Experiment-1

Implement the following Data structures in Java

i) Linked Lists ii) Stacks iii) Queues

i) Linked Lists implementation in JAVA

import java.io.\*;

// Java program to implement

// a Singly Linked List

public class LinkedList<T> {

Node head; // head of list

// Linked list Node.

// This inner class is made static

// so that main() can access it

static class Node<T> {

T data;

Node next;

// Constructor

Node(T d)

{

data = d;

next = null;

}

}

// Method to insert a new node

public static LinkedList<T> insert(LinkedList<T> list, int data)

{

// Create a new node with given data

Node new\_node = new Node<T>(data);

// If the Linked List is empty,

// then make the new node as head

if (list.head == null) {

list.head = new\_node;

}

else {

// Else traverse till the last node

// and insert the new\_node there

Node last = list.head;

while (last.next != null) {

last = last.next;

}

// Insert the new\_node at last node

last.next = new\_node;

}

// Return the list by head

return list;

}

// Method to print the LinkedList.

public static void printList(LinkedList<T> list)

{

Node currNode = list.head;

System.out.print("LinkedList: ");

// Traverse through the LinkedList

while (currNode != null) {

// Print the data at current node

System.out.print(currNode.data + " ");

// Go to next node

currNode = currNode.next;

}

}

// Driver code

public static void main(String[] args)

{

/\* Start with the empty list. \*/

LinkedList <String>list = new LinkedList();

vlist = insert(list, "1");

//

// \*\*\*\*\*\*INSERTION\*\*\*\*\*\*

//

// Insert the values

/\*list = insert(list, 1);

list = insert(list, 2);

list = insert(list, 3);

list = insert(list, 4);

list = insert(list, 5);

list = insert(list, 6);

list = insert(list, 7);

list = insert(list, 8);

\*/

// Print the LinkedList

printList(list);

}

}

ii) Implementation of Stacks in JAVA

public class Stack {

private int maxSize;

private int top;

private int array[];

public Stack(int max\_size) {

this.maxSize = max\_size;

this.top = -1; //initially when stack is empty

array = new int[max\_size];//type casting Object[] to V[]

}

//returns the maximum size capacity

public int getMaxSize() {

return maxSize;

}

//returns true if Stack is empty

public boolean isEmpty(){

return top == -1;

}

//returns true if Stack is full

public boolean isFull(){

return top == maxSize -1;

}

//returns the value at top of Stack

public int top(){

if(isEmpty())

return -1;

return array[top];

}

//inserts a value to the top of Stack

public void push(int value) throws overflow{

if(isFull()) {

throw new overflow("overflow");

}

array[++top] = value; //increments the top and adds value to updated top

}

//removes a value from top of Stack and returns

public int pop() throws underflow{

if(isEmpty())

throw new underflow("underflow");

return array[top--]; //returns value and top and decrements the top

}

public class underflow extends Exception

{

String m;

underflow(String z)

{

m=z;

}

public String toString(){

return ("Exception Occurred: "+m);

}

}

public class overflow extends Exception

{

String s;

overflow(String z)

{

s=z;

}

public String toString(){

return ("Exception Occurred: "+s);

}

}

}

class StackDemo {

public static void main(String[] args) {

Stack stack = new Stack(5);

System.out.print("Elements pushed in the Stack: ");

try

{

for (int i = 0; i < 5; i++) {

stack.push(i); //pushes 5 elements (0-4 inclusive) to the stack

System.out.print(i + " ");

}

}

catch (Stack.overflow o)

{

System.out.println("\n Stack is full? \n");

}

System.out.print("Elements popped from the Stack: ");

try

{

for (int i = 0; i < 5; i++) {

System.out.print(stack.pop()+" "); //pops all 5 elements from the stack and prints them

}

}

catch(Stack.underflow u)

{

System.out.println("\nStack is empty... \n");

}

}//end of main

}

iii) Implementation of queue in JAVA

public class Queue<V> {

private int maxSize;

private V[] array;

private int front;

private int rear;

private int currentSize;

/\*

Java does not allow generic type arrays. So we have used an

array of Object type and type-casted it to the generic type V.

This type-casting is unsafe and produces a warning.

Comment out the line below and execute again to see the warning.

\*/

@SuppressWarnings("unchecked")

public Queue(int maxSize) {

this.maxSize = maxSize;

array = (V[]) new Object[maxSize];

front = 0;

rear = -1;

currentSize = 0;

}

public int getMaxSize() {

return maxSize;

}

public int getCurrentSize() {

return currentSize;

}

public boolean isEmpty() {

return currentSize == 0;

}

public boolean isFull() {

return currentSize == maxSize;

}

public V top() {

return array[front];

}

public void add\_queue(V value) {

if (isFull())

return;

rear = (rear + 1) % maxSize; //to keep the index in range

array[rear] = value;

currentSize++;

}

public V remove\_queue() {

if (isEmpty())

return null;

V temp = array[front];

front = (front + 1) % maxSize; //to keep the index in range

currentSize--;

return temp;

}

}

class QueueDemo {

public static void main(String[] args) {

Queue<Integer> queue = new Queue<Integer>(5);

//equeue 12 4 16 7 19 at the end

queue.add\_queue(2);

queue.add\_queue(4);

queue.add\_queue(6);

queue.add\_queue(8);

queue.add\_queue(10);

//remove\_queue 2 elements from the start

queue.remove\_queue();

queue.remove\_queue();

//add\_queue 12 and 14 at the end

queue.add\_queue(12);

queue.add\_queue(14);

System.out.println("Queue:");

while(!queue.isEmpty()){

System.out.print(queue.remove\_queue()+" ");

}

Queue<String> qstr=new Queue<String>(2);

qstr.add\_queue("abc");

qstr.add\_queue("xyz");

System.out.println("String Queue:");

while(!qstr.isEmpty()){

System.out.print(qstr.remove\_queue()+" ");

}

}

}