Strings.

→ Nothing but Character Array

⇒ User input :-
↳ Two ways.

```
⇒ String str = sc.next ( ) ;
⇒ string str = sc.nextline( ) ;
```

sc.next ( ) → Just consider 1st word. Not entire statement. → After Space it will skip everything

I/p ⇒ Chandrakant Mane.

sc.next ( ) → Chandrakant.

sc.nextline( ) → "Chandrakant Mane.

⇒ Declaration :-

```
String s = "Hello" ;
```

CharAt( ) And length( ) methods.

length( ) ↳ use' to find length of String

CharAt( ) ↳ use to Access ~~at~~ a perticular Char ʒ. in String using its index

```
             012345
String s = " Raghav" ;               s.length( ) ⇒ 6.
   s[3]        s.charAt(3) ──→ 'h'
    ✗
```

indexOf( ) And compareTo ( ).

compareTo ( ) → is used to compare 2 Given String lexographicelly → 'a' & 'A' are different.

indexOf( ) ↳ Returns the index of first Occurance.

```
s.indexOf ('a') ;
   ↳ 1
s.indexOf ('z') ;
   ↳ -1
```

lastIndexOf → Last Occurance idx.

```
s.lastIndexOf('a') ;
   ↳ 4
```

```
c = "abc"
a = "abc" ;          b = "dbc" ;
a.compareTo( b) . ;  ──→ Case①.
'a' - 'd'
 ↓     ↓
 97   100    Ans ⇒ 97-100 ⇒ -3.
```

→ Case②.

```
a.compareTo (c) ;  Ans ⇒ 0
```

```
c = "abe"
a = "abezzzz"  → Returns this Length
a.compareTo (c) ; ──→ 4
c.compareTo (a) ; ──→ -4
```

Contains( )

~~↳ Checks that a substring is~~

↳ Checks that is String contaiing the given substring. in continouse order

```
s. Contains ("ghav") ;  ──→ True
s. contains ( "hard") ; ──→ false
```

StartsWith ( ) or endsWith ( ).
↳ similar to contains
↳ just cheks from start

```
s. starts.with ("Rag") ; ──→ True
```

→ Returns the modified String didn't make changes in actual one

toLowerCase() / to UpperCase().

S = "Raghav"

s.toLowerCase(). → "raghav"

s.toUpperCase(). → ~~Raghav~~ RAGHAV.

concat(). → use to Concat two strings.

a = "abc"   b = "xyz".

a.concat(b); → abcxyz. → It also returns the modified string, didn't make changes in actual one [In this case in a]

Substring(i) && substring (i, j).

i → start idx

(i, j) → i start idx, j → end idx +1.

→ Returns the substring from start index to end last ~~idx~~ index.

[Substing → Continous part of string]

s = "abcde"; (0 1 2 3 4)

s.substring(3); → "de"   (3 → start index.)

s.substring(1, 4); → "bcd".   → Gets part of string from i to j-1 index

s.substring(2, 2); → ""

s.substring(0, 5); → abcde

+ Operator → It is to costy. Every time creates New string

It follows the Rule of operators

Sout("abc" + 10 + 20); → "abc1020"

Sout(10 + 20 + "abc"); → "30abc"

Integer.toString(n). → Convert Given No. as it is in string

Integer.toString(789); → "789".

Interning

String is Immutable → unable to Change

String s = "Raghav";

s.charAt(0) = 'M'; → Error.

S = "Madhav"; → This is possibe. ~~Not~~

String q = "Madhav";

String x = new String("Madhav")

Internally ⇒

S → "Raghav"
S → "Madhav" ← q
Now S is pointing to Madhav.
x → "Madhav"

String Immutability in Java.

↳ we cannot Change individual Character in a string. we can but we will waste lot of time & Space.

It checks Char by Char & gives the Ans

equals ( ) V/s == · Only for string it.

↳ [So we use this.]

== → equality Operator.

↳ Gives the Answer based on their Addresses.

Concept :-

```
String s = "abcxyz";
```

#Both s & q are pointing to same string.

```
String q = "abcxyz".
```

when you to this stuff new memory is created

```
String t = "abc" ;
t += "xyz" ;
Sout (s == t) ;  → False
```

It is giving based on memory Address.

```
Sout (s == q). ;
  ↳ True

String p = new string ("abcxyz");
sout (p == s) ; → false.

s.equals( t ) ;  → True
s.equals ( p) ;  → True
s.equals ( q ) ;  → True
```

—— x —— x —— x —— x —— x —— x

StringBuilder → mutable

→ You can use string methods with String Builder.

⇒ Declaration & Initialization

```
StringBuilder sb = new StringBuilder ("abc");
StringBuilder sb1 = new StringBuilder (10);
```

↳ It will reserve the space in memory

[You can also give the initial capacity]

⇒ By default Capacity is 16

```
sb.capacity() ; → 16 (default) + 3 (length) => 19.
sb1.length () ; → 0
sb1. capacity () ; → 10.
```

Capacity & length is different.

User Input : -

```
String s = sc.nextLine ();
StringBuilder sb = new StringBuilder (sc.nextLine());
```

M T W T F S S

Page No.:

Date:

YOUVA

setCharAt()

StringBuilder sb = new StringBuilder("Raghav");

index

Value ~~a~~ needed to set.

sb.setCharAt(1, 'a');

Sout(sb); → "Raahav".

append() → use to append Anything in StringBuilder from Back

→ ~~Array also~~ Character Array also

sb.append("xyz"); → "Raahavxyz".

sb.append(25); → "Raahavxyz25".

0 1 2 3 4 5

sb = "Raghav";

deleteCharAt(idx);

→ Based on idx Particular Character will be removed from String Builder. Also shifts the characters to fill the index

delete(); → used to delete an ~~p~~ substring or continous part of string

→ sb.delete(2, 5);

→ "Ray".

it will conside -1 means (4)

insert(~~i~~ idx, value) → index → value need to insert. → It can be int, char, String ...

Also shifts Rest of the Elements.

Sb.insert(2, 'c'); → "Rac~~hav~~ghav".

reverse() → use to reverse A StringBuilder

→ Sb.reverse() → "vahgaR".

Char[] a = new ~~int~~ Char[6];

→ Default value is null Character ['\0'].