/\*

Chris Clemmons

CSC310

Professor Kim

2/23/19

\*/

package hw4;

class MyCircularDeque{

int[] queue; //creates queue

int size = -1; //sets size

int front =-1; //sets front position

int rear = -1; //sets rear position

public MyCircularDeque(int k){ //creates queue with size

this.queue =new int[k]; //sets queue to size k

size =k; //sets size to k

for(int i = 0; i <k; i++){ // sets items to -1

queue[i]=-1;

}

}

void insertFront(int k){ //inserts front

// checks if queue is full

if (isFull()){

System.out.println("Full"); //returns full

return;

}

// sets front and rear to 0 if empty

if (front == -1) {

front = 0;

rear = 0;

}

// front is set to first position of queue

else if (front == 0)

front = size - 1 ;

else // decreases end by 1

front = front-1;

// adds current item to queue

queue[front] = k ;

System.out.println("Insert Front: "+k);

}

void insertLast(int k){ //insert last

//checks if queue is full

if (isFull()) {

System.out.println("Full");

return;

}

// sets front and rear to 0 if queue is empty

if (front == -1){

front = 0;

rear = 0;

}

// sets rear to last position fo queue

else if (rear == size-1)

rear = 0;

// increases rear by 1

else

rear = rear+1;

// adds current item to rear of queue

queue[rear] = k ;

System.out.println("Insert Rear: "+k);

}

int getRear(){

// checks if queue is empty or not

if(isEmpty() || rear < 0) {

System.out.println("Empty");

return -1 ;

}

return queue[rear]; // returns rear

}

int getFront(){

// check whether Deque is empty or not

if (isEmpty())

{

System.out.println(" Underflow");

return -1 ;

}

return queue[front];

}

boolean isFull(){

return ((front == 0 && rear == size-1)||

front == rear+1);

}

boolean isEmpty(){

return (front == -1);

}

void deleteLast(){

//checks if queue is empty or not

if (isEmpty()) {

System.out.println("Empty");

return ;

}

// checks if queue has only onr element

if (front == rear) {

front = -1;

rear = -1;

}

else if (rear == 0)

rear = size-1;

else

rear = rear-1;

}

void deleteFront(){

// checks if queue is empty or not

if (isEmpty()) {

System.out.println("Empty");

return ;

}

// checks if queue only has one element

if (front == rear) {

front = -1;

rear = -1;

}

else

// sets intial position

if (front == size -1)

front = 0;

else // increases by one

// sets new value for front

front = front+1;

}

}

class myQueue{

int []queue= {0,0,0,0,0}; // sets initial queue size

int enqueue(int k){// enqueue method

for(int i =0; i<queue.length; i++){ // runs through length of queue

if(queue[i]==0){// if element is 0 set to k

queue[i]=k;

break;

}

}

return k;

}

void dequeue(){ // dequeue method

for(int i =queue.length-1; i>=0; i--){ // runs through length of queue

if(queue[i]!=0){//if spot is not empty

System.out.println("Dequeue: "+ queue[i]); //prints out operation

queue[i]=0; // sets spot to 0

break; // ends operation

}

}

}

void first(){//prints out first item in queue

System.out.println("First: "+queue[0]);

}

void len(){//prints out length of queue

for(int i =queue.length-1; i>=0; i--){ // runs through queue to see when it ends

if(queue[i]!=0){ // prints out length

System.out.println("Length: "+(i+1));

break;

}

}

}

boolean isEmpty(){//checks if queue is empty

if(queue[0]==0){//if item at 0 = 0 return true

return true;

}else{ // if else, return false

return false;

}

}

boolean search(int k){ // checks whether item is in queue

int bool= 0;

for(int i =queue.length-1; i>0; i--){// runs through length of queue

if(queue[i]==k){

bool = 1; // if item is in queue, set to 1

}

}

if(bool==0){// if 0, return false

return false;

}

else{// if else return true

return true;

}

}

}

public class HW4 {

static int[] linkedList(int[] list1, int[] list2){ // linked list method

int k=list1.length+list2.length;

// sets new list length to accomodate both lists

int [] newList =new int [list1.length+list2.length];

int position1 = 0; // position of list 1

int position2 = 0; // position of list 2

int l = 1; //int to check whether its adding from list 1 or 2

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

if(list1.length>position1 && l==1){//if set to list one and list one isn't empty

newList[i] = list1[position1];

position1++; // increases position in list 1

l=2;//set to list 2

continue;

}else{//set to list 2

l=2;

}

if(list2.length>position2 && l==2){ // if set to list 2 and list 2 isn't empty

newList[i]= list2[position2];

position2++;// increses position in list2

l=1;//sets back to list 1

continue;

}

else{

if(list1.length>position1){ // if list 2 is empty, add form list 1

l=1;

newList[i]= list1[position1];

position1++;// increases position in list 1

}

}

}

return newList; // returns new list

}

public static void main(String[] args) {

// MyCircularDequeue operations

MyCircularDeque circularDeque = new MyCircularDeque(3);

circularDeque.insertLast(1);

circularDeque.insertLast(2);

circularDeque.insertFront(3);

circularDeque.insertFront(4);

System.out.println("rear: "+ circularDeque.getRear());

circularDeque.isFull();

circularDeque.deleteLast();

circularDeque.insertFront(4);

System.out.println("front: "+ circularDeque.getFront());

// linked list operations

int [] list1 = new int [3];

int list2[]= new int [3];

list1 [0]=1;

list1 [1]=2;

list1 [2]=4;

list2[0]=1;

list2[1]= 3;

list2[2]=4;

int[] list3 =linkedList(list1,list2);

//print list 3

for(int i =0; i<list3.length;i++){

System.out.print(list3[i]+" ");

}

System.out.println();

// myQueue operations

myQueue q = new myQueue();

System.out.println("Enqueue: "+ q.enqueue(1));

System.out.println("Enqueue: "+ q.enqueue(2));

System.out.println("Enqueue: "+ q.enqueue(3));

System.out.println("Enqueue: "+ q.enqueue(4));

System.out.println(q.search(4));

q.len();

q.dequeue();

q.dequeue();

q.len();

q.first();

q.dequeue();

q.dequeue();

}

}