**Singly Circular Linked List**

//----------------------------------Node Class------------------------//

public class Node {

private int data;

private Node next;

public Node(int data) {

this.data = data;

next = null;

}

public int getData() {

return data;

}

public void setData(int data) {

this.data = data;

}

public Node getNext() {

return next;

}

public void setNext(Node next) {

this.next = next;

}

}

//----------------------------CircularLinkedList Class------------------//

public class SCLL {

private Node head; //ref to first node of list

public SCLL()

{

this.head=null;//empty list creation

}

public Node getHead()

{

return this.head;

}

public void setHead(Node head)

{

this.head=head;

}

boolean insertByVal(int data){

Node newNode=new Node(data);

if(newNode==null)//mem alloc fails...issue in node creatioin

{

//System.out.println("error in node creation");

return false;

}

//chk if list is empty

if(this.head==null)//list is empty ...so new node becomes head

{

this.head=newNode;

newNode.setNext(head);

return true;//exit

}

//traverse list till last node

Node temp=head;

while(temp.getNext()!=head)

{

temp=temp.getNext();

}

//connect last node with new node n newnode with head

temp.setNext(newNode);

newNode.setNext(head);

return true;

}

void display() {

Node temp=head;

System.out.println();

do

{

System.out.print(temp.getData()+ " ");

temp=temp.getNext();

} while(temp!=head);

}

boolean insertByPos(int data,int pos){

Node newNode=new Node(data);

if(newNode==null)// issue in node creation

{

//System.out.println("error in node creation");

return false;

}

if(pos==1) {

//chk if list is empty

if(this.head==null)//list is empty ...so new node becomes head

{

head=newNode;

head.setNext(head);

return true;

}

else //non empty list

{

newNode.setNext(head);

Node temp=head;

while(temp.getNext()!=head)

{

temp=temp.getNext();

}

temp.setNext(newNode);

head=newNode;

return true;

}

}

else //pos other than first

{

Node prev=head;

for(int i=1;i<pos-1;i++)

{

if(prev.getNext()==head)//out of bound pos

return false;

prev=prev.getNext();

}

newNode.setNext(prev.getNext());

prev.setNext(newNode);

return true;

}

}

boolean deleteByVal(int data)

{

Node del,prev,temp;

del=prev=temp=head;

if(del.getData()==data) //chk match with head or first

{

//traverse list till last node

while(temp.getNext()!=head)

{

temp=temp.getNext();//forward by one node

}

//connect last node with second

temp.setNext(head.getNext());

head=head.getNext();

return true;

}

else

{

//locate deletable node by matching data

while(del.getData()!=data)

{

if(del.getNext()==head) //unmatched not found

{

return false;

}

prev=del;

del=del.getNext();

}

prev.setNext(del.getNext());

return true;

}

}

boolean deleteByPos(int pos)

{

Node del,prev,temp;

del=temp=prev=head;

if(head==null)//empty list

return false;

if(pos==1)

{

if(head.getNext()==head)//only one node in list

{

head=null;

return true;

}

//non empty list

//locate last node

while(temp.getNext()!=head)

{

temp=temp.getNext();

}

// connect last n second node

temp.setNext(head.getNext());

head=head.getNext();//update head

return true;

}

else //other than first pos

{

//set prev to pos-1

for(int i=1;i<pos-1;i++)

{

if(prev.getNext()==head)//out of bound

{

return false;

}

prev=prev.getNext();

}

del=prev.getNext();

prev.setNext(del.getNext());

return true;

}

}

}

}

**Stack**

//\*--------------Stack Class-----------------------\*/

public class Stack {

private int [] arr;

private int size;

private int top;

public Stack() {

size = 5;

arr = new int[5];

top = -1;

}

public Stack(int size) {

this.size = size;

arr = new int[size];

top = -1;

}

public boolean isEmpty() {

return top == -1;

}

public boolean isFull() {

return top == (size - 1);

}

public boolean push(int data) {

if( isFull() ) {

return false;

}

arr[++top] = data;

return true;

}

public int pop() {

if(isEmpty()) {

return -999;

}

return arr[top--];

}

public int peek() {

if(isEmpty()) {

return -999;

}

/\*int data = pop();

push(data);

return data;\*/

return arr[top];

}

/\*public void display() {

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

System.out.print(arr[i] + " ");

}

}\*/

}

//\*----------------------TesterMain Class------------------\*/

public class Main {

public static void main(String [] args) {

Stack s = new Stack(4);

System.out.println("Pop: " + s.pop());

System.out.println("Push: " + s.push(0) );

System.out.println("Push: " + s.push(20) );

System.out.println("Push: " + s.push(30) );

System.out.println("Push: " + s.push(40) );

System.out.println("Push: " + s.push(50) );

System.out.println("peek: " + s.peek());

System.out.println("Pop: " + s.pop());

/\*System.out.println("Pop: " + s.pop());

System.out.println("Pop: " + s.pop());

System.out.println("Pop: " + s.pop());

System.out.println("Pop: " + s.pop());\*/

}

}

**Stack Implementation using Linked List :**

//\* ---------------------Stack Node Class-----------------\*/

public class Node {

//data members

private int data;

private Node next;

public Node()//def

{

this.data=0;

this.next=null;

}

public Node(int data)//param.

{

this.data=data;

this.next=null;

}

int getData()

{

return this.data;

}

Node getNext()

{

return this.next;

}

void setData(int data )//local var

{

this.data=data;

}

void setNext(Node next) //localvar

{

this.next=next;

}

}

//\*---------------Stack Class with Operations--------------------\*/

public class StackLiLi {

private Node top;

public StackLiLi()

{

this.top=null;

}

boolean push(int data)

{

Node newNode=new Node(data);

if(newNode==null)

{

return false;

}

//chk if stk is empty

if(top==null)

{

top=newNode;//set newnode as first top

return true;

}

//if non empty stack then make newnode as top

newNode.setNext(top);

top=newNode;

return true;

}

int pop()

{

if(top==null) //empty stack

return -999;

//non empty stack

int val=top.getData();

top=top.getNext();

return val;

}

int peek()

{

if(top==null) //empty stack

return -999;

//non empty stack

int val=top.getData();

return val;

}

void display()

{

Node temp=top;

System.out.println();

while(temp!=null)

{

System.out.print(temp.getData()+ " " );

temp=temp.getNext();

}

}

}

//\*-------------------------Tester Main ---------------------\*/

package com.stklili;

public class TesterStack {

public static void main(String[] args) {

StackLiLi s1=new StackLiLi();//default is 5 but we can pass size also

s1.push(10);

s1.push(20);

s1.push(30);

System.out.println("peek : "+s1.peek());

s1.push(40);

s1.push(50);

s1.display();

s1.push(60);

System.out.println("popped: "+s1.pop());

System.out.println("popped: "+s1.pop());

System.out.println("popped: "+s1.pop());

System.out.println("popped: "+s1.pop());

System.out.println("popped: "+s1.pop());

System.out.println("popped: "+s1.pop());

s1.display();

}

}

**Linear Queue implementation using Array**:

//\*----------------------------------Queue Class--------------------\*/

public class Queue {

private int [] arr;

private int front, rear;

private int size;

public Queue(int size) {

this.size = size;

this.arr = new int[size];

front = rear = -1;

}

public boolean isEmpty() {

return (front == -1 && rear == -1) || (front > rear);

}

public boolean isFull() {

return rear == (size -1);

}

public boolean insert(int data) {

if(isFull()) {

return false;

}

arr[++rear] = data;

if(front == -1) {

front = 0;

}

//bulk move operation

/\*

if(rear == size - 1 && front > 0 ) {

//code to shift the data to the front of the queue

}

\*/

return true;

}

public int delete() {

if(isEmpty()) {

return -999;

}

return arr[front++];

}

public void display() {

for(int i = front; i <= rear; i++) {

System.out.print( arr[i] + " ");

}

}

}

//\*------------------------Tester Main Class------------\*/

public class Main {

public static void main(String [] args) {

CircularQueueNew cq = new CircularQueueNew(5);

System.out.println("ins : " + cq.insert(10));

System.out.println("ins : " + cq.insert(20));

System.out.println("ins : " + cq.insert(30));

System.out.println("ins : " + cq.insert(40));

System.out.println("ins : " + cq.insert(50));

System.out.println("ins : " + cq.insert(60));

System.out.println("del : " + cq.delete());

System.out.println("ins : " + cq.insert(60));

System.out.println("del : " + cq.delete());

System.out.println("ins : " + cq.insert(70));

/\*Queue q = new Queue(5);

System.out.println("Ins: " + q.insert(10) );

System.out.println("Ins: " + q.insert(20) );

System.out.println("Ins: " + q.insert(30) );

System.out.println("Del : " + q.delete());

System.out.println("Del : " + q.delete());

System.out.println("Del : " + q.delete());

System.out.println("Del : " + q.delete());

System.out.println("is Empty : " + q.isEmpty() +" is Full : " + q.isFull());

System.out.println("Ins: " + q.insert(40) );

System.out.println("Ins: " + q.insert(50) );

System.out.println("Ins: " + q.insert(60) );

System.out.println("Del : " + q.delete());

System.out.println("Del : " + q.delete());

System.out.println("Del : " + q.delete());

System.out.println("is Empty : " + q.isEmpty() +" is Full : " + q.isFull());

\*/ }

}

**Circular Queue implementation using Array:**

//\*--------------------------------- Circular Queue Class------------------------\*/

public class CQueue {

private int []arr;

private int front,rear;

private int size;

public CQueue()

{

this.size=5;

arr=new int [this.size];

front=-1;

rear=-1;

}

public CQueue(int size)

{

this.size=size;

arr=new int [this.size];

front=-1;

rear=-1;

}

public boolean isEmpty()

{

return (front==rear);

}

public boolean isFull()

{

return ((front==-1&& rear==size-1)||((rear+1%size)==front));

}

boolean insert(int data)

{

if(isFull())

return false;

else

{ rear=(rear+1)%size;//cyclic increment

arr[rear]=data;

return true;

}

}

public int delete()

{

int val=-9999;

if(isEmpty())

return val; //unpredicted data

front=(front+1)%size;

val = arr[front];

return val;

}

public void display()

{

int i=front+1;

while(i!=rear)

{

System.out.print(arr[i]+ " ");

i=(i+1)%size;

}

System.out.println(arr[i]);

}

public int peek()

{

if(isEmpty())

return -9999; //unpredicted data

/\*

\* int val=arr[front++]; return val;

\*/

return(arr[front]);

}

}

//\*--------------------------------- Tester Main Class------------------------\*/

public class TesterLinQArr {

public static void main(String[] args) {

CQueue Q1=new CQueue(); //default size 5

Q1.insert(10);

Q1.insert(20);

Q1.insert(30);

//System.out.println(Q1.peek());

Q1.insert(40);

Q1.insert(50);

Q1.display();

System.out.println();

System.out.println("deleted :"+Q1.delete());

System.out.println("deleted :"+Q1.delete());

Q1.insert(60);

Q1.insert(70);//bubble prob

Q1.display();

System.out.println("deleted :"+Q1.delete());

Q1.insert(70);

Q1.display();

}

}