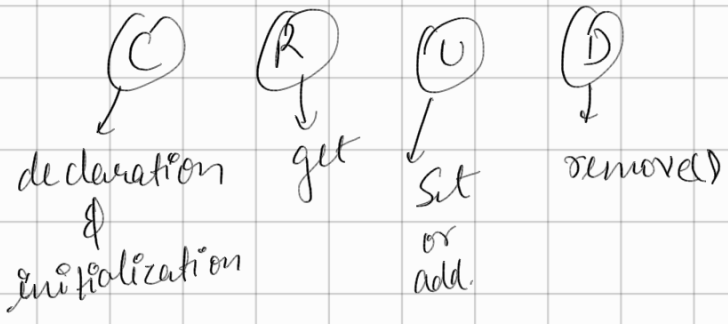
Topics to be covered today:

① String Builder
- initialization
- insert {at last, at given index}
- update {set charAt}
- delete {delete charAt}

② Array List
→ size()



(Q1) Remove primes
(Q2) Ring rotate <HW discussion>

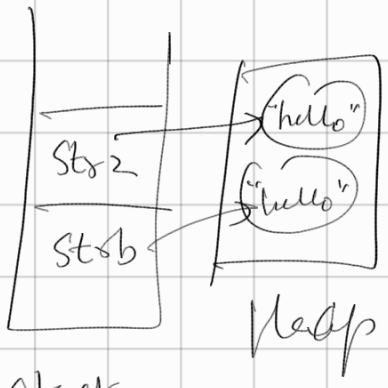
① StringBuilder (IS MUTABLE) unlike Strings.
here reference can also be changed
object can also be changed

mutability gives
- setCharAt(position, char); O(1)
- deleteCharAt(position); O(n)
- insert(position, char); O(n)
- append(char); O(1)

NO INTERNING IN STRINGBUILDER

Similar memory mapping as arrays.

StringBuilder strb = new SB("hello");
- - - - str2 = . . . ("hello");



setCharAt → CamelCase

CODE:

```
fun( string x, stringBuilder y) {
        x = x + "is changed";
        y. append (" is changed");
}
main ( ) {
    String s = "string";
    StringBuilder strb = new StringBuilder ("string builder");
    fun(str, strb);
    Syso( str);
    Syso( strb);
}
```

Output:

String
String Builder is changed

because string uses interning & changes reference to make new string in heap.

when fun() block ends the new String object will be destroyed whereas StringBuilder object is still same

Strings in C++ = String Builder in Java

(2) ARRAYLIST (similar to vector in C++ STL)

Array → Static, size cannot be changed

ArrayList → dynamic, generic

Declaring arraylist.

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

- `Integer` → data type in arraylist. not Int/int
- `ArrayList<>()` → Constructor

default.	8 primitive data types	Class equivalent/Wrapper class
0	int	Integer → ParseInt, Max, Min
NULL	char	Character
0.0	float	Float
0.0	double	Double
0	short	Short
false	boolean	Boolean
0	byte	Byte
0	long	Long
	STACK	HEAP.

wrapper class has some special extra properties

→ Here, always NULL is stored by default in reference

To Read: Auto boxing & Auto Unboxing

CODE: for ArrayList

```
Scanner sc = new Scanner(System.in);
int n = sc.nextInt();
ArrayList<Integer> arr = new ArrayList<>();

// Input.
for(int i=0; i<n; i++){
    Integer x = sc.nextInt();
    arr.add(x);          // adds at end
}

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

↑ random access like array

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

↑ index ← value

```
// Delete / remove
arr.remove(2);
```

↑ index

```
// Traverse after updation for unknown size
for(int i=0; i<arr.size(); i++){
    Syso(arr.get(i) + " ");
}
```

→ gives size of array list.

// To String()

Syso (arr);

↑

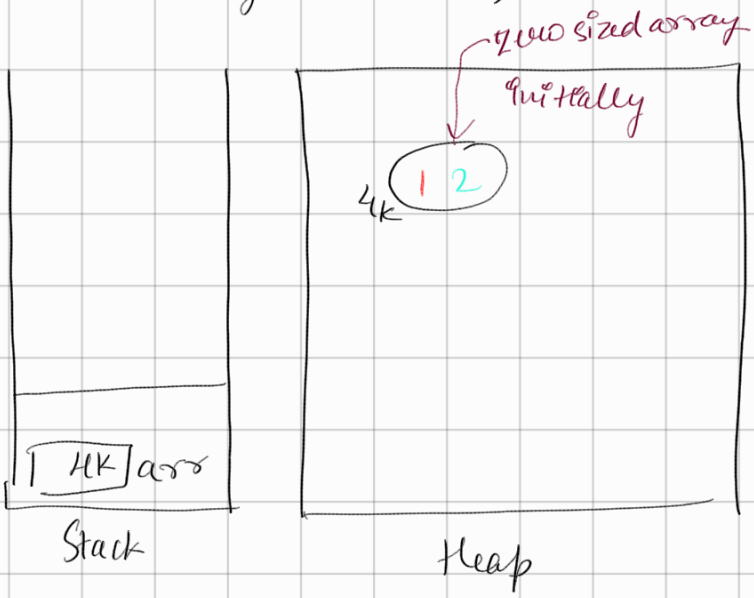
gives output as string

[0, 1, 2, . . . , n]

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

arr.add(1);

arr.add(2);



✡ for (int val: arr) {

Syso (val + " ");

}

} for each loop.

prints values in arr instead of index

Similar to

for (auto &i: arr)

cout << i ;

in C++.

① Remove primes.

Approach:

Traverse & check if Prime

if Prime

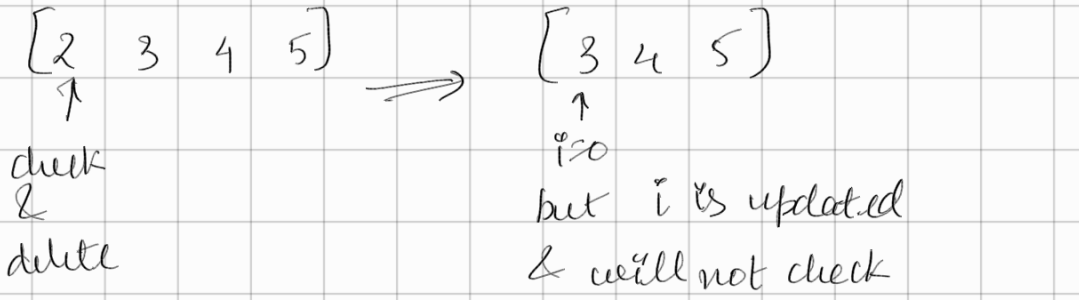
remove

else

move on

BUT

If we just move left to right & check & remove, then for a prime element



but i is updated & will not check for i=0 or arr[0]=3

So it will skip 3.

So, it fails