## 10. Stacks and Queues.

**Aim/Objective:** To analyse the implementation of the concept of Stacks and Queues with Interfaces for the real-time scenario.

**Description:** The student will understand the concept of Stacks and Queues.

**Pre-Requisites:** Classes and Objects in JAVA

**Tools:** Eclipse IDE for Enterprise Java and Web Developers

**Pre-Lab:**

1) Write a JAVA program for Basic ADT Operations on Stack Data Structure.

```
class stack{
  private int maxsize;
  private int[] stackArray;
 private int top;
  public Stack(int size){
     this.maxSize = maxSize;
     this.stackArray = new int[maxSize];
     this.top= -1;
}
 public void push(int val){
     if(top == maxSize-1){
       System.out.println("Stack Overflow!"+val);
   }else{
       stackArray[++top] = value;
       System.out.println("Pushed "+val+" to stack");
   }
}
```

```
public int pop() {
    if (isEmpty()) {
     System.out.println("Stack Underflow!");
     return -1;
  } else {
     return stackArray[top--];
   }
}
public static void main(String[] args) {
     Stack s = new Stack(5);
      s.push(10);
      s.push(20);
      s.push(35);
     System.out.println("Top is :"+s.pop());
   }
}
```

| Experiment# | | Student ID | |
|---|---|---|---|
| Date | | Student Name | |

**In-Lab:**

1) Create a generic interface for a stack data structure with additional methods for peeking at the top element without removing it and checking if the stack is empty. Implement the interface using a linked list and an array. Test the implementations with different data types.

Procedure/Program:

```
interface GenericStack<T> {
    void push(T value);
    T pop();
    T peek();
    boolean isEmpty();
}
class LinkedListStack<T> implements GenericStack<T>{
   private class Node {
    T data;
    Node next;
    Node(T data) {
      this.data=data;
    }
}
private Node top;
 public void push(T value) {
  Node newNode = newNode(value);
  newNode.next=top;
  top=newNode;
  System.out.println("Pushed"+value+"to stack");
} public T pop() {
```

```
if (isEmpty()) {
    system.out.println("Linked List Stack is empty ! cannot pop.");
    return null;
}public T Peek() {
    if (isEmpty()) {
        System.out.println("Linked List Stack is null");
        return null;
    }
    return top.data;
}public class StackTest{
    public static void main(String[] args) {
        System.out.println("Testing Linked List Stack with int.");
        LinkedListStack.push(10);
        LinkedListStack.push(20);
        System.out.println("+LinkedListStack.peek());
        System.out.println("+Popped"+LinkedListStack.pop());
        System.out.println("empty?"+LinkedListStack.isEmpty());
    }
}
```

| Experiment# | | Student ID | |
|---|---|---|---|
| Date | | Student Name | |

✓ **Data and Results:**

O/P:

Linked list Stack is empty ! cannot pop

Linked list Stack is null

Testing linked list stack with int.

10

20

✓ **Analysis and Inferences:**

This program demonstrates the implementation of creation of generic interface for stack data structure using linked list & an array.

| Experiment# | | Student ID | |
|---|---|---|---|
| Date | | Student Name | |

**VIVA-VOCE Questions (In-Lab):**

1) List the Differences between Stack and Queue ADT.

Stack:

1) LIFO 2) Push & Pop 3) Operations occur at one end

Queue:

1) FIFO

2) Enqueue & Dequeue

3) Operations occur at both ends.

2) Discuss about the Priority Queue with example

A priority queue is an abstract data type where each element has a priority & elements are dequeued based on their priority rather than their order.

E.g.: In a hospital, patients with more critical conditions (higher) are treated before those with less severe conditions.

3) Illustrate about "PERFORMING QUEUE ADT USING STACK DATA STRUCTURE".

i) 2 Stacks approach:

Use 2 stacks Stack1 for enqueue & stack2 for dequeue.

ii) Enqueue: Push element onto stack 1.

iii) Dequeue: If stack 2 is empty, pop all elements from stack 1 & push them onto stack2 then pop from stack 2.

| Experiment# | | Student ID | |
|---|---|---|---|
| Date | | Student Name | |

4) Illustrate about "PERFORMING STACK ADT USING QUEUE DATA STRUCTURE".

i) Two queue approach:

Use 2 queues queue1 for push operations & queue2 for pop-operations.

ii) Push operation: Enqueue elements into queue1.

iii) Pop operation: Dequeue all elements from queue1, except the last one, enque them into queue2, then swap the names of queue1 & queue2.

5) Discuss the necessity of Priority Queue Data Structure.

A Priority queue is essential for efficiently managing tasks where certain elements need to be processed before others based on priority.

| Experiment# | | Student ID | |
|---|---|---|---|
| Date | | Student Name | |

**Post-Lab:**

1) Create a generic class that implements a priority queue data structure. Test the class with different data types such as integers, doubles, and strings.

Procedure/Program:

```
import java.util.Arraylist;
importjava.util.Comparator;
class Priority Queue <T> {
    private ArrayList<T> elements;
    private Comparator<?super T> comparator;
    public Priority Queue(comparator<?super T>comparator) {
        this.elements = new Arraylist<>();
        this.comparator = comparator;
} public void enqueue (Telement) {
    elements.add(element);
    int i= elements.size()-1;
    while (i>0) {
     int p=(i-1)/2;
     if(comparator.compare(elements.get(i), elements.get((p))<=0) {
        break;
    }
    T temp = elements.get(index);
    elements.set(i,elements.get(P));
    elements.set(P, temp);
    i = P;
} System.out.println ("Enqueued:"+element);
```

```
} public T dequeue() {
    if(isEmpty()) {
      System.out.println("Priority Queue is empty!");
      return null;
    }
    T result = elements.get(0);
    T lastelement = elements.remove(elements.size()-1);
     if(!isEmpty()) {
       elements.set(0, lastElement);
       bubbleDown(0);
     }
     return result;
}
public class Priority{
   public static void main(String[] args) {
      System.out.println("Testing Priority Queue with int");
      PriorityQueue<Integer>int Queue = new
       PriorityQueue<>(comparator.naturalOrder());
       int Queue.enqueue(10);
       int Queue.enqueue(20);
       System.out.println("Dequeued:"+intQueue.dequeue());
       System.out.println("Peek:"+intQueue.peek());
    }
}
```

| Experiment# | | Student ID | |
|---|---|---|---|
| Date | | Student Name | |

✓ **Data and Results:**

Testing priority queue with int

10

20

Dequeued : 20

Peek : 10

✓ **Analysis and Inferences:**

This program demonstrates about the creation of priority queue & testing with different types of data types.

| **Evaluator Remark (if Any):** | **Marks Secured: _____out of 50** |
|---|---|
| | **Signature of the Evaluator with Date** |

**Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.**