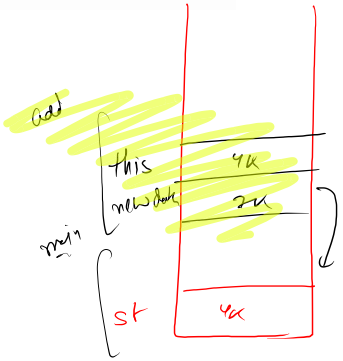
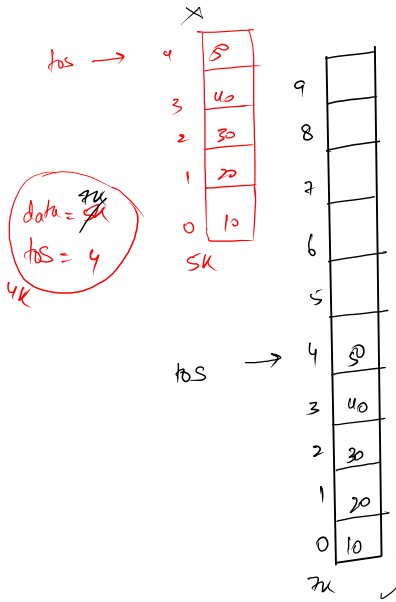
OOP → Implementation

```java
// change the code of this function according to question
void push(int val) {
  if (tos == data.length - 1) {
    System.out.println("Stack overflow");
  } else {
    tos++;
    data[tos] = val;
  }
}
```

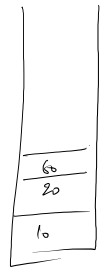
push 60





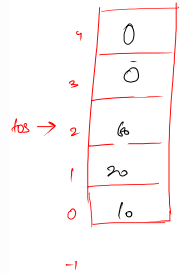
- push 10
- push 20
- push 30
- push 40
- top → 40
- pop → 40
- push 50
- pop → 50
- pop → 30
- top → 20
- push 60
- Display → 60 20 10
- Size

Imagine

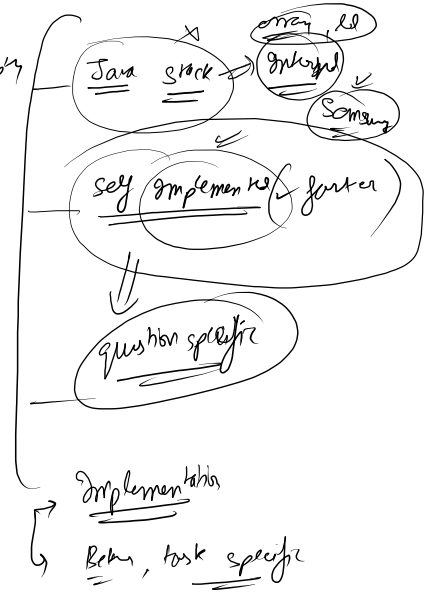



```java
int size() {
  return tos+1;
}

void display() {
  // write ur code here
}

void push(int val) {
  if( this.size() == data.length ){
      System.out.println("Stack overflow");
      return;
  }

  tos++;
  data[tos] = val;
}

int pop() {
  if(this.size() == 0){
      System.out.println("Stack underflow");
      return -1;
  }

  int val = data[tos];
  data[tos] = 0;
  tos--;
  return val;
}

int top() {
  if(this.size() == 0){
      System.out.println("Stack underflow");
      return -1;
  }

  return data[tos];
}
```

Implement



poll
isempty



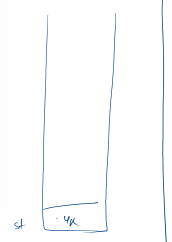
Stack → LIFO

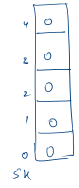
OPERATIONS → Push ()
Pop ()
Peek ()
Size ()

```java
public static class CustomStack {
  int[] data;
  int tos;

  public CustomStack(int cap) {
    data = new int[cap];
    tos = -1;
  }

  int size() {
    // write ur code here
  }

  void display() {
    // write ur code here
  }

  void push(int val) {
    // write ur code here
  }

  int pop() {
    // write ur code here
  }

  int top() {
    // write ur code here
  }
}
```

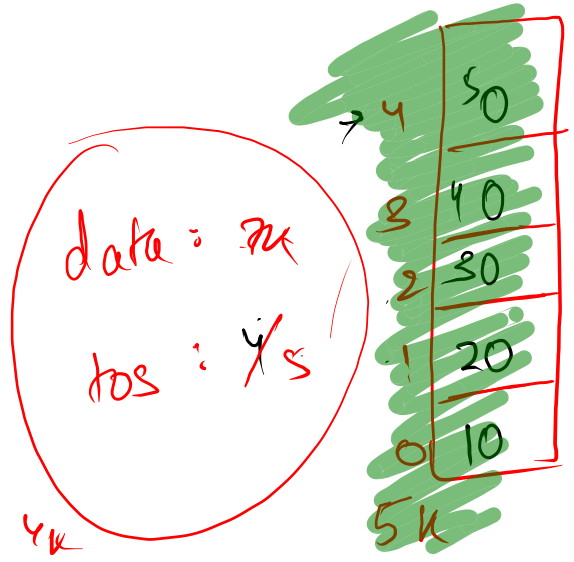
CustomStack st = new CustomStack(n);

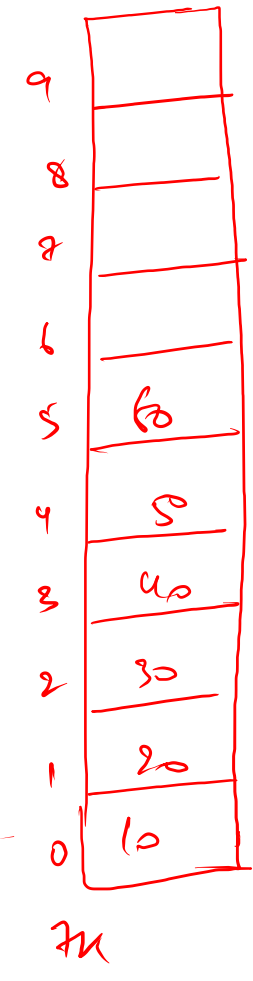
Fixed Size → Dynamic





```java
void push(int val) {
  if (tos == data.length - 1) {
    int newData[] = new int[this.data.length * 2];

    for(int i = 0 ; i < data.length ; i++){
        newData[i] = data[i];
    }

    this.data = newData;

    push(val);
  } else {
    tos++;
    data[tos] = val;
  }
}
```





✓ push 10

✓ = 20

✓ = 30

✓ = 40

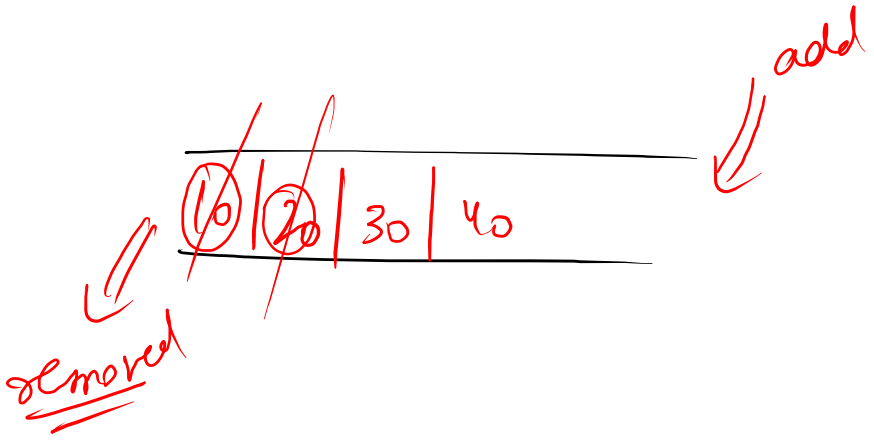
✓ = 50

✓ = 60

Queue → Discipline → FIFO

Element about to processed

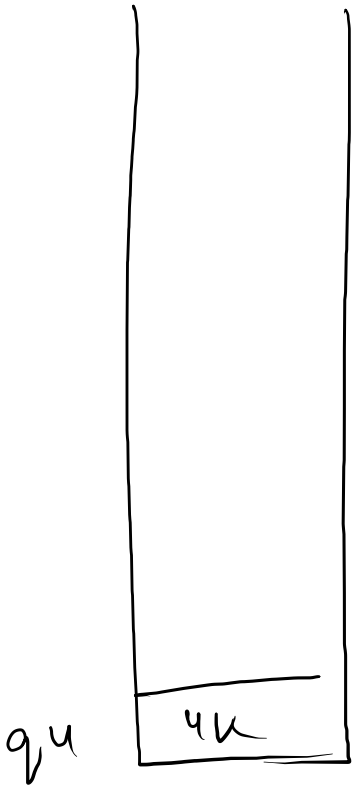
Operations → remove, add, peek, size

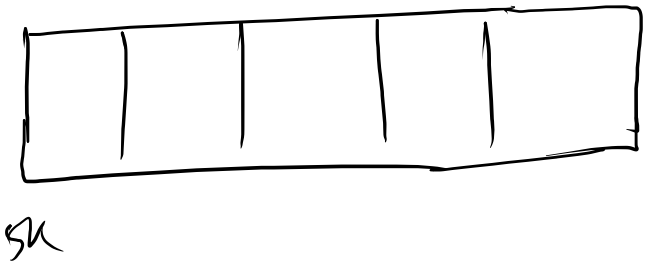


✓add 10
✓add 20
✓add 30
✓add 40
10 ← peek()
10 ← remove()
20 ← peek
20 ← remove()
2 ← Size

```java
public static class CustomQueue {
  int[] data;
  int front;
  int size;

  public CustomQueue(int cap) {
    data = new int[cap];
    front = 0;
    size = 0;
  }

  int size() {
    // write ur code here
  }

  void display() {
    // write ur code here
  }

  void add(int val) {
    // write ur code here
  }

  int remove() {
    // write ur code here
  }

  int peek() {
    // write ur code here
  }
}
```

```java
int n = Integer.parseInt(br.readLine());
CustomQueue qu = new CustomQueue(n);
```





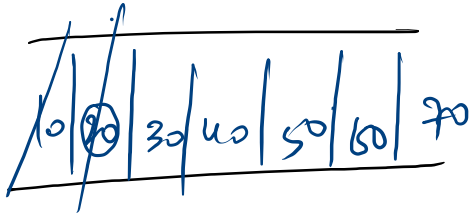
Operations

- ✓ add 10
- ✓ add 20
- ✓ add 30
- ✓ add 40
- ✓ add 50
- ✓ remove → 10
- ✓ add 60
- ✓ add 80 → Overflow
- ✓ remove → 20
- ✓ add 70

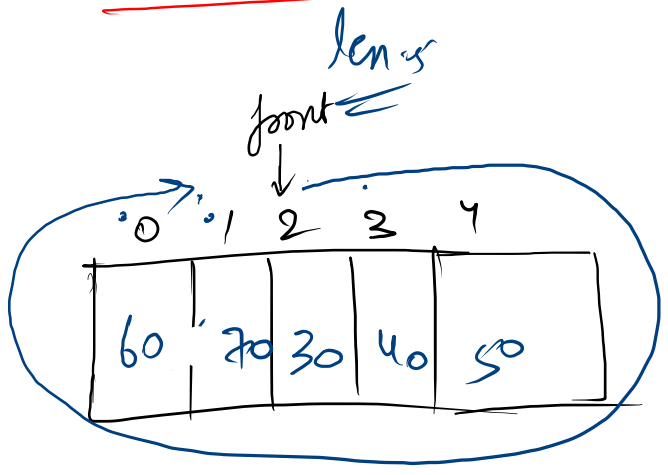
Display → 30 40 50 60 70

peek() → 30

Abstract view



Implementation View



front = 0 1 2

Size = 0 1 2 3 4 5

starting pos

Logic

add(val)

idx ⇒ (front + size) % len

data[idx] = val

size ++;

remove()

val : data[front]

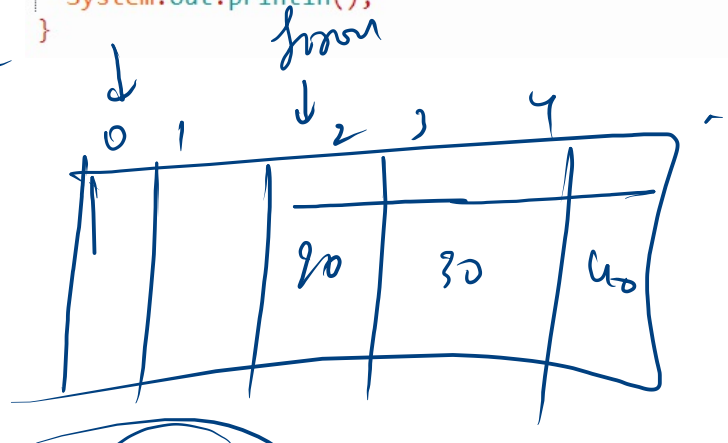
data[front] = 0

front ++;

size --;

return val

```
void display() {
    int n = data.length;
    for(int idx = 0 ; idx < this.size ; idx++){
        System.out.print(data[(this.front+idx) % n]+" ");
    }
    System.out.println();
}
```

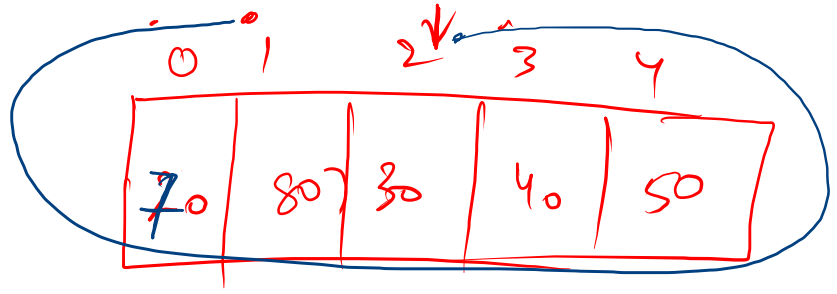


f : 2

s : 2

f + s → Empty Space

- ✓ add 10
- ✓ add 20
- ✓ add 30
- ✓ add 40
- ✓ add 50
- ✓ add 60 → Of
- ✓ remove() → 10
- ✓ // → 20
- ✓ add 70
- ✓ add 80
- remove()
- //
- //
- //
- //



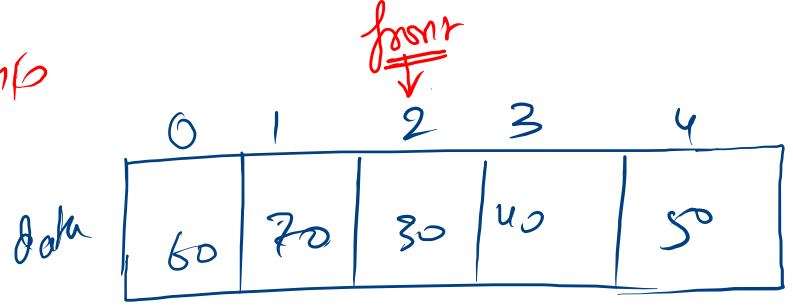
front = 0 X 2

size = 0 X 2 3 4 5 4 3 4 5

idx	(front + idx) % len	
0	2	→ 30
1	3	→ 40
2	4	→ 50
3	0	→ 70
4	1	→ 80

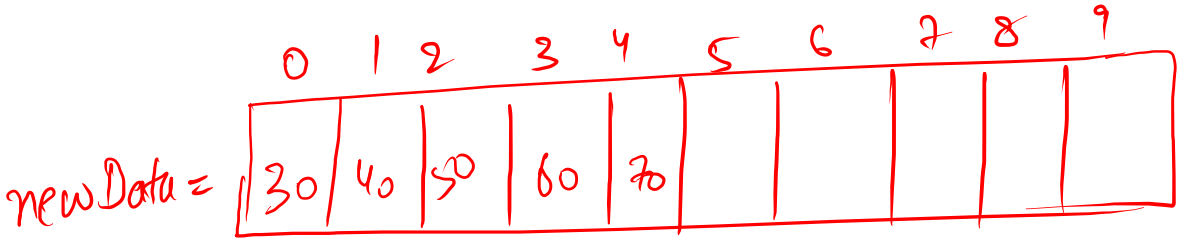
add 10 ✓
add 20 ✓
add 30 ✓
add 40 ✓
add 50 ✓
remove() ✓
remove() ✓
add 60 ✓
add 70 ✓
add 80

Queue → FIFO

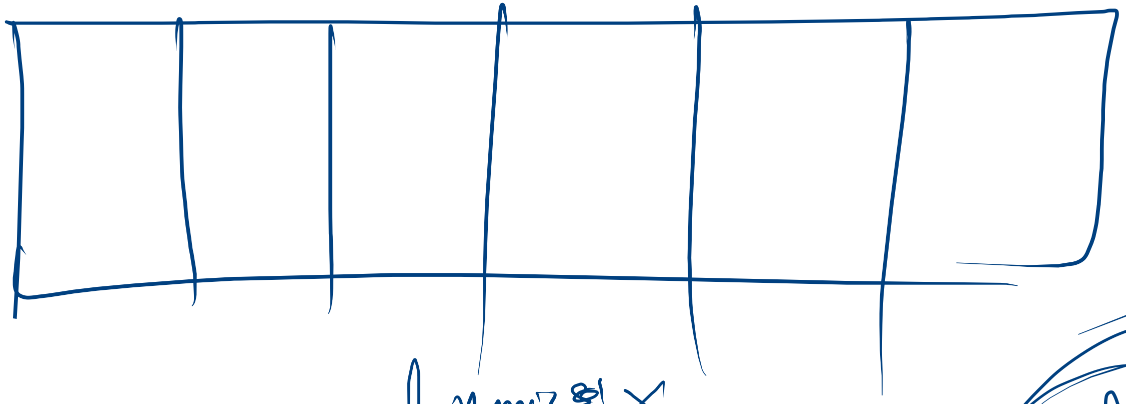


front : ~~2~~0

Size = 5



Todo



Queue

~~Circular~~ ×

~~space~~ ×

time

Cap → ~~dynamic~~ × ~~unlimited~~ size

illusion

static size → no

dynamic size → automatically grow

array → 4

~~stack~~

1 array (new cap)

2 copy

3 make size


Normal Stack
Easy
10 Auth 0 Public Sol 14
Dynamic Stack
Easy
10 Auth 0 Public Sol 15
Queues - Introduction And Usage
10 Auth 0 Public Sol 16
Normal Queue
Easy
10 Auth 0 Public Sol 17
Dynamic Queue
Easy
10 Auth 0 Public Sol 18

</>	Queue To Stack Adapter - Push Efficient	Easy	10	✓Auth	0	☐Public	✓Sol	19
</>	Queue To Stack Adapter - Pop Efficient	Easy	10	✓Auth	0	☐Public	✓Sol	20
</>	Stack To Queue Adapter - Add Efficient	Easy	10	✓Auth	0	☐Public	✓Sol	21
</>	Stack To Queue Adapter - Remove Efficient	Easy	10	✓Auth	0	☐Public	✓Sol	22
</>	Two Stacks In An Array	Easy	10	✓Auth	0	☐Public	✓Sol	23