PRIORITY QUEUE :

package day15;

class PriorityQueue{

int size, arr[],capacity;

public PriorityQueue(int capacity)

{

this.capacity = capacity;

arr = new int[capacity];

size=0;

}

public void insert(int val)

{

if(size==capacity)

{

System.***out***.println("queue is full");

return;

}

arr[size]=val;

size++;

System.***out***.println("inserted:"+val);

}

public int remove()

{

int max=arr[0];

if(size==0)

{

System.***out***.println("queue is empty");

return 0;

}

for(int i=1;i<size;i++)

{

if(arr[i]>max)

{

max=arr[i];

}

}

int highPriority=max;

for(int i=0;i<size-1;i++)

{

arr[i]=arr[i+1];

size--;

}

return highPriority;

}

public boolean isEmpty()

{

return size==0;

}

public void display()

{

if(size==0)

{

System.***out***.println("size is empty");

return ;

}

for(int i=0;i<size;i++)

{

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

}

}

}

public class task2 {

public static void main(String args[])

{

PriorityQueue pq = new PriorityQueue(5);

pq.insert(10);

pq.insert(40);

pq.insert(50);

System.***out***.println(pq.remove());

System.***out***.println(pq.isEmpty());

pq.display();

}

}

Double Ended Queue :

package day15;

class Dqueue

{

int front,rear,capacity,size,arr[];

public Dqueue(int capacity)

{

this.capacity=capacity;

front=0;

rear=0;

arr=new int[capacity];

size=0;

}

public boolean isFull()

{

return size==capacity;

}

public boolean isEmpty()

{

return size==0;

}

public void addRear(int val)

{

if(isFull())

{

System.***out***.println("full");

}

arr[rear]=val;

rear=(rear+1)%capacity;

size++;

}

public void addFront(int val)

{

if(isFull())

{

System.***out***.println("full");

return;

}

front=((front-1)+capacity);

arr[front]=val;

size++;

}

public void display()

{

if(isEmpty())

{

System.***out***.println("empty");

return;

}

for(int i=0;i<size;i++)

{

System.***out***.println(arr[front+i%capacity]);

}

}

public int removeRear()

{

if(isEmpty())

{

System.***out***.println("Dequeue is empty");

}

rear=(rear-1+capacity)%capacity;

int remove = arr[rear];

size--;

return remove;

}

public int removeFront()

{

if(isEmpty())

{

System.***out***.println("Dequeue is empty");

}

int remove =arr[front];

size--;

return remove;

}

}

public class task4 {

public static void main(String args[])

{

Dqueue d = new Dqueue(5);

d.addRear(10);

d.addRear(1);

d.addRear(14);

d.display();

d.addFront(10);

System.***out***.println( d.removeRear());

System.***out***.println( d.removeFront());

10

1

14

14

10

Blocking queue :

package day15;

class BlockingQueue{

int front,rear,capacity,size,arr[];

public BlockingQueue(int capacity)

{

this.capacity=capacity;

front=0;

rear=-1;

arr=new int[capacity];

size=0;

}

public synchronized void enqueue(int val)

{

if(size==capacity)

{

System.***out***.println("full");

return;

}

while(size==capacity)

{

try {

wait();

}

catch(InterruptedException e) {

System.***out***.println(e);

}

}

rear = (rear + 1) % capacity;

arr[rear] = val;

size++;

System.***out***.println(val + "--> produced");

notifyAll();

}

public synchronized void dequeue()

{

while(size==0)

{

System.***out***.println("empty");

try {

wait();

}

catch(InterruptedException e)

{

System.***out***.println(e);

}

}

int remove=arr[front];

front=(front+1)%capacity;

size--;

notifyAll();

System.***out***.println("dequeued :"+remove);

}

}

public class task3 {

public static void main(String args[]) throws InterruptedException

{

BlockingQueue b = new BlockingQueue(5);

Thread producer = new Thread(()->{

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

{

b.enqueue(i);

try {

Thread.*sleep*(1000);

}

catch(InterruptedException e) {

System.***out***.println(e);

}

}});

Thread consumer = new Thread(()->{

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

{

b.dequeue();

try {

Thread.*sleep*(1000);

}

catch(InterruptedException e) {

System.***out***.println(e);

}

}});

producer.start();

Thread.*sleep*(1000);

consumer.start();

}

}

Arr size 10

Arr value inside 🡪 for loop

F = 0 s=0

While(size > 1)

{

front = front + (k-1) % size ;

}

Linked list

package day15;

class Node

{

int data ;

Node next;

public Node(int data)

{

this.data=data;

next=null;

}

}

class LinkedList{

Node head = null;

Node tail = null;

public void insert(int data)

{

Node nn = new Node(data);

if(head==null)

{

head=nn;

tail=nn;

}

else {

tail.next=nn;

tail=nn;

}

}

public void traverse()

{

Node temp = head;

while(temp!=null)

{

System.***out***.print(temp.data+"-->");

temp=temp.next;

}

System.***out***.println("null");

}

public void insertAtHead(int data)

{

Node nn = new Node(data);

if(head==null)

{

head=tail=nn;

}

nn.next=head;

head=nn;

}

public void insertAtPosition(int data,int pos)

{

Node temp = head;

Node nn = new Node(data);

if(pos==1)

{

insertAtHead(data);

}

for(int i=1;i<pos-1 && temp!=null;i++)

{

temp=temp.next;

}

nn.next=temp.next;

temp.next=nn;

if(nn.next==null)

{

tail=nn;

}

}

public int search(int data)

{

Node temp=head;

int i=0;

while(temp!=null)

{

i++;

if(temp.data==data)

{

System.***out***.println("search found..");

return i;

}

}

return -1;

}

public void reverse()

{

Node current ;

Node pre = null;

Node next ;

current = head;

while(current!=null)

{

next=current.next;

current.next=pre;

pre=current;

current=next;

}

}

public void delete(int data)

{

Node temp = head;

Node pre = null;

while(temp!=null) {

if(temp.data==data)

{

if(pre==null)

{

temp=temp.next;

}

else {

pre.next=temp.next;

}

if(temp==tail)

{

tail=pre;

return;

}

}

}

System.***out***.println("not found");

}

}

public class task5 {

public static void main(String args[])

{

LinkedList l = new LinkedList();

l.insert(0);

l.insert(10);

l.insert(20);

l.traverse();

l.insertAtHead(20);

l.traverse();

l.insertAtPosition(40,1);

}

}