Stacks and Queues

What is a Stack?

A Stack is a linear data structure that follows the **LIFO (Last-In-First-Out)** principle. Stack has one end, whereas the Queue has two ends (**front and rear**).

Some key points related to stack

- It is called as stack because it behaves like a real-world stack, piles of books, etc.
- A Stack is an abstract data type with a pre-defined capacity, which means that it can store the elements of a limited size.
- It is a data structure that follows some order to insert and delete the elements, and that order can be LIFO or FILO.

Standard Stack Operations

The following are some common operations implemented on the stack:

- **push():** When we insert an element in a stack then the operation is known as a push. If the stack is full then the overflow condition occurs.
- **pop():** When we delete an element from the stack, the operation is known as a pop. If the stack is empty means that no element exists in the stack, this state is known as an underflow state.
- **isEmpty():** It determines whether the stack is empty or not.
- **isFull():** It determines whether the stack is full or not.'
- **peek():** It returns the element at the given position.
- **count():** It returns the total number of elements available in a stack.
- **change():** It changes the element at the given position.
- **display():** It prints all the elements available in the stack.

Uses

Stacks can be used for Backtracking, i.e., to check the parenthesis matching in an expression

Stack using array

```
package Stacks;
public class StackUsingArray {
    public static void main(String[] args) {
        Stacks s = new Stacks(5);
        s.push(8);
        s.push(9);
        s.push(10);
        s.push(11);
        s.push(12);
        s.display();
        s.push(13);
        System.out.println(s.pop());
        s.display();
        System.out.println(s.pop());
        s.display();
        System.out.println(s.peek());
//        System.out.println(s.peek());
    }
    private static int[] array;
    private static int position = 0;
    private static int last;
    private static class Stacks {
        public Stacks(int size) {
            array = new int[size];
            last = array.length;
        }

        // insert
        public void push(int val) {
            if (isFull()) {
                System.out.println("stack is full");
                return;
            }
            array[position] = val;
            position++;
        }

        // remove
        public int pop() {
            if (isEmpty()) {
                return -1;
            }
            int val = array[last - 1];
            last--;
            return val;
        }

        // peek
        public int peek() {
            if (isEmpty()) {
                return -1;
            }
            return array[last - 1];
        }

        public boolean isFull() {
           return position == last;
        }


        public boolean isEmpty() {
            return position == -1;
        }


        public void display() {
            for (int i = last - 1; i >= 0; i--) {
                System.out.print(array[i] + " -> ");
            }
            System.out.println("end");
        }
    }
}
```

Stack using ArrayLIst

```java
package Stacks;

import java.util.ArrayList;
import java.util.Stack;

public class StacksAL {

    static class Stacks {
        ArrayList<Integer> list = new ArrayList<>();

        public void push(int value) {
            list.add(value);
        }


        public int pop() {
            return list.remove(list.size() - 1);
        }


        public int peek() {
            return list.get(list.size() - 1);
        }
    }



    public static void main(String[] args) {
        Stacks s1 = new Stacks();
        s1.push(1);
        s1.push(2);
        s1.push(3);
        s1.push(4);
        s1.push(5);


        while(!s1.list.isEmpty()) {
            System.out.print(s1.peek() + " -> ");
            s1.pop();
        }
        System.out.println("start");
    }
}
```

Stacks using LinkedList

```
package Stacks;

public class StacksLL {

    static Node head;
    static class Node {
        int value;
        Node next;

        Node (int value) {
            this.value = value;
            this.next = null;
        }
    }


    static class Stack {
//        to insert data
        public void push(int value) {
            Node node = new Node(value);

            if (isEmpty()){
                head = node;
                return;
            }
            node.next = head;
            head = node;
        }


        public int pop() {
            if (isEmpty()) {
                return -1;
            }

            Node temp = head;
            head = head.next;
            return temp.value;
        }


        public int peek() {
            if (head == null) {
                return -1;
            }

            Node top = head;
            return top.value;
        }



        public boolean isEmpty() {
            return head == null;
        }
    }
```

```
    public static void main(String[] args) {
        Stack s1 = new Stack();
        s1.push(1);
        s1.push(2);
        s1.push(3);
        s1.push(4);
        s1.push(5);

        while (!s1.isEmpty()) {
            System.out.println(s1.peek());
            s1.pop();
        }

    }
}
```

Inbuilt Stack

```
package Stacks;

import java.util.Stack;

public class Inbuilt {
    public static void main(String[] args) {
        Stack<Integer> s1 = new Stack<>();
        s1.push(1);
        s1.push(2);
        s1.push(3);
        s1.push(4);
        s1.push(5);


        while(!s1.isEmpty()) {
            System.out.print(s1.peek()+" -> ");
            s1.pop();
        }
        System.out.println("end");

    }
}
```

Questions on Stacks

Find Largest Area of Histogram

```
package Stacks.Questions;

import java.util.Stack;

public class LargestAreaHistogram {
    public static void main(String[] args) {
        int[] height = {2, 1, 5, 6, 2, 3};
        LargestAreaHistogram l = new LargestAreaHistogram();
//        System.out.println(l.largestRectangleArea(height));
        System.out.println(l.largestRectangleArea(height));
    }



    public int largestRectangleArea(int[] height) {
        Stack<Integer> stack = new Stack<>();
        int max = 0;


        stack.push(0);
        for (int i = 1; i < height.length; i++) {
             while(!stack.isEmpty() && height[i] < height[stack.peek()]) {
                max = getMax(height, stack, max, i);
            }
            stack.push(i);
        }


        int i = height.length;
        while(!stack.isEmpty()) {
            max = getMax(height, stack, max, i);
        }
        return max;
    }



    private static int getMax(int[] height, Stack<Integer> s, int max, int
i){
        int area;
        int popped = s.pop();
        if (s.isEmpty()) {
            area = height[popped] * i;
        }
        else {
            area = height[popped] * (i - 1 - s.peek());
        }
        return Math.max(max, area);
    }

}
```

Push element at buttom

```java
package Stacks.Questions;

import java.util.*;
public class PushAtBottom {
    public static void pushAtBottom(int data , Stack<Integer> s) {
        if (s.isEmpty()) {
            s.push(data);
            return;
        }

        int temp = s.pop();
        pushAtBottom(data, s);
        s.push(temp);
    }

    public static void main(String[] args) {
        Stack<Integer> s = new Stack<>();
        s.push(1);
        s.push(2);
        s.push(3);

        pushAtBottom(4, s);
        while(!s.isEmpty()) {
            System.out.println(s.peek());
            s.pop();
        }
    }
}
```

Reverse the element

```java
package Stacks.Questions;

import java.util.Stack;

public class Reverse {
    public static void reverse(Stack<Integer> stack) {
        while(!stack.isEmpty()) {
            System.out.print(stack.pop() + " -> ");
        }
        System.out.println("start");
    }

    public static void main(String args[]) {
        Stack<Integer> stack = new Stack<>();
        stack.push(1);
        stack.push(2);
        stack.push(3);

        reverse(stack);
    }
}
```

Check valid Parentheses

```java
package Stacks.Questions;
import java.util.*;

public class Valid Parentheses {
    public static void main(String[] args) {
        String s = "{[()]}";
        System.out.println(isValid(s));;
    }

    public static boolean isValid(String s) {
        Stack<Character> stack = new Stack<>();

        for(char ch: s.toCharArray()) {
            if (ch == '{' || ch == '(' || ch == '[') {
                stack.push(ch);
            }
            else {
                if (ch == '}') {
                    if (stack.isEmpty() || stack.pop() != '{') {
                        return false;
                    }
                }

                if (ch == ']') {
                    if (stack.isEmpty() || stack.pop() != '[') {
                        return false;
                    }
                }

                if (ch == ')') {
                    if (stack.isEmpty() || stack.pop() != '(') {
                        return false;
                    }
                }
            }
        }
        return stack.isEmpty();
    }
}
```

Make valid parentheses

```java
package Stacks.Questions;
import java.util.*;

public class Make valid parentheses {
    public static void main(String[] args) {
        Make_valid_parentheses m = new Make_valid_parentheses();
        String s = "(()))";
        System.out.println(m.minAddToMakeValid(s));
    }

    public int minAddToMakeValid(String s) {
        Stack<Character> stack = new Stack<>();

        for(char ch: s.toCharArray()) {
            if (ch == ')') {
                if (!stack.isEmpty() && stack.peek() == '(') {
                    stack.pop();
                }
                else {
                    stack.push(ch);
                }
            }
            else {
                stack.push(ch);
            }
        }
        return stack.size();
    }
}
```

What is a Queue?

- A Queue is a linear data structure that follows the **FIFO (First-In-First-Out)** principle. Queue has two ends (**front and rear**).
- For example, people waiting in line for a rail ticket form a queue.

Basic Operations on Queue:

Some of the basic operations for Queue in Data Structure are:

- **enqueue() –** Insertion of elements to the queue.
- **dequeue() –** Removal of elements from the queue.
- **peek() or front()-** Acquires the data element available at the front node of the queue without deleting it.
- **rear() –** This operation returns the element at the rear end without removing it.
- **isFull() –** Validates if the queue is full.
- **isEmpty() –** Checks if the queue is empty.
- **size():** This operation returns the size of the queue i.e. the total number of elements it contains.

Characteristics of Queue:

- Queue can handle multiple data.
- We can access both ends.
- They are fast and flexible.

Uses

Queues are typically used to manage threads in multithreading and implementing priority queuing systems.

Queue using array

```
package Queues;

public class QueueUsingArray {
//    static class CustomQueue {
        private int[] array;

        int end = -1;

        public QueueUsingArray(int size) {

            array = new int[size];
        }

        public boolean isFull() {
            return end == array.length;
        }

        public boolean isEmpty() {
            return end == -1;
        }

        public void insertFirst(int item)  {
            if (isFull()) {
                System.out.println("queue is full");
            }
            end++;
            array[end] = item;
        }

        public int remove() throws Exception {
            int val = array[0];
            if (isEmpty()) {
                throw new Exception("Queue is empty");
            }

            for (int i = 0; i < end - 1; i++) {
                array[i] = array[i + 1];
            }
            end--;
            return val;
        }


        public int front() throws Exception {
            if (isEmpty()) {
                throw new Exception("Queue is Empty");
            }

            return array[0];
        }


        public void display() {
            for (int i = 0; i <= end; i++) {
                System.out.print(array[i] + " <- ");
            }
            System.out.println("end");
        }
//    }

    public static void main(String[] args) throws Exception {
        QueueUsingArray q = new QueueUsingArray(5);
        q.insertFirst(4);
        q.insertFirst(5);
        q.insertFirst(6);
        q.insertFirst(7);
        q.insertFirst(8);
        q.display();
        System.out.println(q.remove());
        q.display();
        System.out.println(q.front());
    }

}
```

Queue using Arraylist

```
package Queues;

import java.util.ArrayList;

public class QueueAL {
    public static void main(String[] args) {
//
        Que q = new Que(6);
        q.add(5);
        q.add(6);
        q.add(7);
        q.add(8);
        q.add(9);
        q.add(10);
        q.add(11);
        q.add(12);
        q.display();
        System.out.println(q.remove());
        q.display();
        System.out.println(q.remove());
        q.display();
        System.out.println(q.peek());
    }

    private static ArrayList<Integer> a;

    public static class Que {
        Que(int size) {
            a = new ArrayList<>();
        }

        public void add(int val) {
            a.add(val);
        }


        public int remove() {
            if (a.isEmpty()) {
                return -1;
            }
            int val = a.get(0);
            a.remove(0);
            return val;
        }


        public int peek() {
            if (a.isEmpty()) {
                return -1;
            }

            return a.get(0);
        }


        public void display() {
            for (int i = 0; i < a.size(); i++) {
                System.out.print(a.get(i) + " -> ");
            }

            System.out.println("end");
        }
    }
}
```

Queue using LinkedList

```
package Queues;

public class QueueLL {
    public static void main(String[] args) {
        QueueLL q = new QueueLL();
        q.insert(5);
        q.insert(6);
        q.insert(7);
        q.insert(8);
        q.display();

        System.out.println(q.remove());
        q.display();

        System.out.println(q.front());
    }


    private Node head;
    private Node tail;
    private int size;

    public QueueLL() {
        this.size = 0;
    }

    private static class Node {
        int value;
        Node next;

        Node(int value) {
            this.value = value;
        }
    }


//    enqueue
    public void insert(int value) {
        Node node = new Node(value);
        if (tail == null) {
            tail = head = node;
            return;
        }

        tail.next = node;
        tail = node;
        size++;
    }


//    dequeue
    public int remove() {
        int remove = head.value;
        if (tail == null) {
            return -1;
        }
        head = head.next;
        size--;
        return remove;
```

```
    }


//    peek
    public int front() {
        int front = head.value;
        if (tail == null) {

        }
        return front;
    }


    public void display() {
        Node temp = head;
        for (int i = 0; i <= size; i++) {
            System.out.print(temp.value+" -> ");
            temp = temp.next;
        }
        System.out.println("end");
    }

}
```

Queue Inbuilt

```
package Queues;
import java.util.*;
public class QueueInbuilt {
    public static void main(String[] args) {
        Queue<Integer> q = new LinkedList<>();
        q.add(1);
        q.add(2);
        q.add(3);
        q.add(4);

        System.out.println(q);
        System.out.println(q.remove());
        while(!q.isEmpty()) {
            System.out.println(q.peek());
            q.remove();
        }
    }
}
```

Circular Queue

```
package Queues;

public class CircularQueue {

    protected int[] array;
    protected int front = 0;
    protected int end = 0;
    private int size = 0;
    public CircularQueue(int num) {
        array = new int[num];
    }

    public boolean isFull() {
        return size == array.length;
    }

    public boolean isEmpty() {
        return size == 0;
    }

    public void insert(int item) {
        if (isFull()) {
            System.out.println("Full");
            return;
        }

        array[end++] = item;
        end = end % array.length;
        size++;
    }


    public int remove() throws Exception {
        int ret =array[front];
        if(isEmpty()) {
            throw new Exception("Empty");
        }

        front++;
        front = front % array.length;
        size--;
        return ret;
    }


    public int front() throws Exception {
        if (isEmpty()) {
            throw new Exception("Empty");
        }

        return array[front];
    }

    public void display() {
        if (isEmpty()) {
            System.out.println("empty");
            return;
        }
        int i = front;
```

```
        do {
            System.out.print(array[i]+" -> ");
            i++;
            i %= array.length;
        } while(i != end);
        System.out.println("end");
    }

    public static void main(String[] args) throws Exception {
        CircularQueue l = new CircularQueue(5);
        l.insert(5);
        l.insert(6);
        l.insert(8);
        l.insert(7);
        l.insert(10);
        l.display();

        System.out.println(l.remove());
        l.display();

        System.out.println(l.front());
    }

}
```

Question

Queue using stacks

```
package Queues.Question;

import java.util.*;

public class QueueUsingStacks {
    public static void main(String[] args) {
        Queues q = new Queues();
        q.insert(5);
        q.insert(6);

        while (!q.isEmpty()) {
            System.out.print(q.removes() + " ");

        }
        System.out.println("end");

        System.out.println(q.removes());
    }

    static class Queues {
        Stack<Integer> s1 = new Stack<>();
        Stack<Integer> s2 = new Stack<>();

        public void insert(int item) {
            while(!s1.isEmpty()) {
                s2.push(s1.pop());
            }

            s1.push(item);

            while(!s2.isEmpty()) {
                s1.push(s2.pop());
            }
        }

        public int removes() {
            if (isEmpty()) {
                return -1;
            }
            return s1.pop();
        }


        public int front() {
            if (isEmpty()) {
                return -1;
            }

            return s1.peek();
        }

        public boolean isEmpty() {
            return s1.isEmpty();
        }
    }
}
```