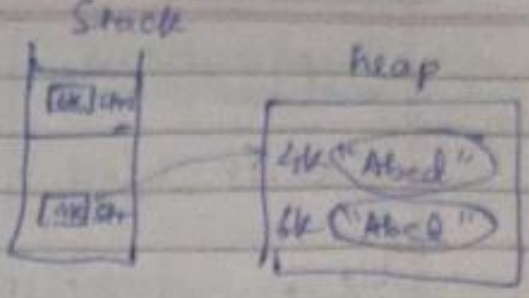
STRINGS



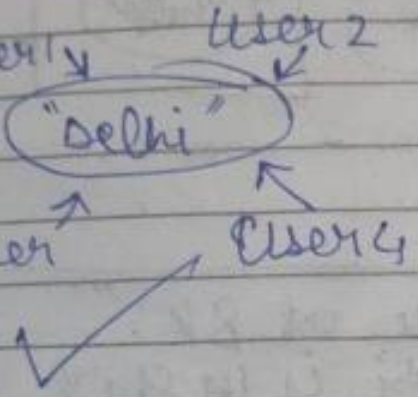
declare String

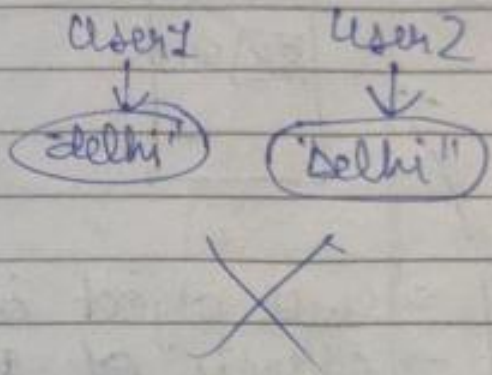
1. String str = "Abcd";
2. String str2 = new String("Abcd");

→ Advantage => Memory Save

String Interning

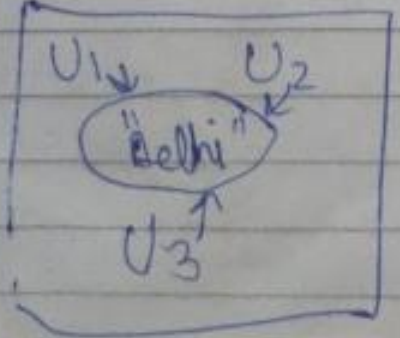
1 string literal is used and all other ~~data~~ points to this literal






```
String U1 = "Delhi";
String U2 = "Delhi";
String U3 = "Delhi";
```

* Created in special segment inside Heap



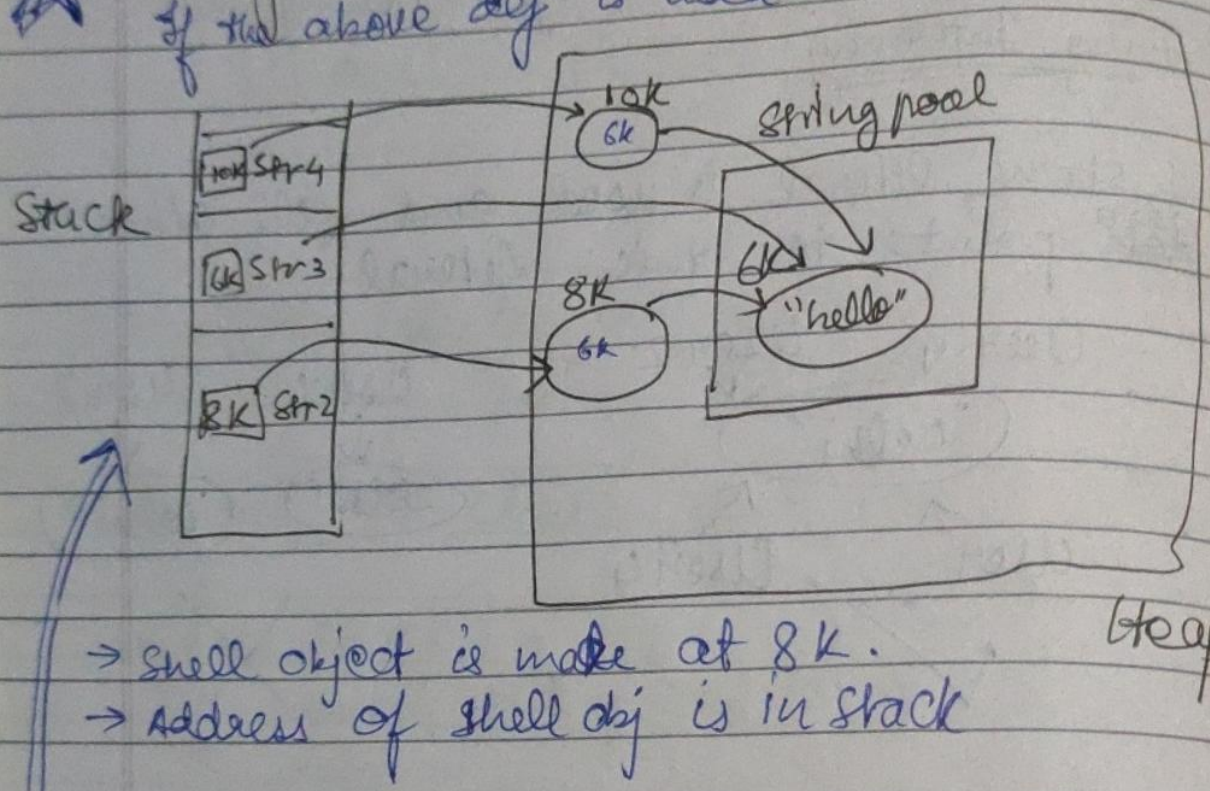
STRING INTERN - POOL
OR
STRING POOL

Implications

↳ diff b/w == and equals()

↳ Immutability

★ String str2 = new String("hello");
If the above def^n is used



→ Shell object is made at 8K.
→ Address of shell obj is in stack

★ String str3 = "hello"

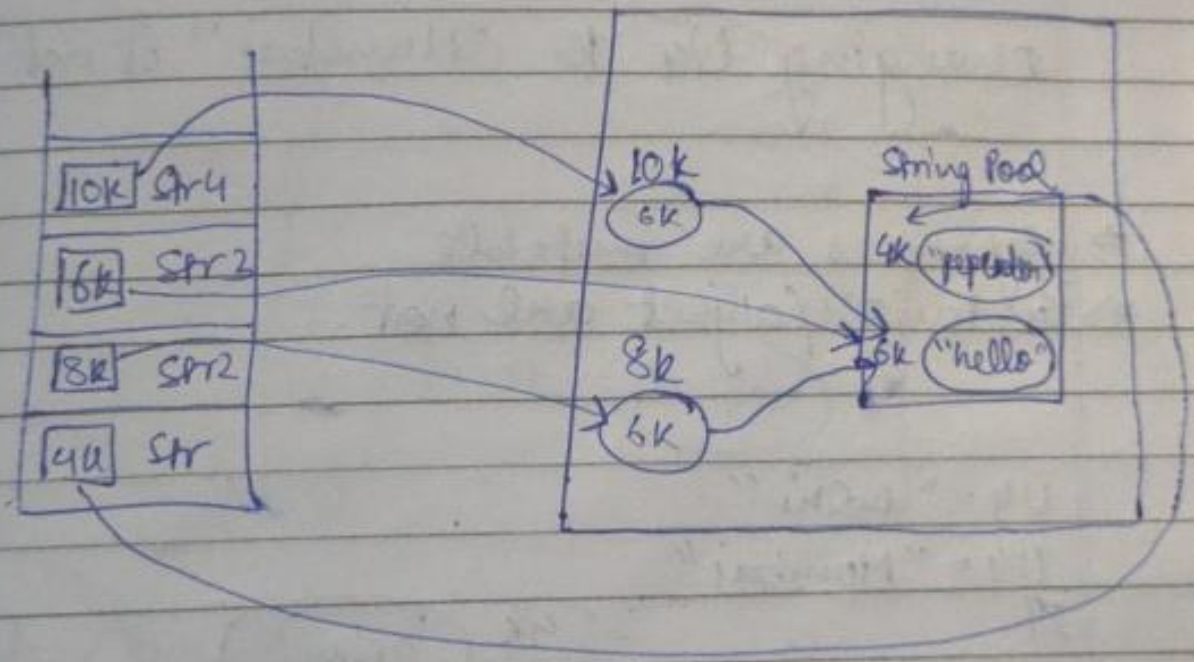
★ String str4 = new String("hello");

Implications

1. str2 == str3 False
2. str3 == str5 True
3. str2 == str4 False
4. str2.equals(str3); True
5. str3.equals(str5); True
6. str3.equals(str4) True

== compares addresses
↳ if add equal ⇒ true
& vice versa

.equals() compares addresses
↳ if add^s equal ⇒ true
↳ if add not equal
↳ if content same ⇒ true
↳ false



String str = "pepcoder";

String str2 = new String("hello");

str3 = "hello";

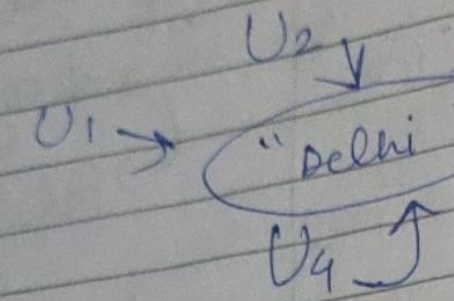
str4 = new String("hello");

str5 = "hello";

Immutability

Strings are immutable

→ due to interning we cannot change the string at an object as other object will be pointing to it.



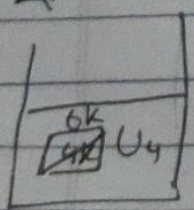
Changing U4 to "Mumbai" is not possible

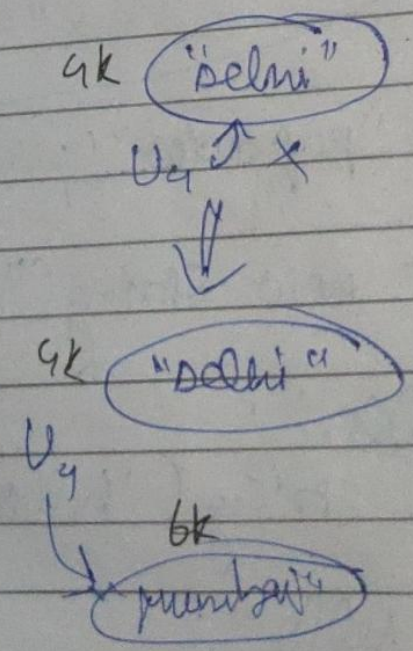
⇒ References are mutable
⇒ Instances/object are not.

U4 = "Delhi"
U4 = "Mumbai"

Reference is changed

&

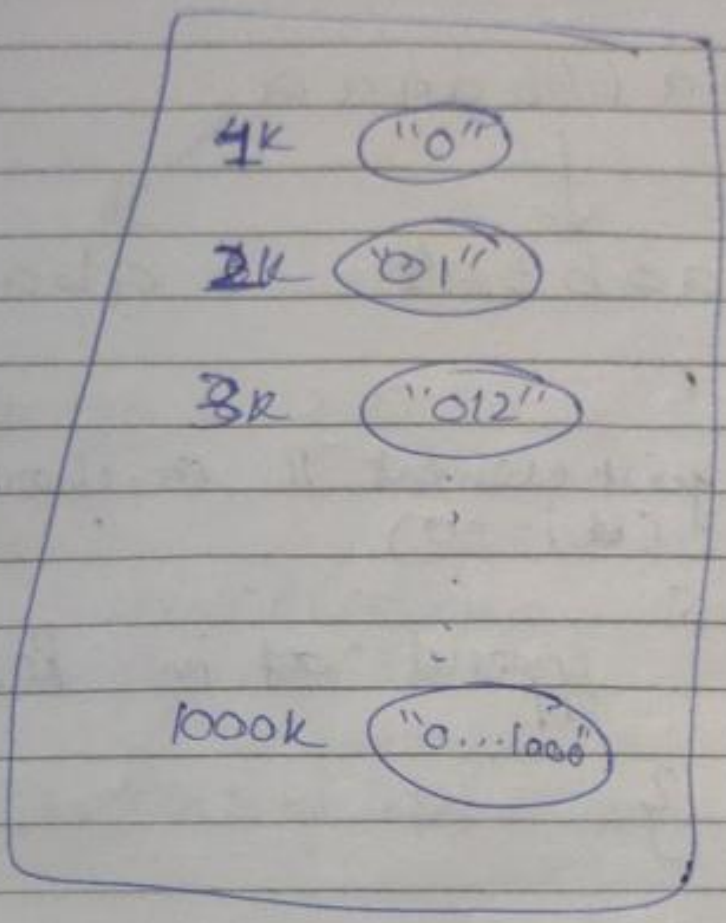




Concatenation

```
String s="0";
for(int i=1; i<=1000; i++){
    s = s+i;
}
```

1000k
:
3k
2k
1k S



Not constant

Final $1 + 2 \cdots \cdots n$

$$= \frac{n \times (n+1)}{2} = \frac{n^2}{2} + \frac{n}{2}$$

$O(n^2)$

String Compression

wwww aaa de xxxxxxx → wade x

↓

w4 a3 de x6

aaa bbb aaaa → aba

↓

a3 b3 a4

if (first element || str.charAt(i) != str.charAt(i-1))
(i == 0)

{ Append in Resultant String

}

}

→ freq of prev group
→ char => group starting
→ append last group's freq. after loop end

HW

* String With Difference of Every Two Consecutive Characters

pepCODinG

ASCII Diff

e - p = -11

Append

p-11 e ....

p - e = 11

p-11 e 11 p-45 C 12 O ....

~~char~~ ch1 = str.CA(idx)

ch2 = str.CA(idx+1);

res += ch2 - ch1

String Builder ⇒ String

↳ mutable

↳ No Interen pool concept.

Str

C++

String

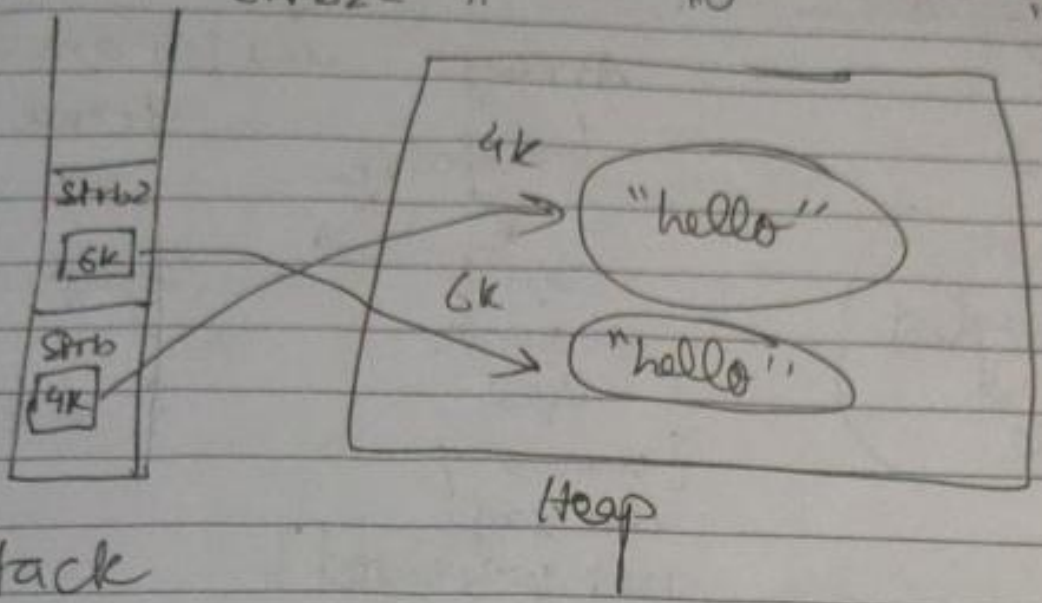
Syntax

```
StringBuilder strb = new StringBuilder("hello");
System.out.println(strb); // prints hello
strb.length(); // gives length of string
for(int i=0; i<strb.length(); i++){
    System.out.print(strb.charAt(i)+" ");
}
// update ⇒ Constant
strb.setCharAt(3, 'd'); // Changes char at i=3
strb.deleteCharAt(1); // deletes char at i=1
strb.insert(2, 'e'); // Insert at i=2
strb.append('s'); // appends at last (Constant time)
```

Variable time

Concatenation in SB is $O(n)$ & in String is $O(n^2)$

StringBuilder strb = new StringBuilder("hello");
" strb2 = " "



strb & strb2 are separate & changes in one doesn't affect other.

Vector in C++

ArrayList (Dynamic Array)

Array → Static
- can't change size
- {memory wastage not prevented}

int[] arr = new int[10]
fixed size 10

Generic
(Any type of arraylist can be made)

- can't be resized
- {not dynamic}

Syntax

ArrayList<Integer> arr = new ArrayList<>();

↖ Wrapper Class

8 primitive datatypes

Heap

int → Integer
char → Character
float → Float
double → Double
boolean → Boolean
long → Long
short → Short
Stack byte → Byte

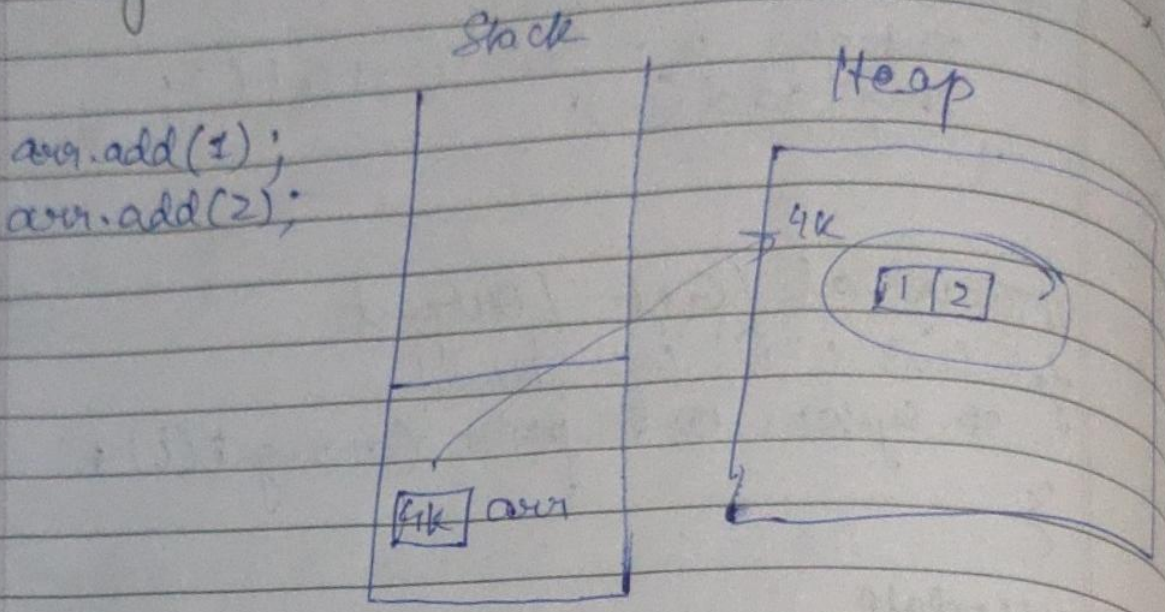
Wrapper Class (Immutable)

```
// Input
ArrayList<Integer> arr = new ArrayList<>();
for(int i=0; i<n; i++){
    Integer x = sc.nextInt();
    arr.add(x);
}

// Traversal / Get / Output
for(int i=0; i<n; i++){
    System.out.print(arr.get(i)+" ");
}

// Update
arr.set(3, 100);

// Removal / Deletion
arr.remove(2);

// Traversal / Get / Output
for(int i=0; i<arr.size(); i++){
    System.out.print(arr
```

```
// Size / Length
arr.size();

// toString() in arr                    // Output
System.out.println(arr); // [1, 2, 5, 3]
// Equivalent to
// String str = arr.toString();
// System.out.print(str)
```

Memory Mapping

ArrayList<Integer> arr = new ArrayList<>();



* For Each Loop

ARRAY

```
int[] arr = {1, 2, 3, 4};

for(int val : arr){
    System.out.print(val + " ");
}
```

ARRAYLIST

Same as ARRAY

```
for(Integer val : arr){
    System.out.print(val + " ");
}
```

Remove Primes

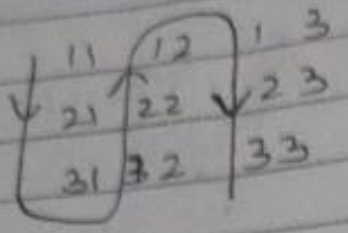
4
12, 13, 14, 15 ← Take arraylist & remove prime

output [12, 14]

```
for(int i=0; i<arr.size(); i++){
  if(isPrime(arr.get(i))==true){
      arr.remove(i);
      i--; → When we remove elements go back-ward & i shouldn't increase. So i-- ensures that
  }
}
```

2D Arrays

Wave Traversal



odd col
row ↑

even col
row ↓

⇒ columns always ↑se ⇒ column loop is outside

```
for(int j=0; j<arr[0].length; j++){
    if(j%2==0){ //even column
Top to Bottom   for(int i=0; i<arr.length; i++){
            System.out.println(arr[i][j]);
        }
    }else{
        for(int i=arr.length-1; i>=0; i--){
Bottom to Top       System.out.println(arr[i][j]);
        }
    }
}
```

SUBSTRING

a b c c b c
0 1 2 3 4 5

Print palindromic substring

01 a
02 ab 12 b
03 abc 13 bc 23 c
04 abcc 14 bcc 24 cc 34 c
05 abccb 15 bccb 25 ccb 35 cb 45 b
06 abccbc 16 bccbc 26 ccbc 36 cbc 46 bc 56 c
ij ij ij ij ij ij

i goes 0 → l-1
j goes i+1 → l

Finding all substrings

```
for(int i=0; i<s.length(); i++) {
    for(int j=i+1; j<=s.length(); j++) {
        String ss = str.substring(i,j); // get substr.
        // check if substr is palindrome
        if(isPalindrome(ss) == true) {
            System.out.println(ss);
        }
    }
}
```

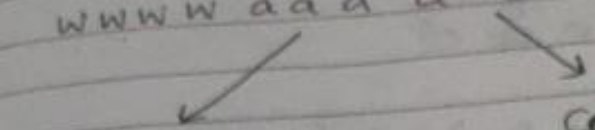
Checking if a string is palendrome

N I T I N
i → ← j

```
int i=0;
int j= s.length()-1;

while(i<=j){
   char ch1 = s.charAt(i);
   char ch2 = s.charAt(j);

   if (ch1!=ch2){
      return false;
   }
   else{ i++; j--; }
}

return true;
```

String Compression

wwww aaa d e xxxxxx



Compression 1
wadex

Compression 2
w4a3dex6

Compression 1

→ output = "";

→ Loop through entire string from i → s.length()-1

→ Check if (i == 0 || s.charAt(i) != s.charAt(i-1))
First char

→ If true ⇒ output += s.charAt(i);

→ Else do nothing

→ Return output;

Compression 2

→ Output = ""; freq = 0;

→ Traverse entire String from i = 0 → s.length()-1;

→ Check if (i == 0; s.charAt(i) != s.charAt(i-1))

→ If true ⇒ if (freq > 1) { output += freq; }
output += s.charAt(i);
freq = 0;

→ Outside if → freq++;

→ Outside loop → if (freq > 1) { output += freq }

→ Return o/p

String With Different of Every Two Consecutive characters

i/p : a b e c d

o/p : a1 b3 e-2 c1 d

Explanation: 'b' - 'a' = 1
'e' - 'b' = 3
'c' - 'e' = -2
'd' - 'c' = 1

→ output = " ";

→ for (i=1 ; i < str.length() ; i++) {

output += str.charAt(i-1);
int diff = str.charAt(i) - str.charAt(i-1);
output += diff;
}

→ output += str.charAt (str.length() - 1);

→ return output;

Toggle Case

i/p : abCdE

o/p : ABcDe

In ASCII,

'p' - 'a' = 'P' - 'A'

'p' = 'a' + 'P' - 'A'

'P' = 'A' + 'p' - 'a'

we can derive this formula

~~for~~ lowercase uppercase

lc = 'a' + uc - 'A'

uc = 'A' + lc - 'a'

```
→ StringBuilder sb = new StringBuilder(str);
→ for(i=0; i<sb.length(); i++){
    char ch = sb.charAt(i);
    if(ch>='a' && ch<='z'){
      char uch = (char)('A' + ch - 'a');
      sb.setCharAt(i,uch);
    }else if(ch>='A' && ch<='Z'){
      char lch = (char)('a' + ch - 'A');
      sb.setCharAt(i,lch);
    }
  }
→ return sb.toString();
```

Remove Primes

Ip 4
12, 13, 14, 15

Op [12, 14]

use ArrayList

⇒ Solution

```
public static void solution(ArrayList<Integer> arr)

for(int i=0; i<arr.size(); i++){
    if(isPrime(arr.get(i))==true){
        arr.remove(i);
        i--;
    }
}
}
```

Check if no. is Prime & return true or false

Remove from Arraylist

Go back ∵ i shouldn't ↑ so when element is removed.

⇒ Main → Input Arraylist

```
int n=scn.nextInt();
ArrayList<Integer> al = new ArrayList<>();
for(int i=0; i<n; i++){
    al.add(scn.nextInt());
}
solution(al);
System.out.println(al);
```

Permutation of String

I/p: abc
O/p: abc
bac
cab
acb
bca
cba

n! permutations

Observation

⇒ abc

~~a~~ b, c₂

0 | 3 | 0
2 | 0 ← Perm = 0 ⇒ remove 0th char & Print it
1 | 0 0
0 0

New string ~~b c₁~~

~~b c₁~~

~~c~~

O/p: a b c

1 | 3 | 1
2 | 0 1
1 | 0 0
0 0

a₀ ~~b₁~~ c₂

~~a c₁~~

~~c~~

O/p: b a c

2 | 3 | 2
2 | 0 | 2
1 | 0 | 0
0 | 0

a₀ b₁ c₂

cab

So on till 5 or n! - 1

⇒ Solution

```
public static void solution (String str){
  int n = str.length(). //length of string
  int f = factorial(); // factorial(n)

  for(int i=0; i<f; i++){ // n! - 1 cases
    StringBuilder sb = new StringBuilder(str);
    int temp = i; // i shouldn't be changed

    for(int div = n; div >= 1; div--){ //division loop
      int q = temp/div; // quotient
      int r = temp % div; // remainder
      // print char at remainder position
      System.out.print(sb.charAt(r)); //print(r)
      sb.deleteCharAt(r); // delete rem char

      temp = q; // change dividend to quotient
    }                  // 2→0, 0→0, 0→0 above
    System.out.println(); //Go to next line
  }
}
```