

# LinkedList Stack & Queue

By Malapchai Chaisihat (P'1), CS28



# Outline

- LinkedList Revisited
- Stack
  - What is Stack?
  - Method Concepts of Stack
  - Java Stack Implementation
- Queue
  - What is Queue?
  - Method Concepts of Queue
  - Java Queue Implementation

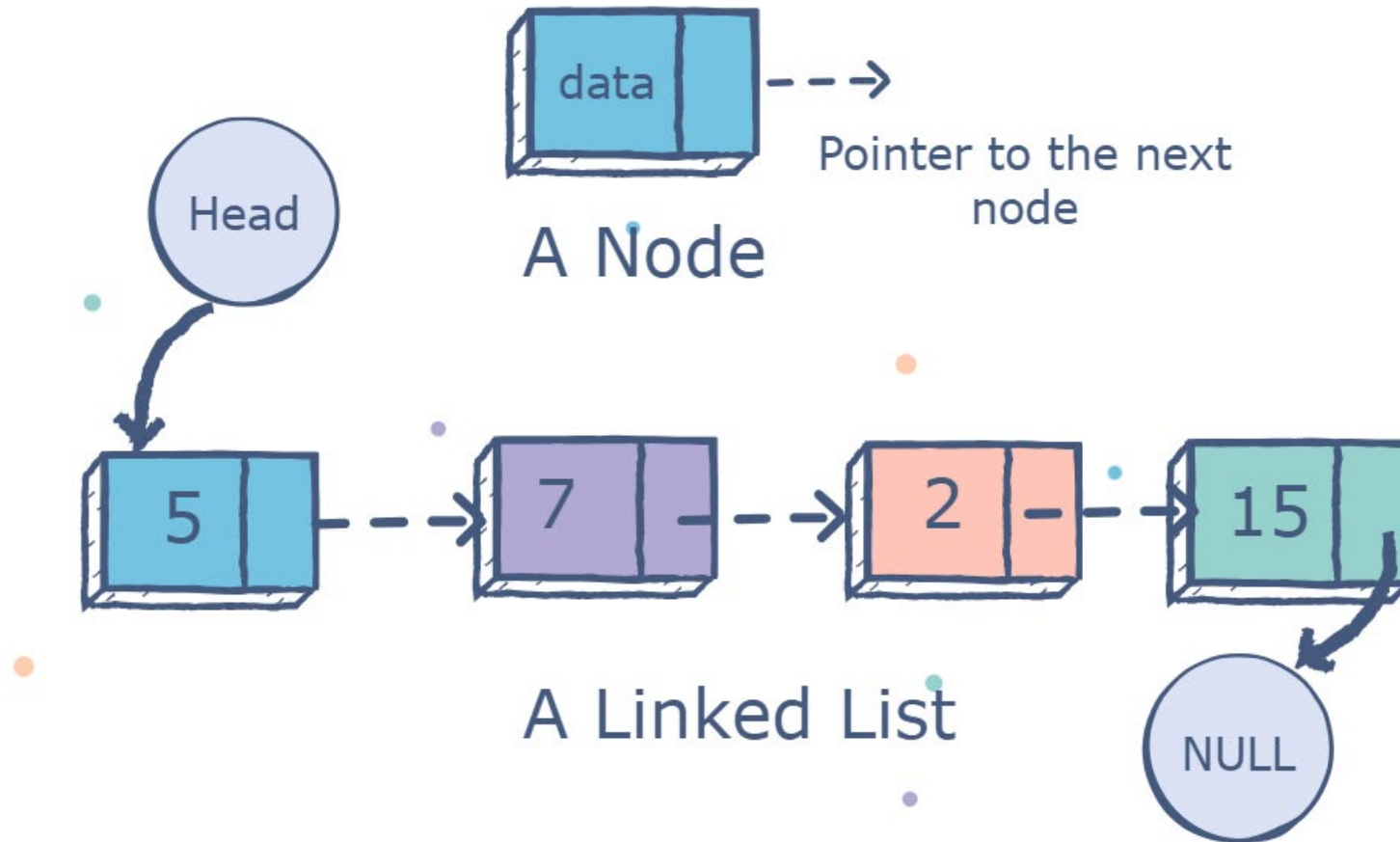
This class will be lecture in Thai Language; don't worry.

# LinkedList Revisited

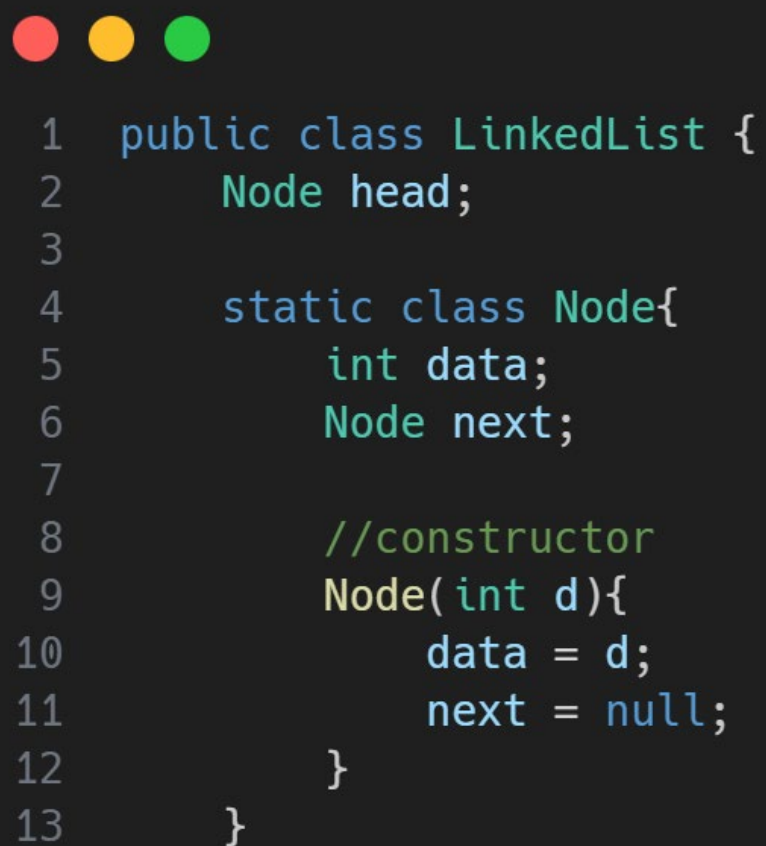


```
1  import java.util.LinkedList;
2
3  class List{
4      public static void main(String[] args) {
5
6          LinkedList<String> list = new LinkedList<String>();
7
8          list.add("Banana");
9          list.add("Apple");
10         list.add("Orange");
11         list.add("Ananas");
12         list.add("Tomato");
13
14         System.out.println(list);
15
16         list.remove(2);
17
18         System.out.println(list);
19     }
20 }
```

```
[Banana, Apple, Orange, Ananas, Tomato]
[Banana, Apple, Ananas, Tomato]
```



<https://www.educative.io/answers/what-is-a-linked-list>



```
1 public class LinkedList {  
2     Node head;  
3  
4     static class Node{  
5         int data;  
6         Node next;  
7  
8         //constructor  
9         Node(int d){  
10             data = d;  
11             next = null;  
12         }  
13     }
```

สร้าง class Node

โดยเก็บ data และตัวชี้ (next) ยังไม่ชี้ไปหาอะไร (null)



```
1 public static LinkedList insert(LinkedList list, int data){
2
3     Node new_node = new Node(data);
4
5     if(list.head == null){          หากไม่มี node อยู่ใน list เลย
6         list.head = new_node;      ให้สร้างขึ้นมาเป็น node แรก
7     }
8     else{
9         Node tail = list.head;
10        while(tail.next != null){   หากมี head อยู่แล้ว, ใส่ node
11            tail = tail.next;        ใน list จนกว่าจะเจอ tail ที่ถูก
12        }                           (tail.next จะชี้ไป null)
13        tail.next = new_node;
14    }
15    return list;
16 }
```

ใส่ Node ใหม่ไปใน list





```
1 public static LinkedList deleteByKey(LinkedList list, int key){
2     Node currNode = list.head, prev = null;
3
4     //CASE 1 : If head node itself holds the key to be deleted
5     if(currNode != null && currNode.data == key){
6         list.head = currNode.next;
7
8         System.out.println(key + " found and deleted");
9         return list;
10    }
11
12    //CASE 2 : If the key is somewhere other than at head
13    while(currNode != null && currNode.data !=key){
14        prev = currNode;
15        currNode = currNode.next;
16    }
17    if(currNode != null){
18        prev.next = currNode.next;
19        System.out.println(key + " found and deleted");
20    }
21    //CASE3 : The key is not present
22    if(currNode == null){
23        System.out.println(key + " not found");
24    }
25    return list;
26 }
```

ลบ Node โดยการบอก data (key)

## ลบ Node โดยการบอกตำแหน่ง

```
1 public static LinkedList deletePos(LinkedList list, int idx){
2     Node currNode = list.head, prev = null;
3
4     //CASE 1 : If idx = 0, head node is to be deleted
5     if(idx == 0 && currNode != null){
6         list.head = currNode.next;
7
8         System.out.println(idx + " position element deleted");
9         return list;
10    }
11
12    //CASE 2 : 0 < idx < list.size()
13    int counter = 0;
14
15    while(currNode != null){
16        if(counter == idx){
17            prev.next = currNode.next;
18            System.out.println(idx + "position element deleted");
19            break;
20        }
21        else{
22            prev = currNode;
23            currNode = currNode.next;
24            counter++;
25        }
26    }
27    //CASE 3 : idx >= list.size()
28    if(currNode == null){
29        System.out.println(idx + " position element not found");
30    }
31    return list;
32 }
33
```



```
1  public static void printList(LinkedList list){  
2      Node currNode = list.head;  
3      System.out.print("\nThis LinkedList contains : ");  
4  
5      while(currNode != null){  
6          System.out.print(currNode.data + " ");  
7          currNode = currNode.next;  
8      }  
9      System.out.println("\n");  
10 }
```

```

1 public static void main(String[] args) {
2     LinkedList list = new LinkedList();
3
4     list = insert(list, 1);
5     list = insert(list, 2);
6     list = insert(list, 3);
7     list = insert(list, 4);
8     list = insert(list, 5);
9     list = insert(list, 6);
10    list = insert(list, 7);
11    list = insert(list, 8);
12    printList(list);
13
14    deleteByKey(list, 1);
15    printList(list);
16
17    deleteByKey(list, 4);
18    printList(list);
19
20    deleteByKey(list, 10);
21    printList(list);
22
23    deletePos(list, 0);
24    printList(list);
25
26    deletePos(list, 2);
27    printList(list);
28
29    deletePos(list, 10);
30    printList(list);
31 }
32 } //Class LinkedList

```

ลองคิดดู : Output ที่ได้คืออะไร ?

```

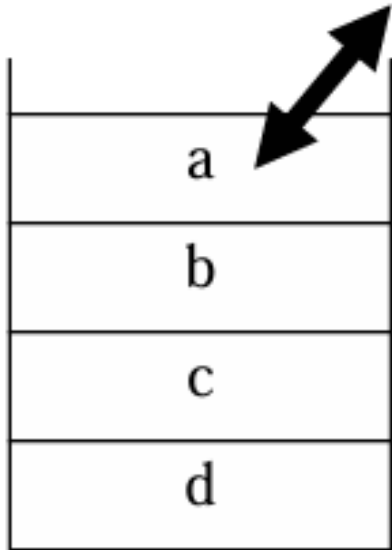
This LinkedList contains : 1 2 3 4 5 6 7 8
1 found and deleted
This LinkedList contains : 2 3 4 5 6 7 8
4 found and deleted
This LinkedList contains : 2 3 5 6 7 8
10 not found
This LinkedList contains : 2 3 5 6 7 8
0 position element deleted
This LinkedList contains : 3 5 6 7 8
2position element deleted
This LinkedList contains : 3 5 7 8
10 position element not found
This LinkedList contains : 3 5 7 8

```

# Stack (in LinkedList)

# What is Stack?

คือ โครงสร้างของข้อมูลประเภทหนึ่งที่มีลักษณะการเก็บข้อมูลเรียงซ้อนกันเป็นชั้นๆไป

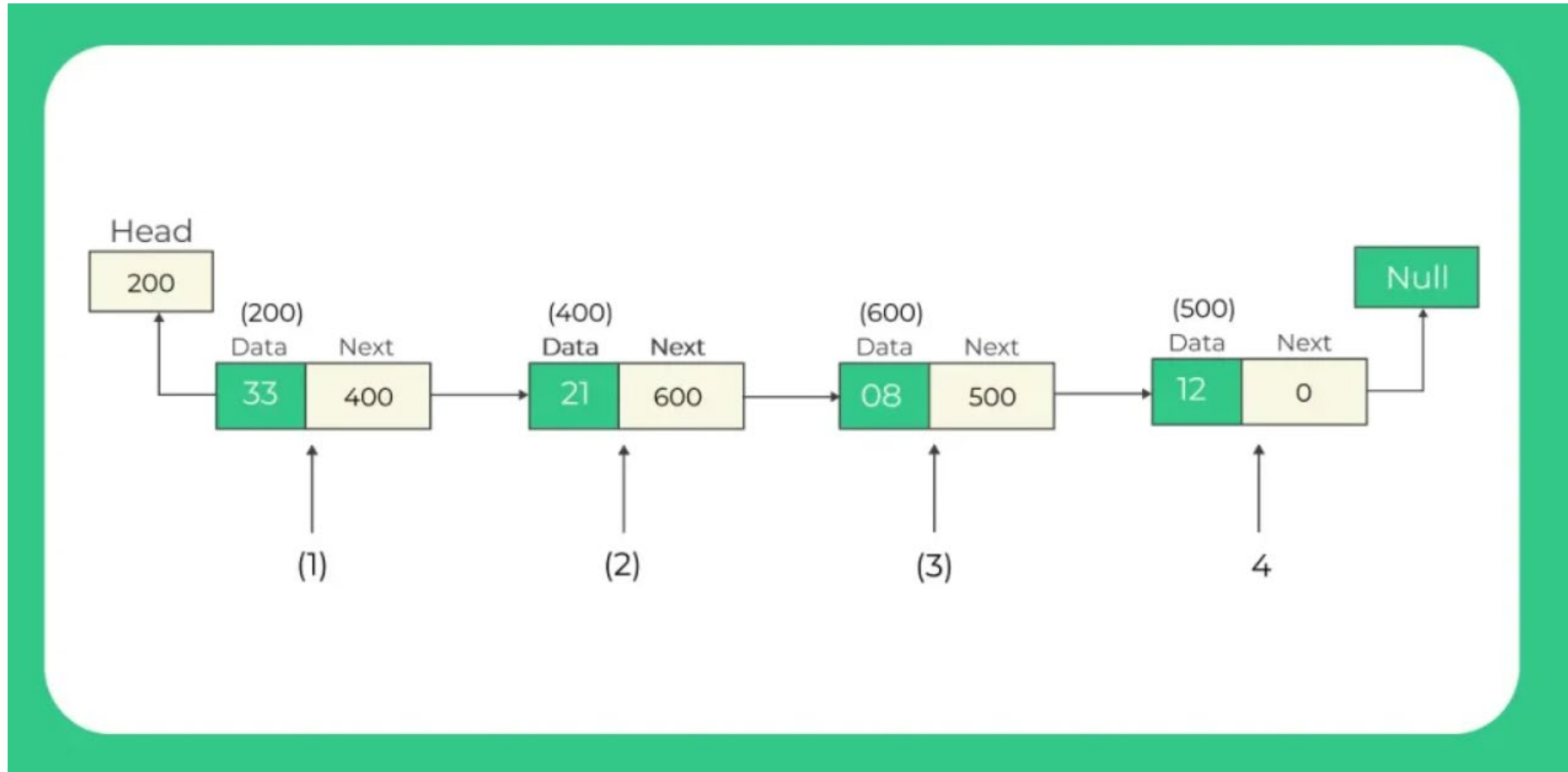


ใส่ของและเอา  
ของออกได้จาก  
ด้านบนเท่านั้น

เราเรียกการเข้าออกข้อมูลแบบนี้ว่า  
Last In, First Out (LIFO)

[aj.vishnu - listStackQueue01.pdf \(chula.ac.th\)](#)

# How stack represents in memory



Stack using Linked List in C | PreplInsta

# How stack represents in memory

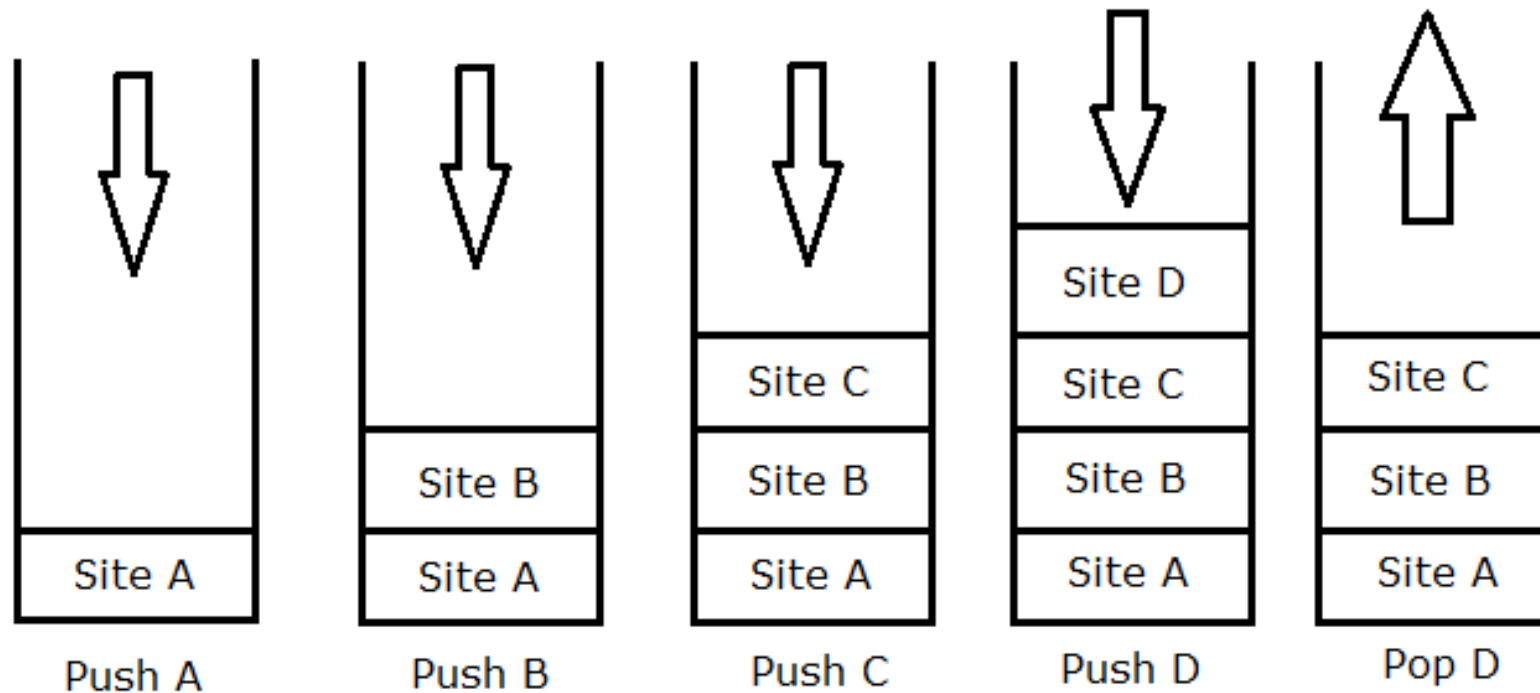
Head

'H'	4	' '	26	'E'	12		
		'L'	22	'L'	10		
						'O'	32
		'W'	28	'O'	30	'R'	34
'.'	2	'd'	36	'L'	null		

อยากรู้ว่าเราจะวาดในรูปแบบ Stack ได้อย่างไร



# Method Concepts of Stack

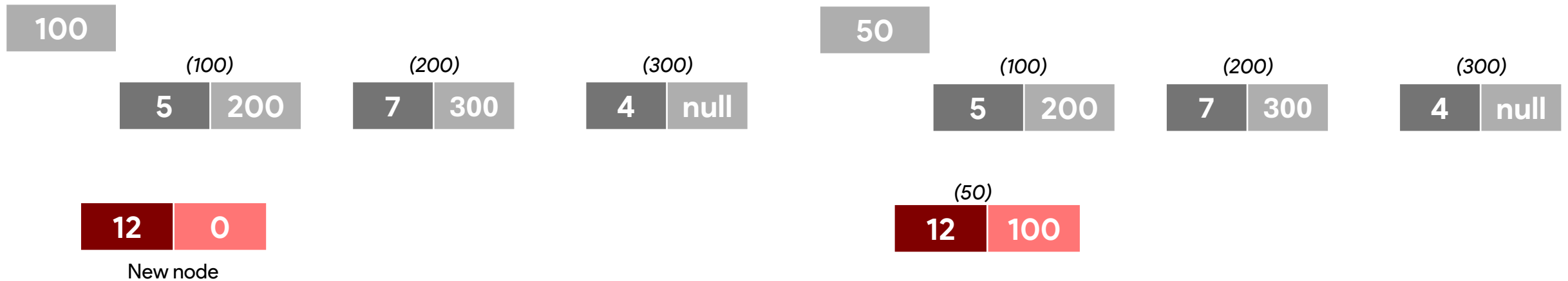
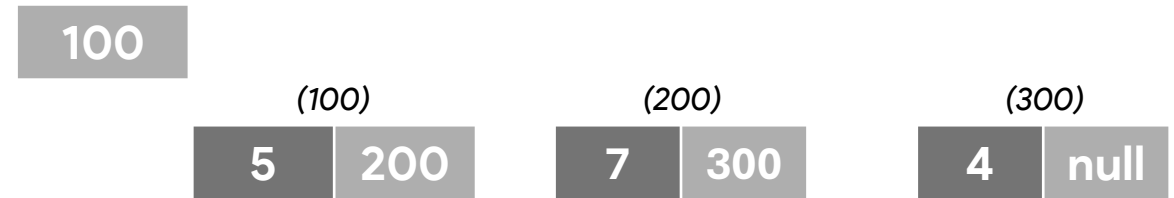


[data structures - Stacks, queues and linked lists - Stack Overflow](#)

push(data) : ใส่ชั้นข้อมูลใหม่ไปใน Stack / pop() : เอาชั้นข้อมูลที่อยู่ข้างบนสุดออก

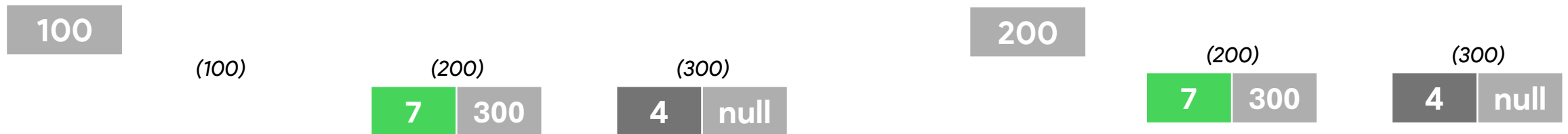
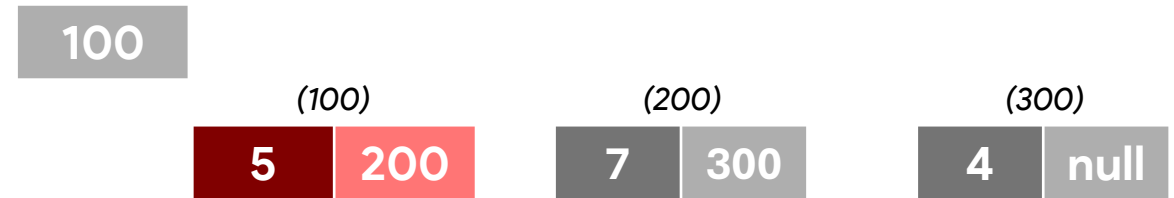
# Method Concepts of Stack : push(data)

1. สร้าง node ใหม่ขึ้นมา
2. ให้ pointer ของ node ใหม่ชี้ไปหาชั้นบนสุดของ Stack นั้น
3. เปลี่ยน head ให้ชี้ไปหา node ใหม่

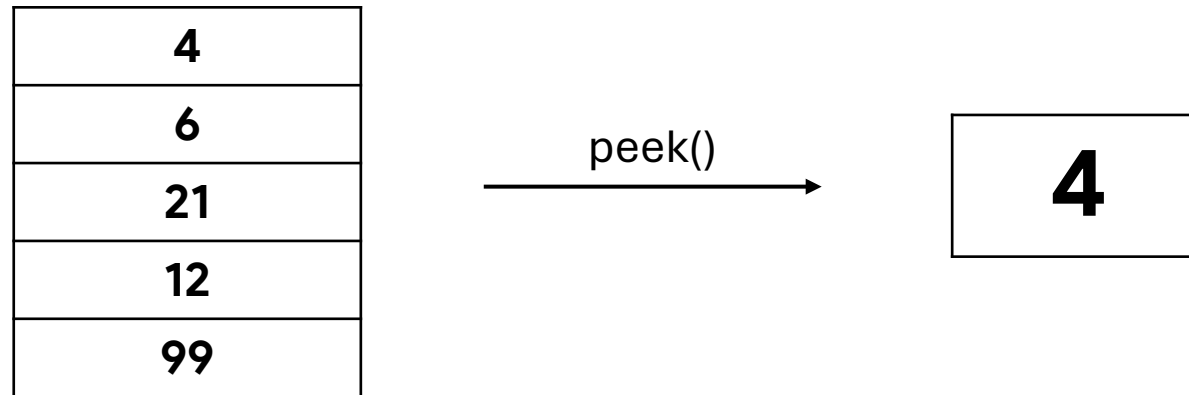


# Method Concepts of Stack : pop()

1. เชื่อกว่า Stack ของเรานั้น มีข้อมูลอยู่หรือเปล่า
2. หากมี ให้สร้าง node เพื่อรองรับตัวที่จะออกจาก stack
3. ให้ย้าย head ไว้ตัวถัดไป (ที่ pointer ของ head ชี้อยู่)
4. ให้ data ใน node ที่รองรับ head ตัวเก่าไว้เป็น null



# Method Concepts of Stack : peek() / top()



`peek()` / `top()` : เรียกดูชั้นบนสุดของ Stack

# Stack implementation using LinkedList

สร้าง class Node

```
1 class Node{
2     int data;
3     Node next;
4
5     Node(int new_data){
6         this.data = new_data;
7         this.next = null;
8     }
9 }
```

สร้าง class Stack โดยที่มี constructor class  
this.head = null หมายถึงไม่มีอะไรใน Stack

```
1 class Stack{
2     Node head;
3
4     Stack(){
5         this.head = null;
6     }
7 }
```

# Stack implementation using LinkedList



```
1 boolean isEmpty(){  
2     return head == null;  
3 }
```

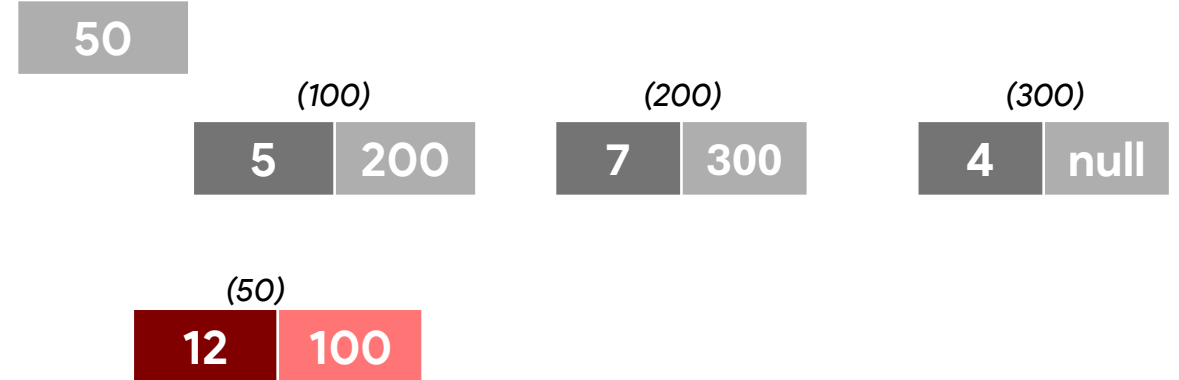
isEmpty() : เช็คว่างหรือไม่

# Stack implementation using LinkedList



```
1 void push(int new_data) {  
2     Node new_node = new Node(new_data);  
3  
4     new_node.next = head;  
5     head = new_node;  
6 }
```

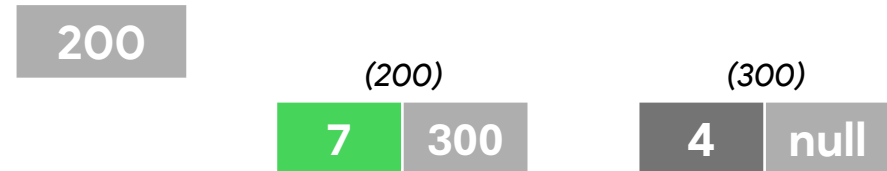
push(data) : เอาข้อมูลใส่ไปใน stack



# Stack implementation using LinkedList

```
1 void pop(){
2     if(isEmpty()){
3         System.out.println("\nStack Underflow");
4         return;
5     }
6     else{
7         Node temp = head;
8         head = head.next;
9         temp = null;
10    }
11 }
```

pop(): เอาข้อมูลที่อยู่บนสุดของ Stack ออก





# Stack implementation using LinkedList

```
1  int peek(){
2      if(!isEmpty()){
3          return head.data;
4      }
5      else{
6          System.out.println("\nStack is empty");
7          return Integer.MIN_VALUE;
8      }
9  }
10 } // CLASS STACK
```

peek() / top() : เรียกดูชั้นบนสุดของ Stack

# Stack implementation using LinkedList



```
1 public class StackEx{
2     public static void main(String[] args) {
3         Stack st = new Stack();
4
5         st.push(11);
6         st.push(22);
7         st.push(33);
8         st.push(44);
9
10        System.out.println("Top element is " + st.peek());
11
12        System.out.println("Removing two elements...");
13        st.pop();
14        st.pop();
15
16        // Print top element of the stack
17        System.out.println("Top element is " + st.peek());
18    }
19 }
```

44	
33	
22	
11	

33	
22	
11	

22	
11	

# Queue (in LinkedList)

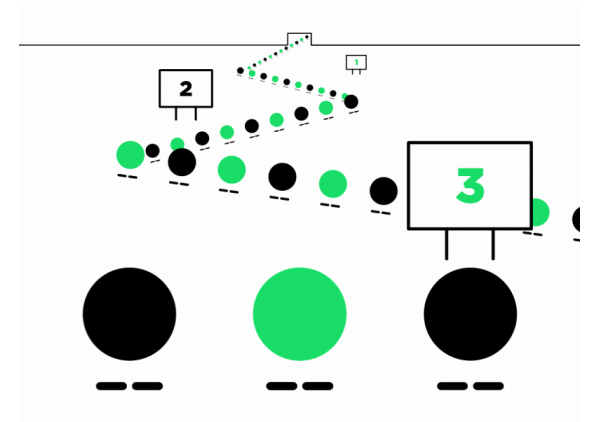
# What is Queue?

คือ โครงสร้างของข้อมูลประเภทหนึ่งที่มีลักษณะการเก็บข้อมูลเรียงแถวเป็นลำดับ



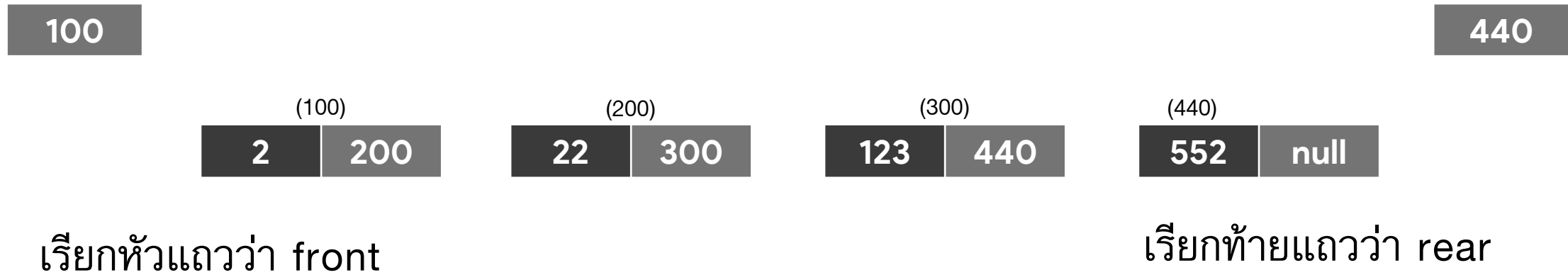
[What is Queue Data Structure? - GeeksforGeeks](#)

เราเรียกการเข้าออกข้อมูลแบบนี้ว่า  
First In, First Out (FIFO)



# How Queue represents in memory?

(เข้าใจง่ายกว่า Stack)



# Method concepts of Queue

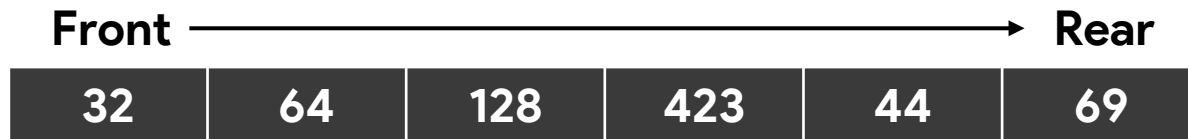


`dequeue()` : หัวคิวออกจากแถว

`enqueue(data)` : ข้อมูลใหม่ต่อท้ายแถว



# Method concepts of Queue



getFront() : เรียกดูหัวแถว

32

getRear() : เรียกดูท้ายแถว

69

# Queue implementation using LinkedList

สร้าง class Node

```
1 class Node{
2     int data;
3     Node next;
4
5     Node(int new_data){
6         this.data = new_data;
7         this.next = null;
8     }
9 }
```

สร้าง class Queue ที่มี constructor เรียกคิวที่ยังไม่มี data ใดๆ

```
1 class Queue{
2     Node front, rear;
3
4     Queue(){
5         front = rear = null;
6     }
}
```



# Queue implementation using LinkedList

isEmpty() : เช็คว่างหรือไม่ (เช็คหัวและท้าย)



```
1  boolean isEmpty(){  
2      return front == null & rear == null;  
3  }
```

# Queue implementation using LinkedList



```
1 void enqueue(int new_data){  
2     Node new_node = new Node(new_data);  
3  
4     if(rear == null){  
5         front = rear = new_node;  
6         return;  
7     }  
8  
9     rear.next = new_node;  
10    rear = new_node;  
11 }
```

enqueue(data) : ใส่ข้อมูลใหม่ท้ายแถว



# Queue implementation using LinkedList

```
1 void dequeue(){
2     if(isEmpty()){
3         System.out.println("Queue is empty");
4         return;
5     }
6
7     Node temp = front;
8     front = front.next;
9
10    if(front == null){
11        rear = null;
12    }
13 }
```

dequeue() : ลบข้อมูลหัวแถว



# Queue implementation using LinkedList

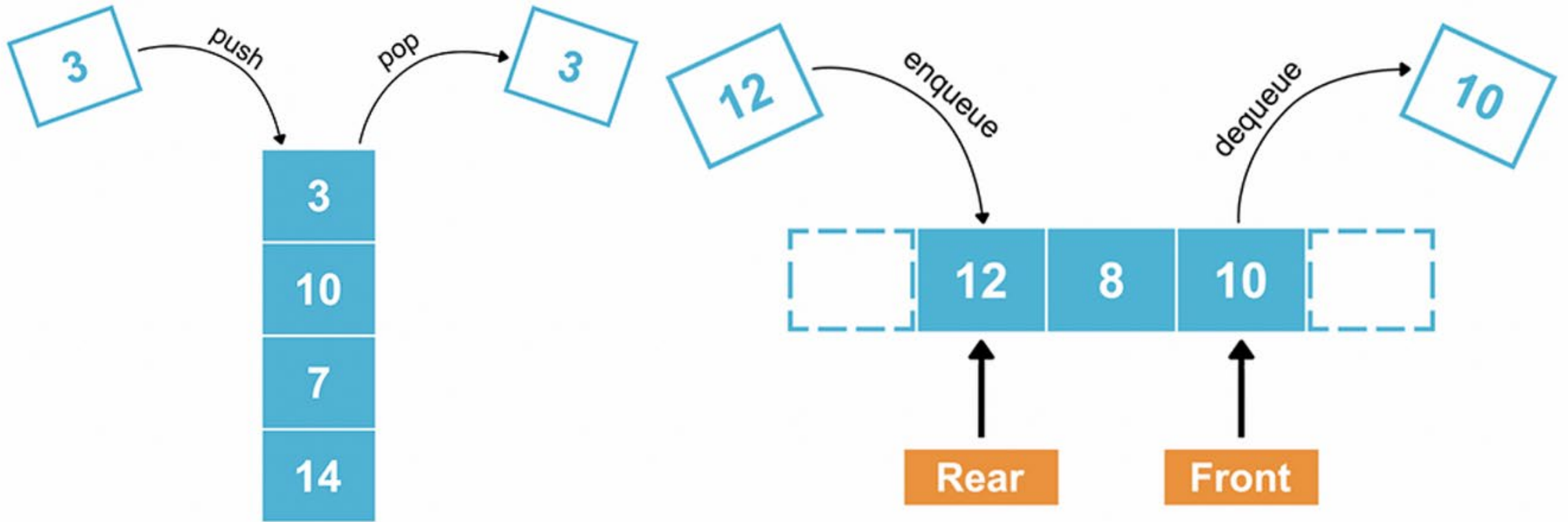
```
1  int getFront(){
2      if(isEmpty()){
3          System.out.println("Queue is empty");
4          return Integer.MIN_VALUE;
5      }
6      return front.data;
7  }
8
9  int getRear(){
10     if(isEmpty()){
11         System.out.println("Queue is empty");
12         return Integer.MIN_VALUE;
13     }
14     return rear.data;
15 }
16 } //CLASS QUEUE
```

getFront() : เรียกดูข้อมูลหัวแถว

getRear() : เรียกดูข้อมูลท้ายแถว



# Summary



[Stack vs Queue — Everything You Need to Know | Better Programming](#)

## Common Data Structure Operations

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
<u>Array</u>	$\theta(1)$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(n)$
<u>Stack</u>	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$\theta(n)$
<u>Queue</u>	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$\theta(n)$
<u>Singly-Linked List</u>	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$\theta(n)$
<u>Doubly-Linked List</u>	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$\theta(n)$

Big-O Algorithm Complexity Cheat Sheet (Know Thy Complexities!) @ericdrowell (bigocheatsheet.com)

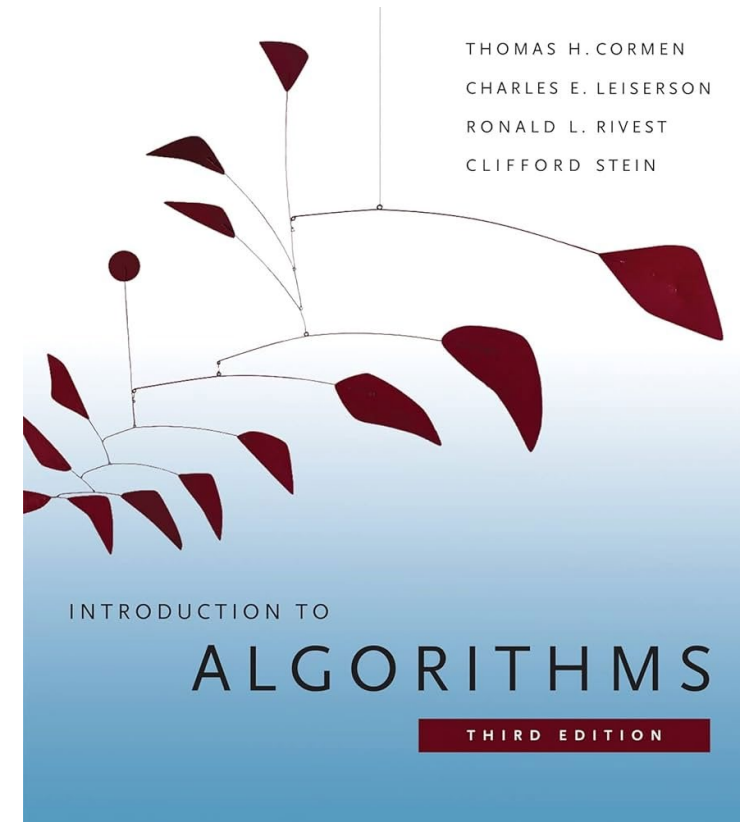
# Code References

[Implement a stack using singly linked list – GeeksforGeeks](#)

[Queue - Linked List Implementation - GeeksforGeeks](#)

## **Introduction to Algorithms, 3rd Edition**

by [Thomas H Cormen](#), [Charles E Leiserson](#), [Ronald L Rivest](#), [Clifford Stein](#)





Thank you for your attention



@bejewelledbud

Can you guys please recommend books that made you cry?



Frease  
@FreaseDaddy



@\_char.mander\_

**Data Structures and Algorithms in Java (2nd Edition)** 2nd Edition  
by Robert Lafore (Author)  
★★★★☆ 114 customer reviews  
[Look inside](#)



Kindle \$29.80	<b>Hardcover</b> \$33.89 - \$45.04	Paperback \$23.39 - \$27.18	Other Se See all 6 versi
<input type="radio"/> Buy used			
<input checked="" type="radio"/> Buy new In Stock			

I cried as hell : r/ProgrammerHumor (reddit.com)