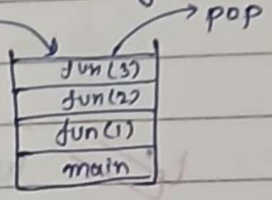
Date: 29/06/24
Page:

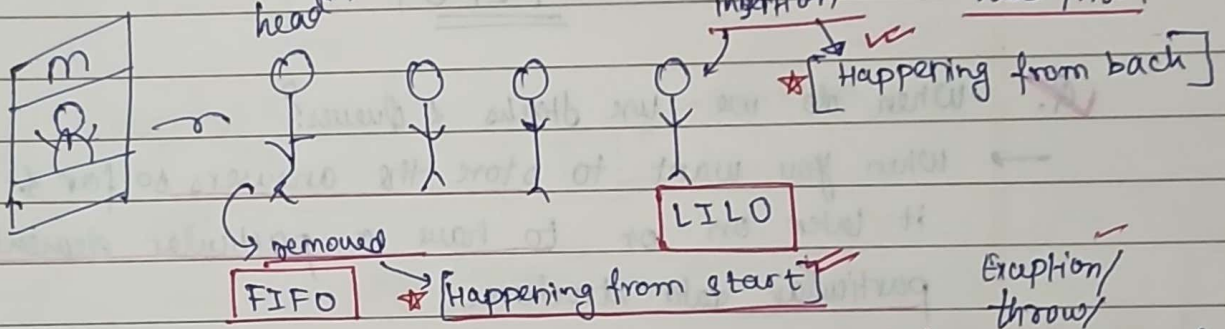
Stacks and Queues

Concept of FILO (first in last out) and LIFO - (Last in first out) → Don't memorize
already covered.
Example → ~~your~~ How you take plates which are stacked in wedding.

Since we can't remove from middle or break the rule of stacks (already knew).
∴ inserting an element in stack = push
∴ taking out" " from " = pop.



Similarly, Queues also follows the same principle in other way.



Note:- revise OOPs must!! abstract class, classes, use effectively, static keyword
Exception/throws/generics, throws keyword

Stack<Integer> stack = new Stack<>();
this variable in stack memory
new object created in Heap memory

to push items in stack → stack.push(34);
to pop " from " → stack.pop(); → ✓ since it removes last one only.
→ think & implement as arrays or any D.S. (no value needed)
if nothing in stack & we'll still remove it → error! not so cool

add item remove item ∴ complexity O(1). (constant)

Stack is a class whereas, Queue is an interface.
ctrl + click to see.

If you write Queue<Integer> queue = new Queue<Integer>()
∴ internally LL is the data type that implements, add, func. (you will get these) offer, remove, peek, etc.
∴ we use it as:-

```
Queue<Integer> queue = new LinkedList<>();
```

use queue.add(3); (from last) to add values in queue.
queue.peek(); → it just gets that item, but (i.e. first at line currently)
to remove → queue.remove(); (from first) do not remove that item.
↳ FIFO!

Q. When do we use stacks & Queues?
→ When you want to store the answers so far & to use it later on or to have a particular element in particular data structures
eg. Binary Tree traversals
eg. Recursion programs into iterations

Deque → (deck)

```
Deque<Integer> deque = new ArrayDeque<>();
```

it is a class that provides us to create object of methods inside deque interface & helps to implements.

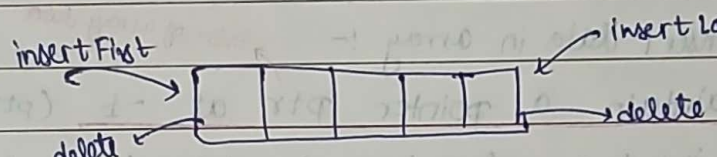
Note:- ctrl + click to know definitions & details.

Date : / /
Page :

- ~~Queue is faster that stack~~
- Class is faster than Stack when used as stack & faster than LL when used as queue.
- ∴ ArrayDeque is faster than LL<>();

Imp. points => You can add or remove from both sides.
null element not allowed.
no capacity restriction (not as in stacks.)

∴ uses →
```
deque.add(5);
deque.addLast(5);
deque.addFirst(5);
deque.removeFirst();
:
etc.
```



[fig: deque]

Doubt

Custom Stack

final → coz we'll not change it & conventionally we use capital letters for it.

this keyword uses :- To reference the object of class data type
To call it as a constructor. (call;)

public class CustomStack {

We know that it has array inside it.

∴ protected int[] data;

private static final int DEFAULT_SIZE=10; → fill defined length, it gets full it throws an error.

∴ can't change directly

for CustomStack it will be same

we will not change it ever

- When obj. of custom stack is created i.e.
  CustomStack stack = new CustomStack();
  ∴ new object will be created in heap i.e. empty array with default size (not arraylist) initially.

for manging size → constructor → public CustomStack(int size){
by creating array this.data = new int[size];
}

if nothing is passed use default size → public " (){
this.(DEFAULT-SIZE);
}
(call)
if you pass your own size then it will not will be called

How to insert data in array :- (index of array data)
maintain a pointer ptr at -1 (ptr = -1)
whenever → to insert a value increase pointer by 1 & enter
ly, to remove a value :- (also called push in stack)
decrease the value of ptr by 1
add another value at its place :-
again increase the value of ptr by 1 & put new value.

If array gets full → throws Exception
int ptr = -1;

Inserting :- public boolean push(int item){
(to insert a value inc. ptr. by 1) → ptr++; (ptr increased)
data[ptr] = item; (put value)
return true;
if(isFull()){ (apply same pointer kind of.)
System.out("stack is full"); (if array gets full)
return false;
}

or directly write as:- ptr == data.length-1
(equals to size)

Date : / /
Page :

✓ create isFull → ✓ private boolean isFull() {

can be done directly

now if you do ptr++ it will go out of bound

✓ return ptr == data.length-1 ; //ptr at last index

✓ [lly create isEmpty → when? → ptr == -1 → directly also.] → private boolean isEmpty () {

return ptr == -1 ;

}

cool! dedicated custom exception Stack

✓ lly create pop as push → public int pop() throws Exception {

✓ if (isEmpty()) {

✓ throw new Exception ("Can't pop from empty stack");

}

✓ int removed = data[ptr];

ptr = ptr-1 ← ptr--;

removing item as data at ptr. ←

✓ return removed ;

↳ ∵ int type.

directly as → return data[ptr--];

we then decrement

pehle data[ptr] mein value milega then ptr-1 hoga.

✓ To peek (what is at the top?) → public int peek() throws Exception {

the value which was added in the last

✓ return data[data.length-1] ;

ptr

✓ if (isEmpty()) {

throw new Exception("Can't peek from empty stack");

Stack Exception Handling

{ (after making new Java class)

public class StackException extends Exception {

public StackException(String message) {

accessing parent msgs in previous Java class ← super(message);

}

(new Java class) class

To run → public, StackMain {

psvm { → throws StackException

CustomStack stack = new C---('');

If we didn't provide any then then DEFAULTSIZE i.e. 10

If we provide size then size = 5 e.g.

Now use this to add items { stack.push(--);

shows error ∴ throws exception ∴ add it in psvm

sout(stack.pop());

(ctrl shortcut keys to access easily, DEBUG etc.)

Put Debug pointer & debug step by step for better clarity.

General Doubt → Original Stack never gets full then why this so?

⇒ in ArrayList → whenever an array gets full it creates new array with double size, copy all items & add.

∴ We will now create Dynamic Stack, it will never be full.

Date : / /
Page :

* Custom Dynamic Stack

→ since we've already made a lot of things in custom stack ∴ extend it.

∴ public class DynamicStack extends CustomStack {

✓ constructor ← public DynamicStack() {
checks which constructor in parent class do not take arguments. ← super(); // it will CustomStack() (DEFAULT_SIZE)
Basic OOPs }

✓ otherwise this also public DynamicStack(int size) {
super(size); // it will call CustomStack(int size)
}

✓ Here in dynamic all will be same of custom stack except for push.

• Now if we've to use push of Dynamic Stack not of custom stack use → @Override (Basics of OOP → revise broooo!)

∴ click alt + insert + select Override → push method.

∴ public boolean ... {
if (this.isFull()) { → make it public from there to use.
// double the array size
arraylist functions { ∴ int[] temp = new int[data.length * 2];
// copy all prev. items in new data
∴ for (int i = 0; i < data.length; i++) {
temp[i] = data[i];
}
Now → data = temp;
}
∴ At this point we know that array is not full
if statement closed
Now insert items → return super.push(item); ← custom stack
}

Now in StackMain Class change CustomStack object to Dynamic stack & push more element it will allow :→ DynamicStack stack = new Dyn... OOPs!!
but can also mention as → CustomStack stack = new DynamicStack(5); ← (parent can be used as type also)
type of all access you can get — what access they can get

∴ Concept of full data → data[2, 9, 1, 8, 7] (previous) → garbage collector

double its size ∴ temp = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

when data() = temp() then temp() = [2, 9, 1, 8, 7, 0, 0, 0, 0, 0]

data = temp then data = [2, 9, 1, 8, 7, 0, 0, 0, 0, 0]

garbage collector

∵ temp is defined/declared on in If condition

again

now new item will be added in data[]

& it repeats.

→ Complexity Analysis → whenisFull condition → O(n)

but on average it will be O(1) ∵ arraylist.

"INDIA WON THE T20 WORLD CUP-2024!!!"

✳ Custom Queue

basic Queue approach → insert from last & remove from first.

(M-1) → easy take 2 pointers & solve.

(M-2) → we only 1 pointer! ∴ Difficult ∵ interview based

public class CustomQueue {

private int[] data;

private static final int DEFAULTSIZE = 10;

int end = 0; [Here we are adding then increasing]

copy paste constructor of DEFAULTsize & size from Stacks & name CustomQueue ally for

isFull & isEmpty → return end == 0;

make return end == data.length

in ptr = -1 → we were first ∴ increasing it & adding the item.

[pointer pointing towards end]

∴ LIFO / FIFO

"end-ptr"

++end → first increment the value of end then assign the item.

end++ → assign it then increase.

Date : / /

Page :

✓ inserting an item → public boolean insert (int item) {

O(1)

↳ adding item ∵ we insert at end in Queue

```
if (isFull()) {
    return false;
}
data[end] = item;
end++;
return true;
```

✓ removing an item → data = [3, 9, 4, 18, 77]

removed

shifted it left by 1

[9, 4, 18, 17, 0]

O(n)

↳ removing n items

(end--) ∵ 1 item is removed

```
public int remove() throws Exception {
    if (isEmpty()) {
        throw new Exception("Queue is Empty");
    }
    int removed = data[0];
    for (int i=1; i<end; i++) {
        data[i-1] = data[i];
    }
    end--;
    return removed;
}
```

shifting items to left

- ~~displaying front~~ → empty Queue

```
public int front() throws Exception {
    if (isEmpty()) {
        throw new Exception("Queue is empty");
    }
    return data[0];
}
```

✓ displaying Queue →

```
public void display() {
    for (int i=0; i<end; i++) {
        sout(data[i] + "←");
    }
    sout("~~END~~");
}
```

START

Make Queue main as new Java class.

public

make object of custom queue & pass size

queue.insert(3);

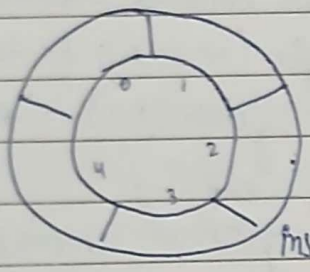
:

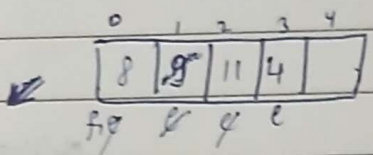
queue.display();

System.out.println(queue.remove());

queue.display(); → first item will get removed but O(n) time to remove it (not liked ∴ circular queue)

* Circular Queue Implementation





insert
- firstly → first & end will be at 0th index
- then after we add element ~~element~~ end increase by 1. → treat as circular

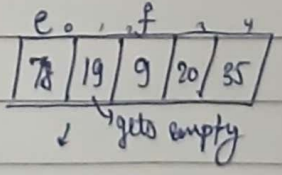
remove
- increase the value of front by 1.
- you'll notice that the element is not removed. But we only consider items in b/w first & end ∴ all the other elements will be overridden when we add again.

Now after inserting till last size of array to add in order for circular Queue ∴ end must be at 0th index after 4th

∴ end % size eg 5 % 5 = 0

any size [CONCEPT]

Hence even if the array becomes like this



You'll read it as 9 → 20 → 35 → 78

Date : / /
Page :

now if end increases ie. 6 ∴ 6 % 5 = 1 ↳ index 1
size of arr. ∴ end at 1

- implementation ⇒ copy past from Custom Queue about attributes, () constr., (size) constr., isFull, isEmpty. ✓
  use protected int[] data;
  + add protected int front = 0; attribute as 0 with end as protected.
  + private int size = 0;
  in isFull method → return size == data.length;
  ∵ end will come back at 0 + stuff.
  in isEmpty method → return size == 0;
- to insert ⇒ public boolean insert (int item) {
  if (isFull()) {
  return False;
  }
  data.[end] = item;
  end ++;
  already discussed. ← end = end % data.length;
  size ++;
  return true;
  }
- to remove ⇒ public int remove() throws Exception {
  if - - -
  int removed = data[front++];
  ntlb data[front];
  front++;
  front = front % data.length
  size--;
  return removed;
- front ⇒ copy paste + edit return data[front];
  ↳ ∵ 0 may contain end

- dly display copy paste while run from i = front till end in for loop.

- dly make main file & make object of Circular Queue.
  insert values → display → (remove) → insert → display.
  Sout

- display ⇒

```
public void display(){
    if (isEmpty()) {
        Sout ("Empty");
        return;
    }
    int i = front;
    do{
        Sout (data[i] + " → ");
        i++;
        i %= data.length;
    } while (i != end);
    Sout ("End");
}
```

start ⟶

might need to 0, 1 (till reaches end) ⟶