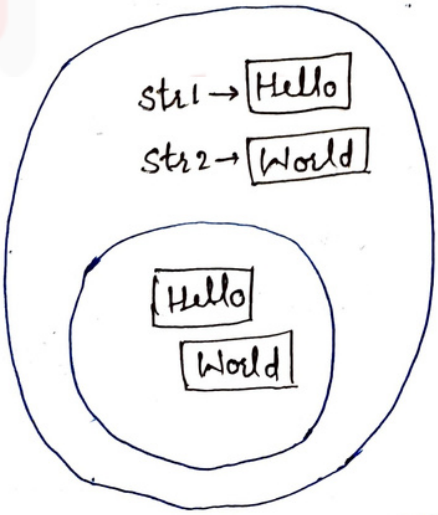
Note:

When same literal is used then it doesn't create another obj. All are pointing only one object.

Eg:- String str1 = new String("Hello");
String str2 = new String("Hello");



Eg:- String str1 = new String("Hello");
String str2 = new String("World");



2) Using String Literal -

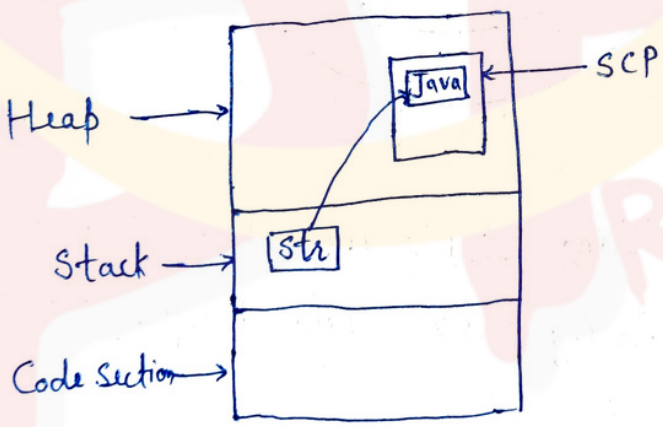
We can create a string by assigning a group of characters to a string type variable.

```
String str = "Java";
```

* In this case, object will create in only string constant pool.

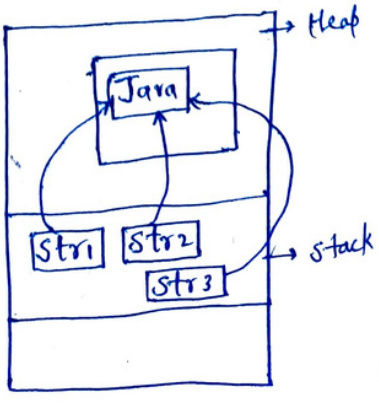
* Here, JVM will go to SCP first and check if there is any object present with string data, if it is there it will assign reference to it, if it is not there it creats new object and assign reference

Eg:- String str = "Java";



Eg:-

```
String str1 = "Java";
String str2 = "Java";
String str3 = "Java";
```



* String Constant pool is not applicable for garbage collection as JVM internally creates reference variable for each string literal object.

* String objects are immutable, they can't be modified.

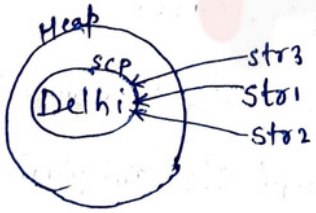
Q. Why Strings are immutable in java?

String are immutable in java because string objects are catched in String pool. Since catched string literals are shared b/w multiple references, there is always risk, if we change data of one string object. It will effect multiple references thats why java says that string objects are immutable ie once we created we cannot modify it.

Eg:- Person1 → String str1 = "Delhi"

Person2 → String str2 = "Delhi"

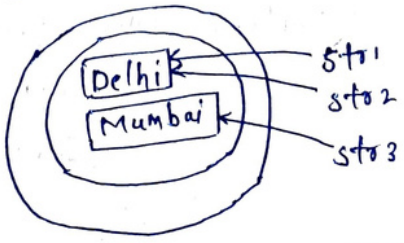
Person3 → String str3 = "Delhi"



But, If a person change his city then

Person3 → String str3 = "Mumbai";

(Because of String immutability it creates another reference and doesn't effect previous one.)



* Methods of string class -

The methods creates a new string before giving the results.

The new objects is then created in heap memory.

1) str.length() -

It returns the length of string mentioned.

Eg:- String str = "Java";

int l = str.length(); → 4

2) str.toLowerCase() -

It converts the given characters of string into lower case.

String str = "Java"

S.o.p(str.toLowerCase()); → java

3) str.toUpperCase() -

It converts the given characters of string into upper case.

String str = "computer";

S.o.p(str.toUpperCase()); → COMPUTER

4) str.trim() -

It is used to remove the leading and tailing spaces from the array if there are any.

String str = " Welcome ";

S.o.p(str.trim()); → Welcome

5) str.substring(int begin) -

It returns a new string by giving the part of a string from where the index is given.

```
String str = "Welcome";
S.o.p(str.substring(3)); → come.
```

6) str.substring(int begin, int end) -

It works same as the above but the starting and ending index both can be given in the substring.

```
String str = "Welcome";
S.o.p(str.substring(3,6)); → com
```

7) str.replace(char old, char new) -

It replaces the old character with the new character.

```
String str = "Welcome";
S.o.p(str.replace('e','#'); → W#lcom#
```

8) str.startsWith(String s) -

To find the particular starting word of a string.

```
String str = "www.abcd.org";
boolean s = str.startsWith("www");
S.o.p(s); → true
```

9) str.endsWith(String s) -

To find the particular ending word of the string.

```
boolean s = str.endsWith("org");
S.o.p(s); → true
```

10) str.charAt(int index) -

To find the particular character present on the index given.

Eg:- String str = "Mohit"; (0 1 2 3 4)

S.o.p(str.charAt(3)) → i

11) str.indexOf(String s) -

To find the index of the given character.

String str = "www.abcd.org"; (0 1 2 3 4 5 6 7 8)

S.o.p(str.indexOf('.')); → 3

S.o.p(str.indexOf('.', 4)); → 8

→ If character will not found. it will print '-1'

starting point from where it starts the searching

12) str.lastIndexOf(String s) -

To find the index of the given character from the end of the array.

13) str.equals(String s) -

To check whether the contents of two string are equal or not.

14) str.equalsIgnoreCase(String s) -

To check the whether the contents of two string are equal or not but it does not depend upon the case of character.

```
String str1 = "JAVA";
String str2 = "java";
String str3 = "Python";
String str4 = "Python";

S.o.p(str1.equals(str2)); → false
S.o.p(str3.equals(str4)); → true

S.o.p(str1.equalsIgnoreCase(str2)); → true
```

boolean type

15) String.valueOf(int i) -

It is used to convert a given data type such as int, float, double etc to its corresponding string representation.

```
int num = 123;
String str = String.valueOf(num);
S.o.p(str); → 123 (It is string)
```

Q. WAP to check count of e/E character present in a string "Elephant".

```
Public class Test
 {
    Public static void main (String[] args)
     {
        String str = "Elephant"
        int count = 0;
        for(int i=0; i< str.length(); i++)
          {
            if (str.charAt(i)== 'e' || s.charAt(i)=='E')
                  count++;
          }
        S.o.p("Count of E/e is :" + count);
     }
 }
```

Q WAP to reverse a ~~string~~ given String.

```
Public class Test
 {
    P. s. v. m(String[] args)
     {
        String str = "Welcome";
        String rev = " ";
        for(int i= str.length()-1; i>=0; i--)
          {
            rev = rev+ str.charAt(i);
          }
        S.o.p("Revesse of a string is" + rev)
     }
 }
```

Q. WAP to check whether the given string is pallindrome or not.

```
Public class Test
 {
    Public static void main (String arg[])
     {
        String str = "mam";
       for(int i = str.length()-1; i >=0 ; i--)
        {
            String rev = rev + str.charAt(i);
        }
       if ( str.equals(rev))
            S.o.p("given String is Pallindrome");
     }
 }
```

Q. WAP to print all characters only once.

```
Public class Test
 {
    Public static void main (String arg[])
     {
        String str = "aaabbccc";
          String ans = "";
       for(int i=0; i < str.length(); i++)
        {
            char ch = str.charAt(i);
            if( ans.indexOf(ch) == -1)
                ans = ans+ch;
        }
     }
 }
```

O/P = abc

16) Str.split(delimiter) -

This method is used to split a string into an array of substring based on the delimiter that is passed as an argument.

```
String str = "apple, banana, grapes";
String [] fruits = str.split(",");
for(String ans : fruits)
    s.o.p(ans);
```

o/p - apple
banana
grapes

```
String str = "java";
String c[] = s.split("") ; // Each character of str will be breaked and stored in array
```

o/p - c[0] = j
c[1] = a

* Difference b/w equals() and == with respect to String.

equals() → compares two String based on String data

== → Compares two String based on reference.

```
String s1 = new String("Java");
String s2 = "Java";
String s3 = "Java";
S.o.p(s1.equals(s2)); → true
S.o.p(s1==s2); → false
S.o.p(s2==s3); → true
```

17) str.concat() -

This method is used to combine two strings. And it returns combined string.

```
String str = "Hello";
S.o.p(str.concat("World")); → Hello World
```

* The key difference b/w concat() and append() is that concat always returns a new string while append modifies the existing string buffer.

* Regular Expression in java -

Regular expression in java is a pattern that is used to match a sequence of characters in a string. It is used to search, replace and manipulate.

→ Matching symbols -

A matching symbol allows you to match one or more characters.

- **'.'** - Matches any single character./ For single charater it is true.

Eg:-
```
String str1 = "a";
String str2 = "abc";
S.o.p(str1.matches("."));  → true
S.o.p(str2.matches("."));  → false
```

- **[abc]** -

Range or set of character/ the string is true if the alphabet is either 'a' or 'b' or 'c'.

```
String str1 = "a";
S.o.p(str1.matches("[abc]"));  → true

String str2 = "p";
S.o.p(str2.matches("[xyz]"));  false
```

- **[a b c] [v z] -**

  Range of multiple symbal / the string is true, if first symbal is among a, b & c and second is among v & z.

- **[^a b c] -**

  The string is true if the symbol is ~~from th~~ other than a, b, c.

- **[a-z1-9] -**

  The string is true if the symbal is from the range a-z ~~and~~ or 1-9.

  Eg:- - Check number is hexadecimal or not -

```
String str = "A21B3";
S.o.P(str.matches("[0-9A-F]+"));  → true
                    ↳ quantifiers
```

- Remove special character from the string

```
String str = "a!B@c#1$2%.3";
S.o.P(str.replaceAll("[^a-zA-Z0-9]", ""));
```

o/p - aBc123

- **A|B -**

  It is true for sigle alphabet either A or B.

* Meta character -

- \d - It will be true if it is a digit among 0-9.
- \D - It will be true if it is any symbol other than digits.
- \s - It will be true if there is just a space.
- \w - It will be true if there are any symbols other that space.
- \W - It will be true if it is any symbol other than alphabets.

* Quantifiers -

These are used to define the no. of symbols you want.

- '*' - It represents no. of occurrences of any of the characters for zero or more times.
- '+' - It represents no. of occurrences of any of the characters for one or more time.
- '?' - Matches zero or one occurrence of the preceding character or group.
- {X} - It represents any of the characters for the size of X value given.
- {X Y} - min and max size given

Eg:- Remove extra spaces from the string.

```
String str = "abc  de fgh  ijk";
str.replaceAll("\\s+", " ");
```

- Split the string letter by letter.

```
String str = "The quick brown fox";
String[] words = str.split("\\s+");
for(String s : words)
    S.o.P(s);
```

Q. WAP to find the email id is on gmail or not and find username and domain from email.

```
public static void main(String[] args)
{
    String str = "Programmer@gmail.com";
    int i = str.indexOf("@");
    String uname = str.substring(0, i);
    String domain = String(i+1, str.length());
    S.o.P("Username : " + uname);
    S.o.P("Domain : " + domain);
```

Finding username and domain

```
    int j = domain.indexOf(".");
    String name = domain.substring(0,j);
    S.o.P(name.equals("gmail");
    or S.o.P(domain.startWith("gmail");
}
```

Checking

* String Buffer and String Builder -

→ These are classes which are present in java.lang package.

→ String buffer and String builder objects can be created only by using 'new' keyword.

```
StringBuffer b1 = new StringBuffer("Java")
StringBuffer b2 = "Java"; // Invalid

StringBuilder b3 = new StringBuilder("Java");
StringBuilder b4 = "Java"; // invalid.
```

→ StringBuffer and StringBuilder objects are mutable i,e once we created we can modify it.

→ StringBuffer and StringBuilder will get created in heap area only.

→ The main difference between StringBuffer and Builder is-

- StringBuffer is thread-safe which means that it can be used in multithreading environment.
- On the other hand, StringBuilder is not thread-safe and should be used in a single-threaded environment.

Note: **Performance**

String	StringBuffer	StringBuilder
↓	↓	↓
slow	fast as compare to string	faster than both