① Encapsulation ② Abstraction ③ Inheritance ④ Polymorphism

Stack
stack.push(10);
stack.pop();

Dynamic Stack -
public void push(int item) throws Exception {
if (this.size == this.data.length)
int[] temp = new int[2*this.data.length];
for (int i = 0; i < this.size; i++) {
temp[i] = this.data[i];
}
this.data = temp;
}
super.push(item); ← push of parent class.

Dynamic Queue -
public void enqueue(int item) throws Exception {
if (this.size == this.data.length) {
int[] temp = new int[2*this.data.length];
for (int i = 0; i < this.size; i++) {
temp[i] = this.data[(i+this.front)%this.data.length];
}
this.front = 0;
}
super.enqueue(item);
}

(size + front) % data.length
5 + 0 % 10
[5]

• Reverse a Stack

public void reverseStack(Stack stack, Stack helper, int idx) {
if (stack.isEmpty()) {
return;
}
int item = stack.pop();
reverseStack(stack, helper, idx+1);
helper.push(item);
if (idx == 0) {
while (!helper.isEmpty()) {
stack.push(helper.pop());
}
}
}

public void reverseQueue(DynamicQueue queue) {
if (queue.isEmpty()) {
return;
}
int item = queue.dequeue();
reverseQueue(queue);
queue.enqueue(item);
}

↳ 2 Stacks → implement the functionality of Queue

Queue {
enqueue()
dequeue()
front()
display()
size()
isEmpty()
}

enqueue(10) {
primary.push(10);
}

dequeue() {
while (!primary.size()) {
secondary.push(primary.pop());
}
int rv = primary.pop();
while (!secondary.isEmpty()) {
primary.push(secondary.pop());
}
return rv;
}

Queue Using 2 Stacks
- Enqueue Efficient
  - O(1) → enqueue
  - O(n) → dequeue
  - O(n) → front
  - size → O(1)
  - isEmpty → O(1)
  - display → O(n)
- Dequeue Efficient
  - enqueue (O(n)) ← push
  - dequeue (O(1))
  - front (O(1))
  - size → O(1)
  - isEmpty → O(1)
  - display → O(n)

Stack using 2 Queues
① Versions
- Push Efficient
- Pop Efficient

① ↳ Queue → Dequeue Efficient
② ↳ Stack → Push ___
③ ↳ ___ → Pop ___

Stacks → Arrays
↳ Queues
Queue → Arrays
↳ Stacks

LinkedList ↳ What Versions
• Break → 8:45 pm ↳ Resume

arr. {3, 8, 1, 5, 2, 6, 4}
next greater element of each element of array.
Ex 1 - O(n²)
↳ O(n).
public static int[] nextGreater(int[] arr) {
Stack<Integer> stack = new Stack<>();
stack.push(arr[0]);
for (int i = 1; i < arr.length; i++) {
while (!stack.isEmpty && arr[i] > stack.peek()) {
int rv = stack.pop();
System.out.println(rv + "->" + arr[i]);
}
stack.push(arr[i]);
}
while (!stack.isEmpty()) {
int rv = stack.pop();
System.out.println(rv + "=> -1");
}
}

43
Ans → 1+1+1+1+1
5+1+1+1+1+1+1
5+1+1+1+1+1+1
14

8>8 ×
5>1 ✓
5>8
2>5 ×
6>2 ✓
6>5 ✓
6>8 ×
4>6 ×

3->8
1->5
2->6
5->6

8>3

4=>-1
6=>-1
8=>-1

(2n) ops → O(n) ops

Check of previous elements

String str. "{[(a+b)*(c+d)]}"
balanced.
{, }, (,), [,] → case 1.
↳ push on stack.
other chars → case 3
else → case 4.
{ } x
{(){ } → case 2.
]}>
"}" → false.

if (cc == ']') {
if (stack.top() == '[') {
stack.pop();
} else {
return false;
}
}

if (stack.isEmpty()) {
return true.
} else {
return false;
}

• Questions ↳ Practical
Tomorrow → x off.
LinkedLists ↳ Monday