

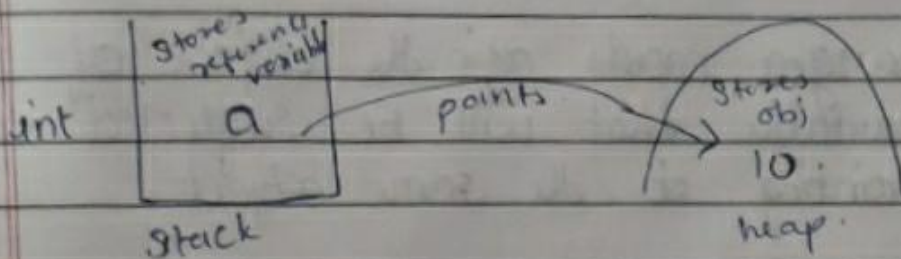
Strings & Stringbuilder in

① Stack memory :- When we declare a variable
eg :- `int a = 10;`

So, here the reference variable is stored in stack memory.

② Heap memory :- Reference variable stored in stack memory is pointing to the object of that variable are stored in heap memory.

`int a = 10`
s h



③ Memory allocations :- Memory allocations specifies the memory address to a program.

• There are two types of memory
a) Stack memory b) Heap memory

4) Static memory allocations :- It performs type checking at compile time.

- Here, errors will show at compile time.
- Declare datatype before you use it.
- More control & Runtime errors are reduced.
- If you have to write a little bit more code, but you have more control over the type of data.

4d) Dynamic memory allocations:- Performs type checking at runtimes.

- Here, errors might not shown at all the program run.
- No need to declare a datatype of a variable.
- It saves time in writing code but might give error at runtime.

5) Garbage collections:-

- More than one reference variable can point to the same object.
- If any changes made in the object of an reference variable that will be reflected to all others pointing to the same object.
- If there is an object without reference variable then the object will be destroyed by "garbage collection".
- So that's how garbage collection works.

Q What are strings? Strings will be in double quote.

Ans:

- Strings are pile or a sequence of characters for Eg "kunal".
- It's a datatype & it is a non-primitive. so, a non-primitive datatype.
- Syntax: Everything that starts with a capital letter is a class.

```
String name = "Deepa-Chaurasiya";
System.out.println(name);
```
- It is the most commonly used class in the JAVA's class library.
- Every string that you create it's actually an object of type String.

*** Concepts:

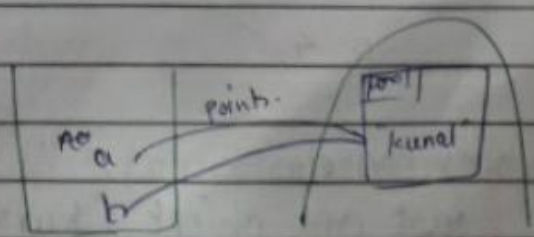
① String pool: It is a separate memory structure inside the heap.

Q Why separate pool? Why not just putting it out in the heap normally? like for every other object!

Ans: All the similar values of strings are not like recreated in the pool.

Eg:

String a = "kunal"
String b = "kunal"



so it's gonna be like it already exists in the pool no need to create it again pt to the kunal.

Use case :- It makes our program more optimized.

Note:-

- If you try to change this object via this reference variable it will not change for b.

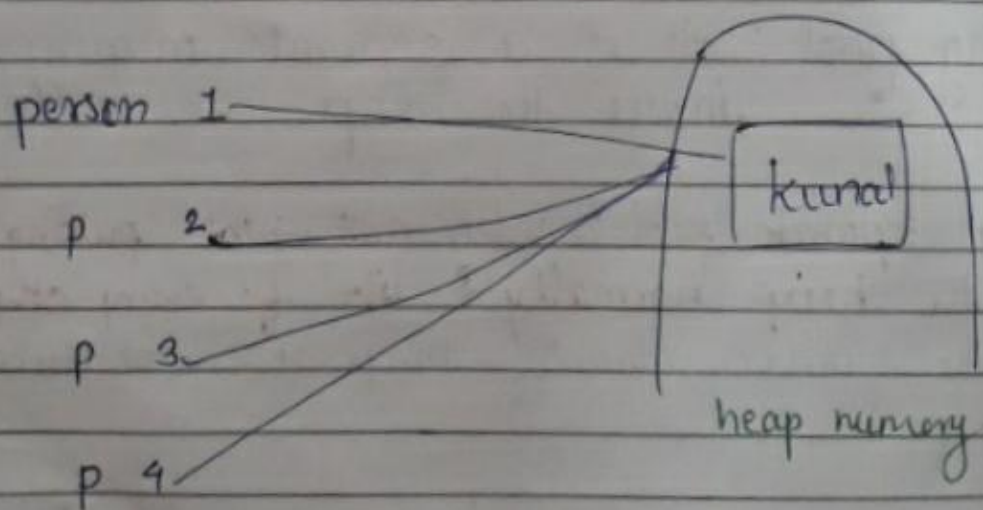
Q Why?

Ans:- Immutability.

- Strings are immutable in java, we can't change the object or modify object.
- If you wanna rename to "kunal" then you've to create a new object for that.
- Strings are immutable for security purpose.

Why Strings are immutable?

Ans:-



All person's name is kunal. \therefore All of them will be having just one object "kunal".

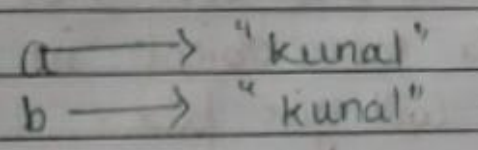
• Suppose one person decides to change their name if it was not immutable, if a person 1 decided to change their name to karan. So, that will change his name to karan

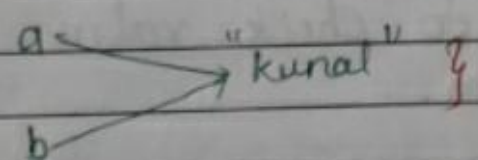
• If it was allowed to modify this then all person 1, 2, 3, 4 will name will be karan in the database.

• Therefore, for the security reasons strings are immutable.

* Comparisons of Strings.

① == method :- comparator

Eg ①  } In this case == will give false.

②  } In this case == will give true

• Computer actually checks for value and reference variable. If the reference variable is pointing to the same object.

③ How to create different objects of same values.
Ans:-

Ex: Suppose,

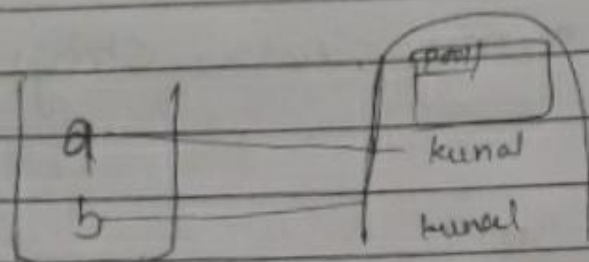
take an ip as a string.

By using new keyword we can create a new obj. →

```
String a = new String("kunal")
```

```
String b = new String("kunal")
```

// creating these values outside the pool but in heap because it is object so it will be in heap only.



`a == b` // false.

even though the values are same the but these two `a` & `b` are not pointing to the same object in that case it will give false.

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

```
so, String name1 = new String("kunal");  
String name2 = new String("kunal");
```

```
System.out.println(name1.equals(name2));  
o/p :- true
```

Here does not care whether the reference variable are pointing to same object or not it just cares about the value.

* class is a name group of ~~pro~~ properties and functⁿ.

* We can't do:

```
System.out.println(name1[0]);
```

- But in arrays we can do this.

- Though it's sort of collection of characters but we cannot do \uparrow do this.

• So, we have to use a method called charAt().

```
System.out.println(name1.charAt(0)); // r = k
```

↓
variable of type printstream.

↓
class - it has method in it
called
println

* function overloading ? or method overloading.

Ans:

Function overloading happens when two functions have same name.

eg:

```
fun () {  
    // code  
}  
  
fun () {  
    // code  
}
```

} * function overloading.

```

1) fun (inta) {
    // code
}

fun (inta, int b) {
    // code
}

```

} It's allowed
having different arguments
with same function/
method name.

- At compile time, it decides which function to run.

* Pretty printing :-

```

float a = 453.1234f;
System.out.printf("Formatted number is %.2f", a);

```

Place holder ↓

printf :- Formatted string

name declared

- Well %. means a place holder & till how many decimal value do we want. for eg:- 2. %.2f (f because it's float).
- It rounds off all as well.

* for π

```
System.out.printf(Math.PI);
```


* for String

~~System.out.println~~

System.out.printf("Hello my name is %s and I am %s", "Kunal", "Student");

c/p :- Hello my name is Kunal and I am Student.

So, the order in which you have place the placeholders, int that order only you've to put the variable

* Some common format specifiers :-

- ① %c :- Character
- ② %d :- Decimal number (base 10)
- ③ %e :- Exponential floating-point number
- ④ %f :- floating-point number
- ⑤ %i :- Integer (base 10)
- ⑥ %o :- Octal number (base 8)
- ⑦ %s :- String
- ⑧ %u :- Unsigned decimal (integer) number
- ⑨ %x :- Hexadecimal number (base 16)
- ⑩ %t :- Date/Time
- ⑪ %n :- New line.

Operator overloading.

* Operator + ("Overloaded for String type")

① `cout ('a' + 'b');` o/p = 195

↓
Here the operator is converting this into its character value integer value. and then adding the Unicode or ASCII value (character)

② `cout ("a" + "b");` o/p = ab

It concatenate the strings.

③ `cout ('a' + 3);` o/p = 100

④ `cout ((char)('a' + 3));` o/p = d

Converted the 100 into a character. (C++ casting).

Note:- 1) When you're doing the addition with characters it converts into its value into a number then it use that numbers to solve the problem.

2) But with string it's not doing that, it actually taking the string value

⑤ `cout ("a" + 1);` o/p = a1

// integer will be converted to integer that will call toString()

Note:- 1) When an integer is concatenated ("added") with a string it is converted to its wrapper class integer

2) This is same as : "a" + "1".

⑥ `System.out ("Kunal" + new ArrayList <> ());`
o/p :- Kunal ()

Initially arraylist is empty.
then it's converted into Kunal

So, we know this will be like an object of type integer hence, it's calling the toString, which is returning a normal brackets. Since it's empty, it will return an empty array.

⑦ `System.out (new Integer (56) + new ArrayList <> ());`
= error.

Operator '+' cannot be applied to integer and array list.

* Operator '+' in Java is only defined for primitives and when any one of these values is a string.

* Operator '+' you can only use with primitives and you can only use this with all the complex objects as well.

But, the only condition is at least one of these objects should be of type string.

Eg: `System.out (new Integer (56) + " " + new ArrayList <> ());`

this will work

complex obj

obj of type string

complex obj

Here the entire result will be of string type. o/p:

56[]

So, on string objects the plus operator is being overloaded, because it concatenate more than one strings.

- In java operator overloading is not supported for some software engineering but consideration. But, in C++ it is supported.
- So, you basically + operator, you can basically modify what the plus '+' operator is doing in C++ & also in python.
- you can also make it act like as a multiplication or substring subtraction, you can add complex data type as well.
- But this results in poor code, that is why in Java it is not supported.
- Java has only given us operator overloading exception for strings, but you cannot do it on your own. If you want to merge two complex objects of your own type like array, hashmaps, it will not allow ever for the modification.
- It's "only" operator that is intentionally overloaded in java to support string concatenation or string joining.
- `new Integer()` :- In future it's going to be removed.

⑧ `scout ("a" + 'a');` o/p :- aa if one of the data type is string ans will be string.

* Methods that String provide (Some).

1) `.toCharArray()` :- It converts ^{strings} char into ^{array of} characters.

Eg:- String name = "Kunal Kushwaha";

`System.out.println(Arrays.toString(name.toCharArray()));`

Output: [K, u, n, a, l, , K, u, s, h, w, a, h, a]

PS: Even that space will be counted.

2) `.length()` :- It will give you the length.

3) `.toLowerCase()` :- It will convert into lowercase.

It's not actually going to convert the original object because of immutability.

4) `.indexOf` :- It will give the index value.

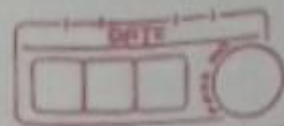
5) `.lastIndexOf` :- It will give the last index value.

6) `.strip()` :- White spaces are removed.

7) `.split()` :- add a regex first :- After adding regex it will split ~~on~~ it over there.

`.split(regex)`
Regex

Code of palindrom string.



Time On

```
public class PalindromString {  
    public static void main (String[] args) {  
        String str = "abcba";
```

```
        System.out.println(isPalin(str));
```

```
    }  
  
    static boolean isPalin (String str) {
```

+this will be having null.

that means

if this is = 0

```
        if (str == null || str.length() == 0) {
```

here length is a method

```
            return true;
```

```
        }
```

```
        str = str.toLowerCase(); → Converts the String into lowercase
```

```
        for (int i = 0; i <= str.length() / 2; i++) {
```

```
            char start = str.charAt(i);
```

(str.length - 1) = last index *← char end = str.charAt(str.length - 1 - i);*

every time we will be doing -i to, suppose i = 2,

then end index should be, end index - 2. Hence...

```
            if (start != end)
```

```
                return false;
```

```
        }
```

```
    }
```

```
    return false;
```

```
}
```

```
}
```

for empty str

you can also put null as false completely depends upon interviewer but the ans will be same.