Data Structures and Algorithms – Assignment 6

Queue, Circular Queue, Deque

1. Declare a class (QueueInt.java) for integer QueueInt with three attributes:

   (a) An array 'arr' of size 5.

   (b) Two variables front and rear initialized to -1.

```
2 usages
public class QueueInt {
    1 usage
    private int[] queue;
    2 usages
    private int front;
    2 usages
    private int rear;
    1 usage
    QueueInt(){
        queue= new int[5];
        front=-1;
        rear=-1;
    }
```

Test cases:

a) Write a separate test driver class (Test.java) and create an object instance of QueueInt class.

QueueInt qi = new QueueInt ();

Compile both the .java files and run Test.java. Ensure no errors.

```
Q1.java   QueueInt.java
public class Q1 {
    public static void main(String[] args) {
        QueueInt q = new QueueInt();
    }
}
```

```
Process finished with exit code 0
```

b) Now try to access the 'front' and ‘rear’ attribute of QueueInt from Test.java directly.

System.out.println("Queue Front is”+qi.front+ ” and Rear is " + qi.rear);

Compile and execute.

```
Q1.java   QueueInt.java
public class Q1 {
    public static void main(String[] args) {
        QueueInt qi = new QueueInt();
        System.out.println("Queue Front is "+qi.getFront() + " and Rear is "+qi.getRear());
    }
}
```

```
C:\Users\aadit\.jdks\openjdk-19.0.2\bin\j
Queue Front is -1 and Rear is -1

Process finished with exit code 0
```

2. Add a default constructor that will create the queue of some standard size (say 10) in case user does not give the size.

```
QueueInt () {
        arr = new int[10];
        front = -1;        rear=-1;
        }
```

Test cases:

```
QueueInt qi = new QueueInt ();
System.out.println(qi.arr.length);
```

```java
public class QueueInt {
    2 usages
    public int[] arr;
    2 usages
    private int front;
    2 usages
    private int rear;
    2 usages
    QueueInt(){
        arr= new int[10];
        front=-1;
        rear=-1;
    }
```

QueueInt.java Q2.java

```java
public class Q2 {
    public static void main(String[] args) {
        QueueInt qi = new QueueInt();
        System.out.println(qi.arr.length);
    }
}
```

Q2

```
C:\Users\aadit\.jdks\openjdk-19.0.2\
10

Process finished with exit code 0
```

3. Add a parameterized constructor that will create the queue with specified size.

```
QueueInt (int sz) {
    arr = new int[sz];
front = -1;        rear=-1;
}
```

Test cases:

```
QueueInt qi2 = new QueueInt (15);
System.out.println(qi2.arr.length);
```

QueueInt.java Q3.java

```
public class Q3 {
    public static void main(String[] args) {
        QueueInt qi = new QueueInt( size: 15);
        System.out.println(qi.arr.length);
    }
}
```

Q3

```
C:\Users\aadit\.jdks\openjdk-19.0.2\bin
15

Process finished with exit code 0
```

4. Add print function to QueueInt .java that will print the contents of the queue. It should be public.

public void print() {

// Your logic here

// Scan through the array from front to rear and the print the values with tab space in between.

}

```
public void print(){
    if (isEmpty()){
        System.out.println("Queue is empty");
        return;
    }
    for(int i=front;i<=rear;i++){
        System.out.print(arr[i]+"\t");
    }
    System.out.println();
}
```

5. Implement enqueue() method first without checking the array length limit.

public void enquue(int item) {

// Your logic here

}

```
public void enqueue(int data){
    if(isEmpty()){
        front++;
        rear++;
    }
    else{
        rear++;
    }
    arr[rear]=data;
}
```

Test cases:

(a) Write test file and try invoking enqueue operations and check.

QueueInt qi = new QueueInt ();

qi.enqueue(100);

qi.print(); qi.enqueue(200);

qi.print();

QueueInt.java q5.java

```
public class q5 {
    public static void main(String[] args) {
        QueueInt qi = new QueueInt();
        qi.enqueue( data: 100);
        qi.print();
        qi.enqueue( data: 200);
        qi.print();
    }
}
```

```
q5
C:\Users\aadit\.jdks\openjdk-19.0.2\bin\
100
100 200

Process finished with exit code 0
```

(b) Write test file that tries to enqueue beyond Queue capacity

QueueInt qi = new QueueInt ();

.....,.

qi.enqueue(900);

qi.print();

qi.enqueue(300);

qi.print();

The execution will abort as soon as the last enqueue is invoked.

ArrayIndexOutOfBoundsException.

QueueInt.java q5.java

```java
public class q5 {
    public static void main(String[] args) {
        QueueInt qi = new QueueInt();
        qi.enqueue( data: 100);
        qi.print();
        qi.enqueue( data: 200);
        qi.print();
        qi.enqueue( data: 300);
        qi.print();
        qi.enqueue( data: 400);
        qi.print();
        qi.enqueue( data: 500);
        qi.print();
        qi.enqueue( data: 600);
        qi.print();
        qi.enqueue( data: 700);
        qi.print();
        qi.enqueue( data: 800);
        qi.print();
        qi.enqueue( data: 900);
        qi.print();
        qi.enqueue( data: 1000);
        qi.print();
        qi.enqueue( data: 1100);
        qi.print();
    }
}
```

q5

```
C:\Users\aadit\.jdks\openjdk-19.0.2\bin\java.exe "-javaa
100
100 200
100 200 300
100 200 300 400
100 200 300 400 500
100 200 300 400 500 600
100 200 300 400 500 600 700
100 200 300 400 500 600 700 800
100 200 300 400 500 600 700 800 900
100 200 300 400 500 600 700 800 900 1000
Exception in thread "main" java.lang.ArrayIndexOutOfBoun
    at QueueInt.enqueue(QueueInt.java:41)
    at q5.main(q5.java:24)

Process finished with exit code 1
```

6. Implement the check to enqueue() method. Don't add an item to the array if rear= =n-1. Instead print "can't enqueue" message.

```
public void enqueue(int item) {

        // Your enhanced logic here

}
```

Test cases:

```
public void enqueue(int data){
    if(isFull()){
        System.out.println("Queue is full");
        return;
    }
    else if(isEmpty()) {
        front++;
    }
    rear++;
    arr[rear]=data;
}
```

(a) Run test file. No exception will be thrown this time around.

```
C:\Users\aadit\.jdks\openjdk-19.0.2\bin\java.exe
100
100 200
100 200 300
100 200 300 400
100 200 300 400 500
100 200 300 400 500 600
100 200 300 400 500 600 700
100 200 300 400 500 600 700 800
100 200 300 400 500 600 700 800 900
100 200 300 400 500 600 700 800 900 1000
Queue is full
100 200 300 400 500 600 700 800 900 1000

Process finished with exit code 0
```

7. Add a getter method getFront() which returns the front element in the queue.

```
public int getFront() {
 // return arr[front];
}
```

```
public int getFront(){
    return arr[front];
}
```

Test cases:

(a) Invoke getFront() from test file and print the top.

```
System.out.println(qi2.getFront());
```

```
qi.print();
qi.enqueue( data: 1000);
qi.print();
System.out.println("Front: "+qi.getFront());
```

```
q5 ×
C:\Users\aadit\.jdks\openjdk-19.0.2\bin\java
100
100 200
100 200 300
100 200 300 400
100 200 300 400 500
100 200 300 400 500 600
100 200 300 400 500 600 700
100 200 300 400 500 600 700 800
100 200 300 400 500 600 700 800 900
100 200 300 400 500 600 700 800 900 1000
Front: 100
```

8. Now implement dequeue() method that removes the front item in the queue and returns it. First without lower bound checking logic. public int dequeue() {

// Your logic here

}

```
no usages
public int dequeue(){
    if(front==rear){
        int temp=arr[front];
        front=-1;
        rear=-1;
        return temp;
    }
    else{
        int temp=arr[front];
        front++;
        return temp;
    }
}
```

Test cases:

(a) Write Test5.java to enqueue and dequeue few items to check it's working.

int item = qi.dequeue();

qi.print();

```
QueueInt.java   q5.java
public class q5 {
    public static void main(String[] args) {
        QueueInt qi = new QueueInt();
        qi.enqueue( data: 100);
        qi.print();
        qi.enqueue( data: 200);
        qi.print();
        qi.enqueue( data: 300);
        qi.print();
        qi.dequeue();
        qi.print();

    }
}
```

```
Run: Q8
C:\Users\aadit\.jdks\openjdk-19.0.2\bin
100
100 200
100 200 300
200 300

Process finished with exit code 0
```

(b) Write test file to dequeue more items than what were enqueued.

int item1 = qi.dequeue();

qi.print();

....

The last call to dequeue() should throw Array out of bounds exception.

```
QueueInt.java  q5.java  Q8.java
public class Q8 {
    public static void main(String[] args) {
        QueueInt qi = new QueueInt();
        qi.enqueue( data: 100);
        qi.print();
        qi.enqueue( data: 200);
        qi.print();
        qi.enqueue( data: 300);
        qi.print();
        qi.dequeue();
        qi.print();
        qi.dequeue();
        qi.print();
        qi.dequeue();
        qi.print();
        qi.dequeue();
    }
}
```

```
Run: Q8
C:\Users\aadit\.jdks\openjdk-19.0.2\bin\java.exe "-javaagent:C:\Progr
100
100 200
100 200 300
200 300
300
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException
    at QueueInt.print(QueueInt.java:26)
    at Q8.main(Q8.java:15)

Process finished with exit code 1
```

9. Now implement the check for front=rear=-1 and print "can't pop" message. Run test file. Note that no exception will be thrown this time.

```
public int dequeue(){
    if(isEmpty()){
        System.out.println("Queue is empty");
        return -1;
    }
    else if(front==rear){
        int temp=arr[front];
        front=-1;
        rear=-1;
        return temp;
    }
    else{
        int temp=arr[front];
        front++;
        return temp;
    }
}
```

```
Run:  Q8
C:\Users\aadit\.jdks\openjdk-19.0.2\bin\
100
100 200
100 200 300
200 300
300
Queue is empty
Queue is empty

Process finished with exit code 0
```

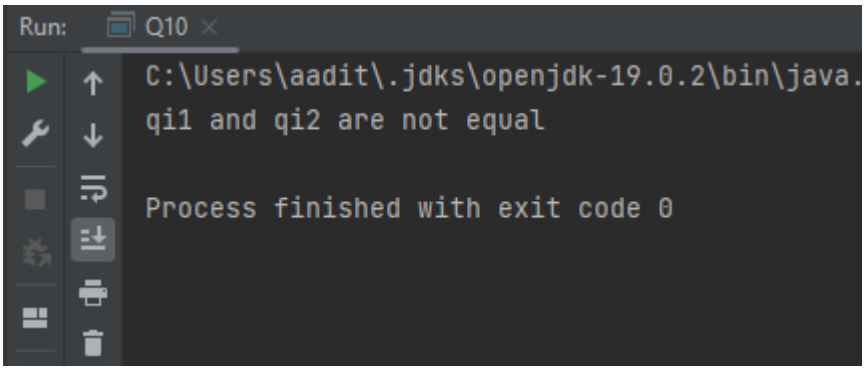
10. You can't check if contents of two queues are same by using = =.

Test cases:

(a) Write the test file.

```
QueueInt qi1 = new QueueInt ();
QueueInt qi2 = new QueueInt ();
qi1.enqueue(100);
qi2.enqueue(100);
qi1.enqueue(200);
qi2.enqueue(200);
if (qi1 == qi2)
System.out.println("Both qi1 and qi2 are same");
else
System.out.println("Both qi1 and qi2 are not the same");
```

Run it and check. It will print qi1 and qi2 are not same. Why?



Because = = operator will only compare 2 addresses. Then how to check the contents?

Creating a function to check is both values are same or not.

11. Implement equals() method which will first compare the elements of two queues. If not same, return false. If same, scan through arrays of both queues to check if each item is one queue is same as an item in another queue. If so, return true. Else return false.

```
public boolean equals(Queue another) {
        // Your logic here
}
```

```java
public boolean equals(QueueInt q){
    if(this.length()!=q.length()) return false;
    for(int i=0;i<this.length();i++){
        if(this.arr[i]!=q.arr[i]) return false;
    }
    return true;
}
```

Test cases:

(a) Now run test file. It will print qi1 and qi2 are same since the contents are same.

```java
public class Q11 {
    public static void main(String[] args) {
        QueueInt qi1 = new QueueInt();
        QueueInt qi2 = new QueueInt();
        qi1.enqueue( data: 100);
        qi2.enqueue( data: 100);
        qi1.enqueue( data: 200);
        qi2.enqueue( data: 200);
        if (qi1.equals(qi2)) {
            System.out.println("qi1 and qi2 are equal");
        } else {
            System.out.println("qi1 and qi2 are not equal");
        }
    }
}
```

```
Run: Q11
C:\Users\aadit\.jdks\openjdk-19.0.2\bin
qi1 and qi2 are equal

Process finished with exit code 0
```

(b) Now write test file to do same number of enqueues but contents different.

QueueInt qi1 = new QueueInt (5);

QueueInt qi2 = new QueueInt (5);

qi1.enqueue(100);

qi2.enqueue(100);

qi1.enqueue(200);

qi2.enqueue(300); if (code)

System.out.println("Both qi1 and qi2 are same");

else

System.out.println("Both qi1 and qi2 are not the same");

QueueInt.java Q10.java Q11.java Q8.java

```java
public class Q11 {
    public static void main(String[] args) {
        QueueInt qi1 = new QueueInt();
        QueueInt qi2 = new QueueInt();
        qi1.enqueue( data: 100);
        qi2.enqueue( data: 100);
        qi1.enqueue( data: 200);
        qi2.enqueue( data: 300);
        if (qi1.equals(qi2)) {
            System.out.println("qi1 and qi2 are equal");
        } else {
            System.out.println("qi1 and qi2 are not equal");
        }
    }
}
```

Run: Q11

```
C:\Users\aadit\.jdks\openjdk-19.0.2\bin
qi1 and qi2 are not equal

Process finished with exit code 0
```

12. Implement a circular queue by following the procedure given in question from 1 to 11.

```
public class CircularQueue {
    8 usages
    private int[] arr;
    13 usages
    private int front;
    10 usages
    private int rear;
    no usages
    CircularQueue(){
        arr= new int[10];
        front=-1;
        rear=-1;
    }
    no usages
    CircularQueue(int size){
        arr= new int[size];
        front=-1;
        rear=-1;
    }
```

```
no usages
public int getFront(){
    return arr[front];
}
no usages
public void print(){
    if (isEmpty()){
        System.out.println("Queue is empty");
        return;
    }
    for(int i=front;i<=rear;i++){
        System.out.print(arr[i]+"\t");
    }
    System.out.println();
}
1 usage
public boolean isFull(){
    return ((rear+1) % length() == front) ? true : false;
}
```

```java
public boolean isEmpty(){
    if ((front == -1) && (rear == -1)) return true;
    return false;
}
no usages
public void enqueue(int data){
    if(isFull()){
        System.out.println("Queue is full");
        return;
    }
    else if(isEmpty()) {
        front++;
    }
    rear=(rear+1)%length();
    arr[rear]=data;
}
```

```java
    public int dequeue(){
        if(isEmpty()){
            System.out.println("Queue is empty");
            return -1;
        }
        else if(front==rear){
            int temp=arr[front];
            front=-1;
            rear=-1;
            return temp;
        }
        else{
            int temp=arr[front];
            front=(front+1)%length();
            return temp;
        }
    }

}

//65 97 100 105 116 104 121 97 110 32 82 97 106 117
```

13. Implement the following methods in the above circular queue:

(a) splitq(), to split a queue into two queues so that all items in odd positions are in one queue and those in even positions are in another queue.

```
public class Q13 {
    public static void main(String[] args) {
        CircularQueue q1 = new CircularQueue();
        q1.enqueue( data: 10);
        q1.enqueue( data: 20);
        q1.enqueue( data: 30);
        q1.enqueue( data: 40);
        q1.enqueue( data: 50);
        System.out.print("Queue 1:");
        q1.print();
        CircularQueue q2 = new CircularQueue();
        q1.splitq(q2);
        System.out.print("Queue 1:");
        q1.print();
        System.out.print("Queue 2:");
        q2.print();

    }
}
```

```
public void splitq(CircularQueue q2){
    CircularQueue q3=new CircularQueue(length());
    for (int i=0;i<length();i++){
        if ((i+1)%2==0) q2.enqueue(arr[i]);
        else q3.enqueue(arr[i]);
    }
    this.arr = q3.arr;
}
```

```
C:\Users\aadit\.jdks\openjdk-19.0.2\bi
Queue 1:10  20  30  40  50
Queue 1:10  30  50  0   0
Queue 2:20  40  0   0   0

Process finished with exit code 0
```

(b) getminElement () to return the minimum element in a queue.

```
no usages
public int getMinElement(){
    CircularQueue q2=new CircularQueue(length());
    int min=arr[front];
    while (!isEmpty()){
        int temp=dequeue();
        if (temp<min) min=temp;
        q2.enqueue(temp);
    }
    this.arr=q2.arr;
    return min;
}
```

```
public class Q13 {
    public static void main(String[] args) {
        CircularQueue q1 = new CircularQueue();
        q1.enqueue( data: 10);
        q1.enqueue( data: 20);
        q1.enqueue( data: 3);
        q1.enqueue( data: 40);
        q1.enqueue( data: 50);
        System.out.print("Queue 1:");
        q1.print();
        System.out.println("Min: " + q1.getMinElement());

    }
}
```

```
Queue 1:10  20  3   40  50
Min: 3

Process finished with exit code 0
```

14. Implement the following operations on Deque using a circular array.

insertFront(): Adds an item at the front of Deque.

insertLast(): Adds an item at the rear of Deque.

deleteFront(): Deletes an item from front of Deque.

deleteLast(): Deletes an item from rear of Deque.

getFront(): Gets the front item from queue. getRear(): Gets the last item from queue.

isEmpty(): Checks whether Deque is empty or not.

isFull(): Checks whether Deque is full or not.

display(): Display queue elements starting from front to rear

```
public class Deque {
    19 usages
    Node front;
    18 usages
    Node rear;
    3 usages
    int capacity;
    12 usages
    int size;
    no usages
    Deque(){
        front=null;
        rear=null;
        capacity=-1;
        size=0;
    }
    no usages
    Deque(int capacity){
        front=null;
        rear=null;
        this.capacity=capacity;
        size=0;
    }
    no usages
    public int length(){
        return size;
    }
    7 usages
    public boolean isEmpty(){
        if (front==null && rear==null) return true;
        return false;
    }
    2 usages
    public boolean isFull(){
        if (size==capacity) return true;
        return false;
    }
```

```java
public void insertFront(int data){
    if (isFull()){
        System.out.println("Deque is full");
        return;
    }
    else if (isEmpty()){
        Node temp=new Node(data);
        front=temp;
        rear=temp;
        size++;
    }
    else{
        Node temp=new Node(data);
        temp.next=front;
        front.prev=temp;
        front=temp;
        size++;
    }
}
```

```java
public void insertLast(int data){
    if (isFull()){
        System.out.println("Deque is full");
        return;
    }
    else if (isEmpty()){
        Node temp=new Node(data);
        front=temp;
        rear=temp;
        size++;
    }
    else{
        Node temp=new Node(data);
        rear.next=temp;
        temp.prev=rear;
        rear=temp;
        size++;
    }
}
```

```java
public int deleteFront(){
    if (isEmpty()){
        System.out.println("Deque is empty");
        return -1;
    }
    else if (front==rear){
        int temp=front.data;
        front=null;
        rear=null;
        size--;
        return temp;
    }
    else{
        int temp=front.data;
        front=front.next;
        front.prev=null;
        size--;
        return temp;
    }
}
```

```java
public int deleteLast(){
    if (isEmpty()){
        System.out.println("Deque is empty");
        return -1;
    }
    else if (front==rear){
        int temp=rear.data;
        front=null;
        rear=null;
        size--;
        return temp;
    }
    else{
        int temp=rear.data;
        rear=rear.prev;
        rear.next=null;
        size--;
        return temp;
    }
}
```

```
no usages
public void display(){
    if (isEmpty()){
        System.out.println("Deque is empty");
        return;
    }
    Node temp=front;
    while(temp!=null){
        System.out.print(temp.data+"\t");
        temp=temp.next;
    }
    System.out.println();
}
no usages
```

```
public int getFront(){
    if (isEmpty()){
        System.out.println("Deque is empty");
        return -1;
    }
    return front.data;
}
no usages
public int getRear(){
    if (isEmpty()){
        System.out.println("Deque is empty");
        return -1;
    }
    return rear.data;
}
```

Test case:

Create a queue of size 5

insetFront1(10):

insertLast(20):

insetFront(30):

deleteFront():

deleteLast():

insertLast(25):

insetFront(40):

insetFront(50):

getRear(): getFront():

```java
public class Q14 {
    public static void main(String[] args) {
        Deque q1 = new Deque();
        q1.insertFront( data: 10);
        q1.insertLast( data: 20);
        q1.insertFront( data: 30);
        q1.deleteFront();
        q1.deleteLast();
        q1.insertLast( data: 25);
        q1.insertFront( data: 40);
        q1.insertFront( data: 50);
        System.out.println("getRear(): "+ q1.getRear());
        System.out.println("getFront(): "+ q1.getFront());
    }
}
```

```
C:\Users\aadit\.jdks\openjdk-19.0.2\b
getRear(): 25
getFront(): 50

Process finished with exit code 0
```

15. You are given a stack data structure with push and pop operations. Implement a queue using instances of stack data str

```
public class QueueS {
    12 usages
    private Stack s1;
    7 usages
    private Stack s2;
    no usages
    QueueS(){
        s1=new Stack();
        s2=new Stack();
    }
    no usages
    public void enqueue(int data){
        s1.push(data);
    }
```

```
no usages
public int dequeue(){
    if (s1.isEmpty()){
        System.out.println("Queue is empty");
        return -1;
    }
    else{
        int d=s1.pop();
        while (!s1.isEmpty()){
            s2.push(d);
            d=s1.pop();
        }
        while (!s2.isEmpty()){
            s1.push(s2.pop());
        }
        return d;
    }
}
```

```java
public int peek(){
    if (s1.isEmpty()){
        System.out.println("Queue is empty");
        return -1;
    }
    else{
        int d=-1;
        while (!s1.isEmpty()){
            d=s1.pop();
            s2.push(d);
        }
        while (!s2.isEmpty()){
            s1.push(s2.pop());
        }
        return d;
    }
}
```