Date | 12 | 01 | 25
Page No.

⭐ Stacks and Queues - Short Notes

- Remembering stacks best example - plates in wedding,
  queue " " - ticket counter,

Stacks:

- `Stack<Integer> stack = new Stack<>();`
  - `stack`: variable stored in stack memory
  - `new Stack<>()`: this new object is created in Heap
- To add items = `stack.push(34);`
  To remove items = `stack.pop();` // Top one will be removed only
- Stacks is a class whereas Queue is an interface

***. Note:- we write `Queue<Integer> queue = new LinkedList<>();` instead of " " " " " " `Queue<>();` bcoz the nature of interfaces in Java. ∴ compilation error

- Queue is an interface & interfaces can't be instantiated directly (They only define a set of methods that class must implements)
  e.g. Queue provides → add(), remove(), offer(), peek() & poll() but doesn't provide any implementation for them.
- Whereas LL is a class that implements Queue/Deque interface 's methods
  e.g. PriorityQueue<>(); is a class
- Hence when we write in this manner we're using LL object but referring it to Queue interface, this is called polymorphism

- To add items → `queue.add(3);`
  To check top item → `queue.peek();`
  " remove item → `queue.remove();` → // bottom one will be removed

Teacher's Signature.....

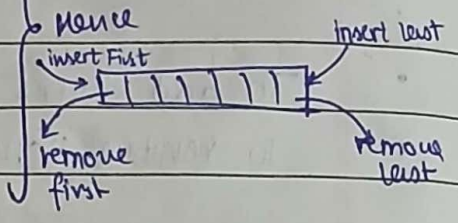
Date | Page No.

- **Deque** (Deck)

[Deque<Integer> deque = new ArrayDeque<>();]

** Similarly, here also ArrayDeque is a class that provides us to create object of methods inside deque interface & helps to implement.

∴ uses
deque.add(9); } by default
deque.addLast(9);
deque.addFirst(8);
deque.removeFirst();
" . " Last();



* **Custom Stack implementation**

public class CustomStack {
protected int[] data; → (protected access modifier is to use this in same package i.e. (Dynamic stack) Also not use default since it is not accessible to subclasses in other packages whereas protected is accessible) + (data is stored in array).

private static final int DEFAULT_SIZE = 10;

public CustomStack() {
this(DEFAULT_SIZE); → non parameterized const^r for default size
}

public CustomStack(int size) {
this.data = new int[size] → parameterized const^r for given size of array in which data is stored
}

int ptr = -1; → initialized pointer

Teacher's Signature.....

Date | Page No.

- To add items → public boolean push(int item){
  ptr++; → ptr increased
  store that item on ptr index → data[ptr] = item;
  if (isFull()){ → when ptr is at last index
  sout("Full"); ↓
  return false; i.e. ptr == data.length-1
  }return true;

- To remove → public int pop() ~~throws~~ ~~Exception~~ {
  if (isEmpty()){
  ~~throw new~~ stackException("can't pop from empty"); ∵ we cannot return -1 (∵ int) as it may be in the stack (valid no.)
  }return data[ptr]; ;

when pointer is at -1. make a file of stackException which extends Exception (in built)
super(message);

* In main fn. → psvm throws stackException ✓

* Dynamic stack → same as of custom stack, only diff. when insert item size functions as of arraylist concept.

public class DynamicStack extends CustomStack {
non-parameter constr of DynamicStack → called super();
size " " " " → " super(size);
@Override
public boolean push(int item){
* if (this.isFull()){ → no need to write isFull func. ∵ extended
int[] temp = new int[data.length * 2]; ← create a temp. array (doubled if full)

copy all prev. items → for(int i=0; i<data.length; i++){
temp[i] = data[i];
update the temp array → data = temp;
}
else{
return super.push(item); → directly.

Teacher's Signature.....

Date 15/01/24
Page No.

* Custom Queue Implementation

basic starting same as of custom stack
except here make end pointer & start with zero.
(therefore, here we'll first add then increase ptr till end)

```
int end = 0;
```

add items → public boolean insert (int item) {
if (isFull()) { → when end ptr at ~~ed~~ data.~~length~~
return false;
data[end] = item;
end++;
return true;

remove items → public int remove() throws Exception { ← same procedure of stack

- if isEmpty() throw exception
- else firstly get the item (returning (showing) purposes) → int removed = data[0]; ← remove from first
- then shift all the elements to left (actual removing)

```
for(int i=1; i<end; i++){
    data[i-1] = data[i]
}
end--;
return removed;
```

peek item (front) → public int front() throws Exception → when end ptr at 0;
if empty → ✓
return data[0]; ← "isFull" & end ptr out of bound pe rehta hai

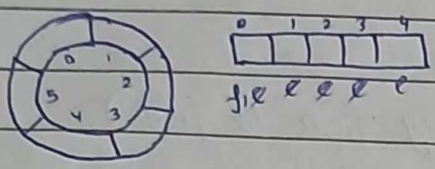
display → for i=0 till, end {
Sout (data[i] + "←");
}
before

Teacher's Signature.....

Date | Page No.

- Ely. as of Dynamic stack we can do for Dynamic Queue (only insert changes)* but it is extended from circular queue.

** Circular Queue implementation



concept: consider items only in two first (f) and end (e).

∴ we can add ∞ items in circular queue. ↓ coz only (f,e) will be consider

*Note:- after inserting till last size of array but to add in order use [end % size] → adding again.

e.g. size of array/queue is 5 + we are adding 6th item it will be stored at index 1 ∵ (6 % 5) = 1

+ isFull/isEmpty

Implementation:- basics, same as of custom Queue +

initialise → protected int end = 0

" " first = 0

add items → public boolean insert (int item) {

if full → return false

else → data[end] = item;

end++;

* end = end % data.length;

end_ptr++;

return true;

end = 0 → 1

1 % 5 = 1 ✓ → check ✓

remove → same as queue + if ✓ else {

int removed = data[front];

front++;

update front also → front = front % data.length;

end_ptr--;

return removed.

Teacher's Signature.....

peek (front) → same as Q.front()

```
public int front() throws Exception {
    if (isEmpty()) {
        - - - - -
    }
    return
```

display → if empty → print empty → return.

else → int i = front; (starting from the front)

```
do {
    Sout(data[i] + "->");
    i++;
    i %= data.length  ⟶ "circular condition" → socho dont memorise
} while (i != end);
Sout("End");
```

[* Q. How to check where stacks is used?
↳ sol → whenever you've to keep track of order while traversing]

Date	16 01 25
Page No.	

* Stacks imp. Questions

1. Implement Queue using Stacks

Q. using two stacks → functions used (push, peek, pop and empty)
make FIFO

~~Concept ⇒ • add items in stack first [2|1]~~

~~• put those in second stack till empty ∵ [1|2] ∵ using pop of first~~

~~then remove from this stack.~~

~~peek → • again put those back in first & return peak [2|1]~~

~~uske baad restore bhi to karoge na. :-~~

~~get that element i.e. removed → using second.pop~~

~~then till second is not empty put it in first.~~

(I easy ← peek → first follow above same steps)

Steps ⇒ • firstly declare both stacks)

• In myQueue method. → initialize both stacks.

• In Push " → push in first stack

• In Pop " → under condition that second is empty now while first is not empty put items from first → second return second.pop

• In peek " → " " " " " " (ditto of pop)

check ↓ • In empty " → return first empty && second empty.

2. Game of Two Stacks

Q. There will be Two stacks A & B from both the elements you've to take the max. no. of items till max Limit.

Steps ⇒ • Create two stack method which takes max. no & both stacks in which we'll call the actual stacks – 1.

Teacher's Signature.................................

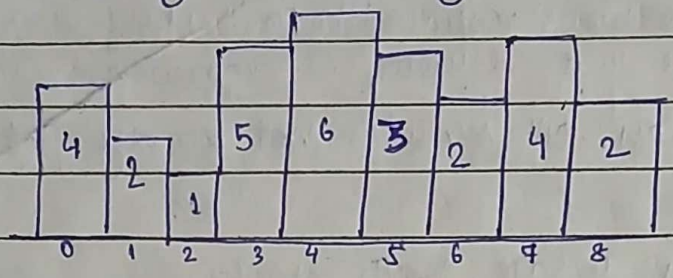
Date | Page No.

- The next method must contain max(x) no., both stacks, Sum (sum) till here & count.
- If sum > x then return count.
- else make two recursion call in a & b in which we'll copy the range and increase count & sum by stack[0] value at first in both ans1 & ans2.

3. Largest rectangle in Histogram

Leetcode Hard

Q.



Concept → • get the height of each indices
⇒ a = {4, 2, 1, 5, 6, 3, 2, 4, 2}.
• get the prev. smaller (any nearest item is less than that particular index then mention its index
⇒ ps = {-1, -1, -1, 2, 3, 2, 2, 6, 2}
if not found
• Ily get for next smaller
⇒ ns = {3, 3, 3, 4, 9, 9,
if not found
⇒ ns = {1, 2, 9, 5, 5, 6, 9, 8, 9}
• Then area of the ques will be :-

$$A = (ns[i] - ps[i] - 1) * a[i].$$

Steps ⇒ • Create an array of ps & call its items from a fn. Ily for ns.
• get maxArea using formula for individual indices

Teacher's Signature.....

Date			
Page No.			

- In prevSmaller method →
  - create an array ps & stack of integer
  - for 0 to a.length till while stack is not empty and top of stack (a[s.peek()]) is greater than a[i] s.pop();
  - if s.isEmpty → ps[i] = -1 , else ps[i] = s.peek();
  - then push that i after checking (always)

  & return ps.

- Ully for nextSmaller method() → everything except go from last to first and for if stack is Empty ns[i] = b.length; // obviously not 9 coz array size might change.

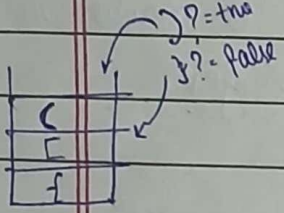
4. Valid Parenthesis

Q. {[()]} → true , {([)[} → false , {[(])]} → false

Concept → • Create a character type array.

∵ for each element in String s (s.toCharArray())
character

- if ch is opening brackets then push them in.
- for closing brackets if only if stack is empty or just prev character (using pop) is not its opening bracket then false
- In the end return stack.isEmpty(); → not true directly bcoz if s = `[` ∴ wrong Ans. ∴ if we're poping out we'll make sure all opening & closing are together successfully.
  `{`
  `(`



5. Minimum Insertions to balance a Parentheses String

Q. for every opening bracket it needs two closing brackets to make it balanced. e.g. s = "))())(" ∴ o/p = 3 → 1 "(" for first "))" and 2 "))" for last "(".

e.g. s = "(()))" ∴ → op: 1.

Teacher's Signature.....

Steps =>

6. Minimum Add to make Parenthesis Valid

eg. ())((→ 2

Steps => • make character stack

• for each character in string

• if ch =) → closing (if) top of stack is "(" opening equal

• ∴ pop no we to count

• else push

• else → push that ch first then check

• return stack.size(); → ∵ we've pushed only those items in stacks to make valid

∴ returning integer.