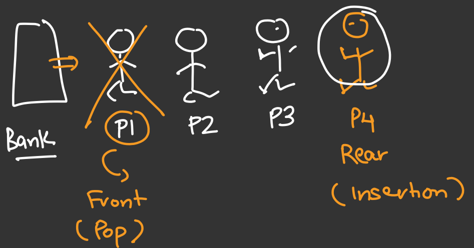
Queues



Stack
↳ Array
↳ Linked list

Support @ scaler com

get the recordings (whole Intermediate Batch) for Mar 22 - Java - beginner - monday

Queue
→ Array
→ Linked list
~~Queue~~

Queue → FIFO

Methods —
- Enque ( ) / push( )
- Deque ( ) / Pop( )
- Rear ( ) { Return last element }
- Front ( ) { " first " }
- size( )
- empty( )

Queue <Integer> q = new Queue

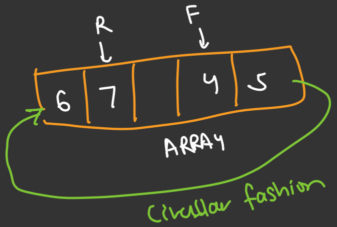
q.

Queue

" Fixed Size Array "

limit

↳ size is fixed.



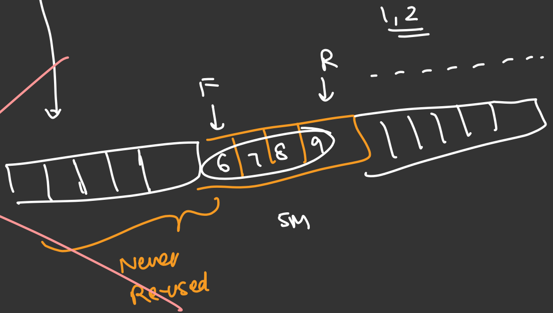
CS = 5 MS = 5

~~Dynamic Array / Arraylist~~

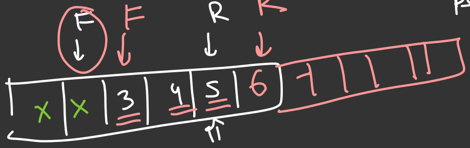
~~↓~~

~~memory is inc ⇑~~

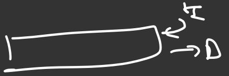
~~⇒ crash your applications.~~



Arraylist



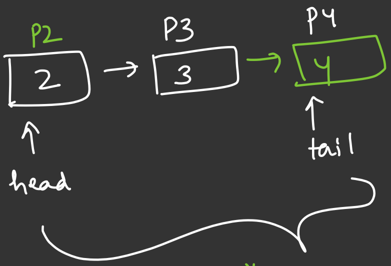
~~pop_b~~
pop-back
push-back



'Queue using linked list'

Adv

→ Dynamic
→ O(1)
→ no wastage of Memory

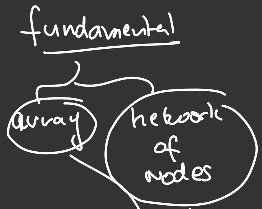


once for all

```
Class Queue {
    Node head
    node tail

    insert() {
        = tail
    }
    pop() {
        = head
    }
}
```

hide the low level details

fundamental



- stack
- queue
- tree
- graph
- heap
- hashtable

Implementing a Queue using (LL)

head tail

class Queue {

Data members ⇒

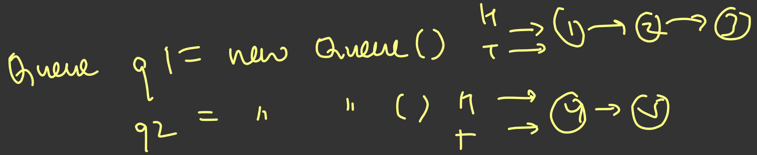
```
Node head;
Node tail,
int size,
```

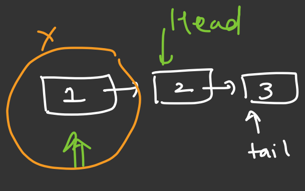
PRIVATE

Member functions (PUBLIC)

```
Queue(){
    head = tail = NULL
    size = 0
}
enqueue(int data){
    if(head == NULL){
        head = tail = new node(data),
    }
    else {
        tail.next = new Node(data),
        tail = tall next,
    }
    size ++ ;
}
```

```
class Node {
    int data;
    Node next;
    Node (d){
        data = d
    } next = null
}
```





No-one is refering (GC)

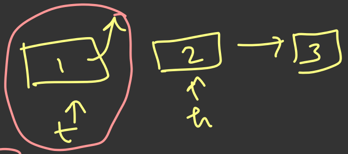
[Java]

```
deque ( ) {
    if (head != NULL) {
        head = head.next ;
        size --
    }
}

getSize() {
    ~~re~~ return size;
}

empty() {
    return size == 0 ;
}

front() {
    return head.data;
}
};
```

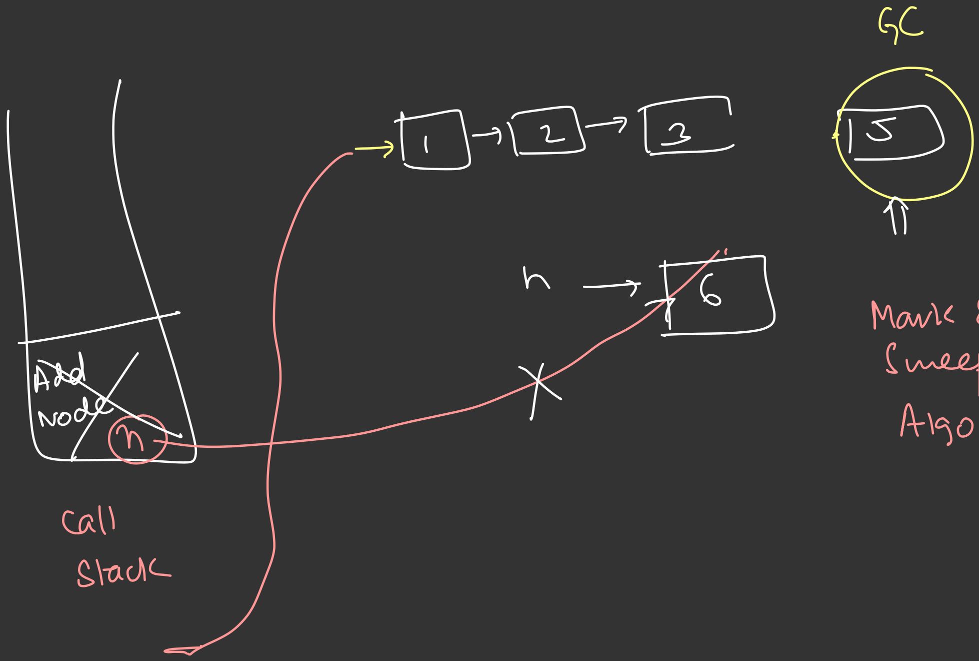
C++

```
Node temp = head;
    head = head.next
    temp.next = null
    delete temp
```

Programmer's task to free d



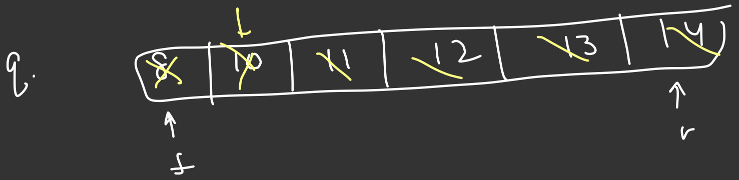
```
rear() {
    return tail.data;
}
```

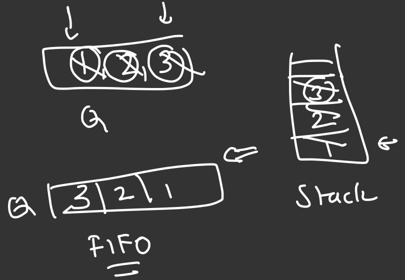


```
method - AddNode() {
    Node n = new Node(5);
=>  n = new Node(6)
    return;
}
```

Mark & Sweep Algo ⇒ Active References

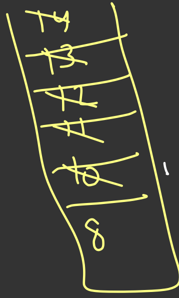
Q. Given a queue, Reverse the data in queue





Queue = { 14, 13, 12, 11, 10, 8 }

Stack



Stack
(LIFO)

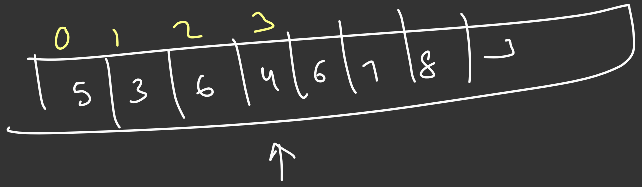
```
reverse ( Queue q) {
    Stack s = new Stack;

Q → S   ①   while( !q empty() ){
                s. push (q. front());
                q. deque();
```

3

$S \rightarrow Q$

②
```
while (!s.empty()){
        q.enque(s.top()),
        s.pop();
}
```

}



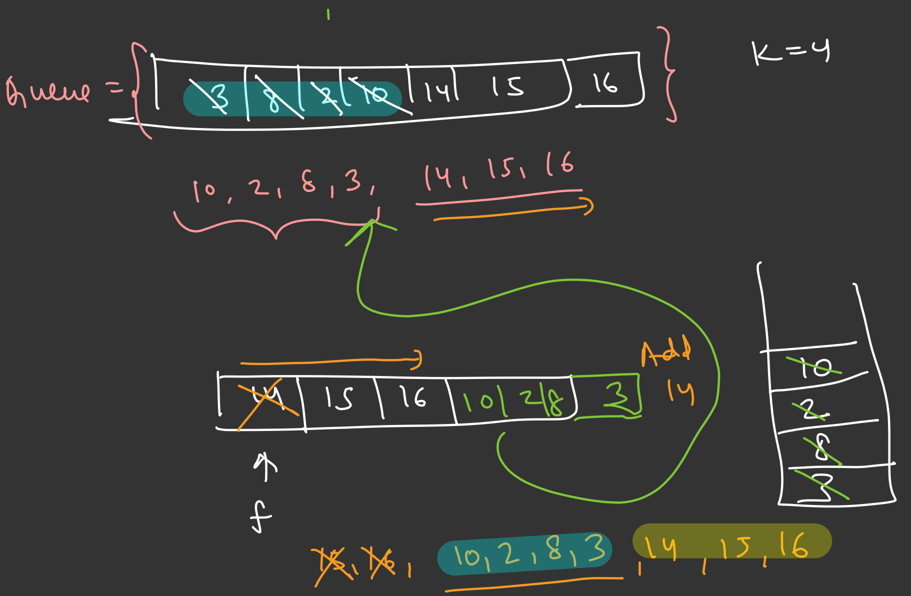
arr[3]

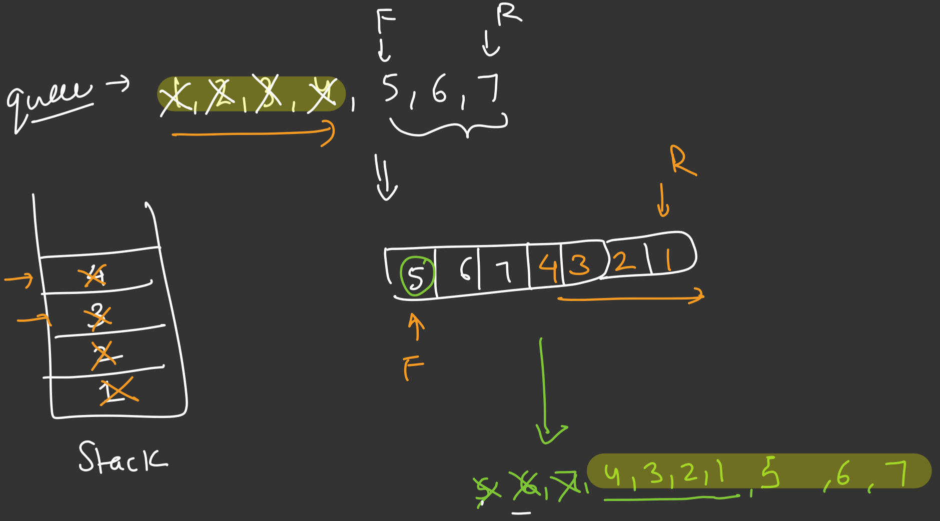
$5 \rightarrow 3 \rightarrow 6 \rightarrow \boxed{4} \rightarrow 6 \rightarrow 7 \rightarrow 8$

Access

- → Docs
- → Java Collection fw
- → Lang Specific

Q Given a Queue, Reverse first K element of Queue.





1 extra Stack of Size K for rev.

→ O(N) Time

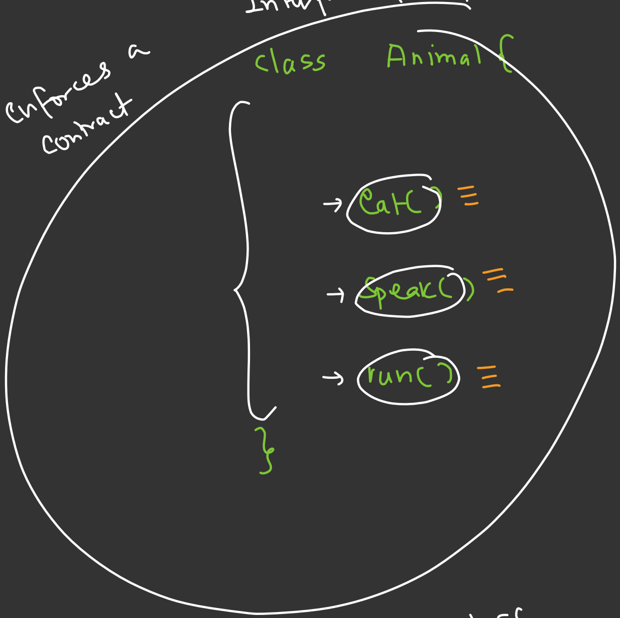
→ O(K) Space

10:35

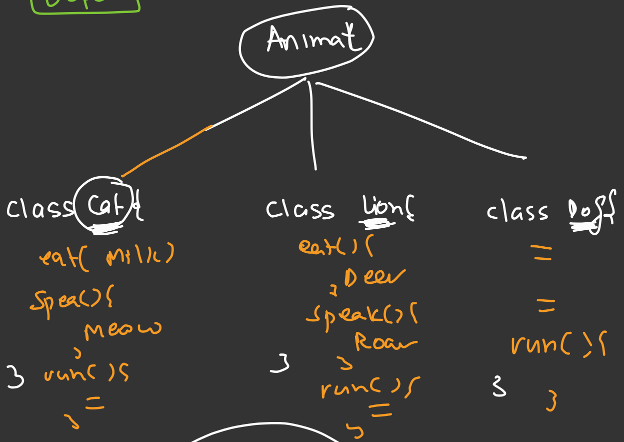
Break :)

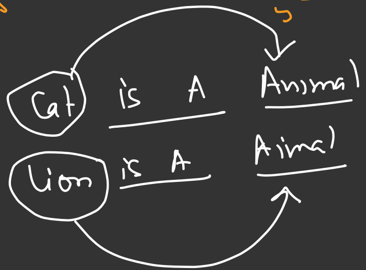
Interface (Abstract classes) Oops

Interface (Blueprint)



Abstract class





Animal a = new ~~Animal()~~

= new cat().

Java collections

(Interface)
class Queue {

peek(){ // front() get the front element
X
}
offer(){
X
}
poll (){
X
}
add(){ // add to rear
X
}

Abstract ~~List~~ class

linkedlist(){

add()

remove(){

}

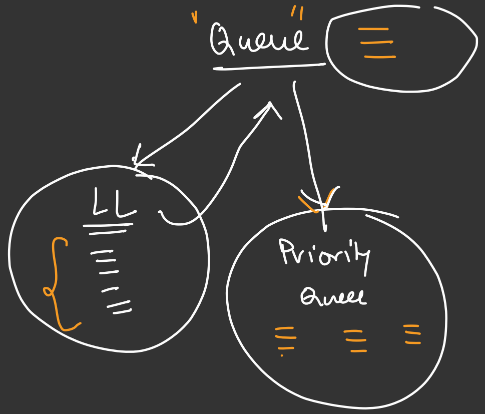
Code

Queue q = new linkedlist();

(abstract class)

LL implements Queue

PQ implements Queue



Fun Problem

generate K^{th} number by using only digits $(\underline{1}, \underline{2}, \underline{3})$

$\boxed{K = 13}$

$K = 5$ output $\rightarrow \boxed{12}$

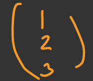
$$\begin{array}{ccccccccccccc}
[\,1, & 2, & 3,\,] & [\,11, & \boxed{12}, & 13\,] & [\,21, & 22, & 23\,] & [\,31, & 32, & 33\,] & [\,111, \; 112, \; 113\,] \\
\boxed{0} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12
\end{array}$$

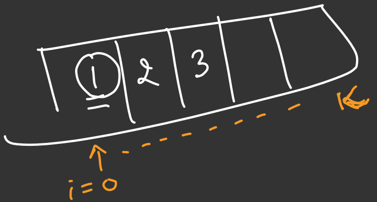
1
2
3
11
12
13
~~14~~

$\Rightarrow \underset{1}{\underline{1}}, \underset{2}{\underline{2}}, \underset{3}{\underline{3}}, \cancel{4}, \cancel{5}, \cancel{6}, \cancel{7}, \cancel{8}, \cancel{9}, \cancel{10}, \underset{4}{\underline{11}}, \underset{5}{\underline{12}}, - - -$

[1, 2, 3] (11 12 13) (21, 22, 23) (31, 32, 33) (111, 112, 113)....

(121, 122, 123)...





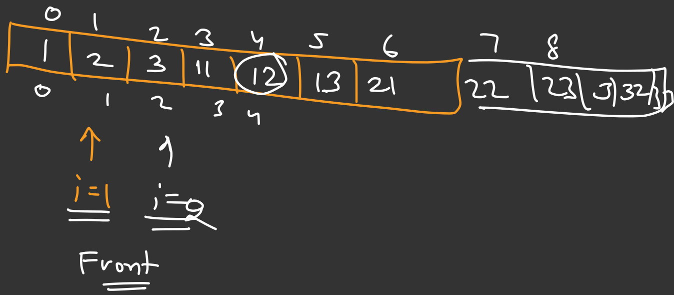
List<String> l =

l add (1)

l.add ("2")

l add ("3")

K = 10



6 < 10

i=0

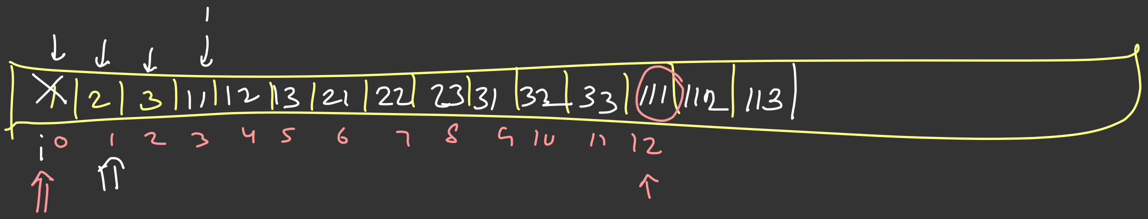
```
while ( l size() < k ) {
    {
        l.add( s[i] + "1" );
        l add (s[i] + "2");
        l add (s[i] + " 3");
    }
    i++;  →
}
```

Stop the loop if list has k elements

$\boxed{+3}$

ans = l[K-1];

12


1, 2, 3
X | 2 | 3 | 11 | 12 | 13 | 21 | 22 | 23 | 31 | 32 | 33 | 111 | 112 | 113
i 0 1 2 3 4 5 6 7 8 9 10 11 12

l.size()	K=13	
3	3<13	
i=0 6	6<13	
i=1 9	9<13	
i=2 12	12<13	
i=3 15	15 ≮ 13	Stop

Array~~Li~~List<int> l = new ArrayList(); ✓

Queue

```
Queue q

q add(1)
q add(2)
q add(3)          cnt = 3

while(          ){

    f = q front();
    q deque();
    q add( f + '1')      cnt += 1  -  if(cnt == k) { f+1 }
    q. add( f + '2')      cnt += 1  -  if(cnt == k) { f+2 }
    q. add( f + '3')      cnt += 1     if(cnt == k) { f+3 }
}
```

ans