Date: / / Page no:

Strings :-

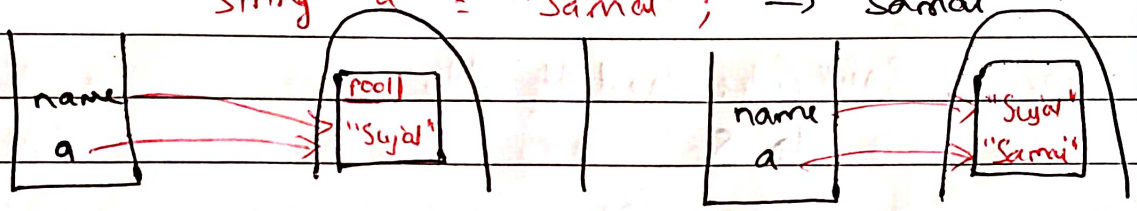
↳ most used class in Java

↳ it is a class as everything that starts with a capital letter in Java is a class.

String pool : Seperate memory structure inside the heap memory. It makes our program more optimised.

↳ Similar values of Strings are not recreated.

for eg :- String name = "Sujal";
String a = "Sujal"; → Sujal.
String a = "Samai"; → Samai



★ Strings are immutable i.e. values can't be changed.

↳ So even if we think that changing object might get reflected to all variables, but with Strings you can't change in the first place.

↳ If we update a variable, it just creates a new object, it doesn't updates the older value.

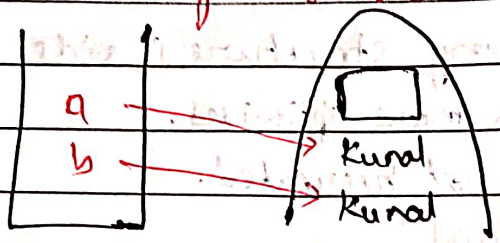
→ "==" checks if ref. variable are pointing to same obj.

a → "Sujal" b → "Sujal"	a → "Sujal" b → "Sujal"
a == b → False	a == b → true

Date: / / Page no:

- How to create diff. objects of same value?
  - Just like array, we can use "new" keyword.

```
String a = new String ("Sujal")
String b = new String ("Sujal")
// creating these values outside the pool but in heap
```




```
a == b          // false
a.equals(b)     // true
```

  - When you only need to check value, use .equals method.

```
Sout(a.charAt(0));
  -> S
```

  - Whatever we give inside sout to print, it converts it into string and returns it.
  - But in case of objects like arrays, hashmaps etc. Java gets confused what exactly it needs to print.
    - For this we use diff. tostring method for diff. objects & overwrite the original tostring method.

for eg:- for arrays we use

Arrays.toString() in sout

Date: / / Page no:

- Pretty Printing :
- Pretty Format → It is used to round off or to print less decimal values of a particular number.

  eg :- Sout ("Pie : %0.3f", Math.PI);

  ↓ use printf not println

  %0.3f → placeholder

- Placeholders or format specifiers in Java :
  - %c → character
  - %d → Decimal number
  - %e → Exponential - floating point no.
  - %f → Floating point no.
  - %i → Integer
  - %o → Octal (base 8)
  - %s → String
  - %u → unsigned integer
  - %x → Hexadecimal (base 16)
  - %t → Date / Time
  - %n → Newline

- String operations :

  Sout ('a' + 'b'); → 195

  Sout ("a" + "b"); → ab // concatenate

  sout ((char) ('a' + 3)); → d

  Sout ("a" + 1); → a1

When an integer is concatenated with a string, it is converted to its wrapper class Integer → calls tostring

Date: / / Page no:

Sout ("Kunal" + new ArrayList <> ()); → Kunal []

→ Whenever there is a object, tostring is called

→ "+" operator in Java is defined only if data type is primitive or if one of the values i.e. to be printed is String type.

Sout (new Integer (56) + new ArrayList <> ()); ✗ error

★ On String objects → "+" operator is overloaded
↳ operator overloading

★ Operator overloading is supported in C++, Python etc and through that we can modify what an operator does in a particular instance. But in Java operator overloading is not supported.
↳ Only supported for Strings ("+")

○ String Performance :

for eg :-

```
String series = "";
for (i=0; i<26; i++){
    char ch = (char) ('a' + i);
    series = series + ch;
}
Sout (series);          // abcdefghijklmnopqrstuvwxyz
```

→ If we observe closely, at each iteration a new object is created everytime as strings are immutable. It is creating a new object, copying the old one & appending the changes.

Date: / / Page no:

[a, ab, ... abcdefghijklmnopqrstuvwxy] → waste
1 2 25

→ This means that all the old strings which are large will have no ref. variable → wastage of space

[1+2+3+ ..... +25(N)] ⇒ N(N+1)/2 ⇒ N²

Time Complexity → O(N²)

- String Builder → A seperate class, which instead of creating a new string object at every iteration, just adds the object to the older one. Hence, no new object is created.

→
```
StringBuilder builder = new StringBuilder();
for (int i=0; i<26; i++){
    char ch = (char)('a'+i);
    builder.append(ch);
}
Sout(builder.toString());
```

- String Methods :

→
```
String name = "Sujal";
Sout(Arrays.toString(name.toCharArray()));
```
↳ [S, U, j, a, l]

→ lowercase, uppercase

→ indexof, strip etc.

```
Sout(name.toLowerCase()); → sujal
```