IARE

IARE
INSTITUTE OF
AERONAUTICAL ENGINEERING
(An Autonomous Institute affiliated to JNTUH, Hyderabad)
Dundigal, Hyderabad - 500 043

LABORATORY WORK BOOK

Name of the Student : N. Ravi Chandika

Class : CSD-B Semester : IInd Semester

Course Code : ACSD11 Course Name : DS Laboratory

Name of the Course Faculty : Mr. A. Srikanth Faculty ID : IARE 10652

Exercise Number : 6 Week Number : 6 Date : 29-10-2024

Roll Number									
2	3	9	5	1	A	6	7	B	3

S. No.	Exercise Number	EXERCISE NAME	MARKS AWARDED						
			Aim/ Preparation	Algorithm / Procedure / Performance in the Lab		Source Code / Calculations and Graphs	Program Execution / Results and Error Analysis	Viva - Voce	Total
			4	4		4	4	4	20
1	6-1	Linear queue							
2	6-2	Stack using queues							
3	6-3	Queue using Stacks	4	2	2	4	4	4	20
4	6-4	Circular queue							
5	6-5	Double ended queue							
6									
7									
8									
9									
10									
11									
12									

Ravi Chandika.N
Signature of the Student

Signature of the Faculty

START WRITING FROM HERE

6.1 Aim: Linear queue is a data structure that stores items in First In-First out manner. With a queue the least recently added item will be removed first.

Code:

```
import java.util.Scanner;
public class LinearQueue {
    static final int MAX = 5;
    static int[] queue = new int[MAX];
    static int front = 0;
    static int rear = 0;
    public static boolean isEmpty() {
        return front == rear;
    }
    public static boolean isFull() {
        return rear == MAX;
    }
    public static void enqueue(int item) {
        if (isFull()) {
            System.out.println("Queue is full. Cannot enqueue " + item);
        } else {
            queue[rear++] = item;
            System.out.println("Enqueued: " + item);
        }
    }
    public static void deque() {
        if (isEmpty()) {
            System.out.println("Queue is empty. Cannot deque.");
```

```
    } else {
        System.out.println("Dequeued: " + queue[front]);
        front++
    }
}
public static void main (String args[]) {
    Scanner scanner = new Scanner(System.in);
    int choice;
    do {
        System.out.println("\nQueue Operations:");
        System.out.println("1. Enqueue");
        System.out.println("2. Dequeue");
        System.out.println("3. Display");
        System.out.println("4. Exit");
        System.out.print("Enter your choice: ");
        choice = scanner.nextInt();
        switch (choice) {
            case 1:
                System.out.print("Enter an element to enqueue: ");
                int item = scanner.nextInt();
                enqueue(item);
                break;
            case 2:
                dequeue();
                break;
            case 3:
                display();
                break;
```

```
            case 4:
                System.out.println("Exiting...");
                break;
            default:
                System.out.println("Invalid choice. Please try again.");
        }
    } while (choice != 4);
    scanner.close();
    }
}.
```

Input:

```
Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter an element to enqueue: 23
```

Output:

```
Enqueued: 23.
```

6.2 Aim: Implement a LIFO stack using only two queues. The implemented stack should support all the functions of a normal stack.

Code:

```
import java.util.Stack;
class MyQueue{
```

```
    private Stack<Integer> stack1;
    private Stack<Integer> stack2;

    public MyQueue() {
        stack1 = new Stack<>();
        stack2 = new Stack<>();
    }
    public void push(int x){
        stack1.push(x);
    }
    public int pop() {
        if(stack2.isEmpty()){
            while(!stack1.isEmpty()) {
                stack2.push(stack1.pop());
            }
        }
        return stack2.pop();
    }
    public int peek() {
        if(stack2.IsEmpty()) {
            while(!stack1.isEmpty()) {
                stack2.push(stack.pop());
            }
        }
        return stack2.peek();
    }
    public boolean empty() {
```

```
        return stack1.isEmpty() && stack2.isEmpty();
    }
    public static void main(String args[]) {
        MyQueue myQueue = new MyQueue();
        MyQueue.Push(1);
        MyQueue.Push(2);
        System.out.println("Peek: " + myQueue.peek());
    }
}
```

Output: Peek: 1
POP: 1
Empty: false.

6.3 Aim: Implement a FIFO queue using only two stacks. The implemented queue should support all the functions of a normal queue.

Code:

```
import java.util.LinkedList;
import java.util.Queue;
class MyStack {
    private Queue<Integer> queue;
    public MyStack() {
        queue = new LinkedList<>();
    }
    public void push(int a) {
        queue.add(a);
        int size = queue.size();
```

```
for(int i=0; i<size-1; i++){
        queue.add(queue.remove());
    }
}
public int pop(){
    if(empty()){
        System.out.println("Stack is empty-cannot pop");
        return -1;
    }
    return queue.remove();
}
public int top(){
    if (empty()){
        System.out.println("Stack is empty-");
        return -1;
    }
    return queue.peek();
}
public boolean empty(){
    return queue.isEmpty();
}
public static void main(String args[]){
    MyStack Stack = new MyStack();
    stack.push(1);
    Stack.push(2);
    Stack.push(3);
```

```
        System.out.println("Top element: " +stack.top());
        Stack.pop();
        Stack.pop();
        System.out.println("Is Stack empty?" + Stack.empty());
    }
}
```

Output:

Top Element : 3

Popped Element : 3

Top Element after pop: 2

Is stack empty ? false

Is stack empty after popping all the elements? True.

6.4 Aim: A Circular queue is an extended version of the normal queue is connected to first element of the queue forming a circle.

Code:

```
class CircularQueue{
    private int size, front, rear;
    private int[] queue;
    public CircularQueue(int size){
        this.size = size;
        this.queue = new int[size];
        this.front = this.rear = -1;
    }
```

```
public void enqueue(int data){
    if((rear+1) % size == front) {
        System.out.println("Queue is Full");
    } else {
        if(front == -1) {
            front = 0;
        }
        rear = (rear+1) % size;
        queue[rear] = data;
        System.out.println("Enqueued : " + data);
    }
}.
```

Output:

Enqueued : 10

Enqueued : 20

Enqueued : 30

Enqueued : 40

Enqueued : 50

Queue : 10 20 30 40 50

Dequeued : 10

Dequeued : 20

Queue : 30 40 50

Enqueued : 60

Enqueued : 70

Queue : 30 40 50 60 70.

6.5 Aim: In a deque one can perform insert & delete operations from both ends of a container.

Code:

```
import java.util.Deque;
import java.util.ArrayDeque;
public class Deque {
    public static void main (String args[]) {
        deque<Integer> deque = new ArrayDeque<>();
        deque.addLast(10);
        deque.addLast(20);
        deque.addLast(30);
        deque.addFirst(5);
        System.out.println("Deque after adding elements: " + deque);
        System.out.println("First element: " + deque.peekFirst());
        deque.removeFirst();
        deque.removeLast();
        System.out.println("Deque after removing elements: " + deque);
    }
}
```

Output:

Dequeue : [0,1,2]

Dequeue : [0,1]

Deque : [0,1]

Count of 1 : 1

Front : 0

Back : 0

Deque : [0,1,3,4,5]

Deque : [-1,-2,0,1,2,3,4,5]

Deque : [4,5,-1,-2,0,1,3].