
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.

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.

```
public class QueueInt {  
    int[] arr ;  
    int rear;  
    int front = rear = -1 ;  
    int count = 0 ;  
  
    public QueueInt(){  
        arr = new int[10];  
        front = rear = -1 ;  
    }  
  
    public QueueInt(int size){  
        arr = new int[size];  
        front = rear = -1 ;  
    }  
  
    public void print() {  
        if ((front==rear&&front==-1)||front>=arr.length) {
```

```

        System.out.println("\nQueue is empty \n");
    }
    else{
        for (int i = front; i <= rear; i++) {
            System.out.print(arr[i]+"->");
        }
    }
}

public void enqueue(int data) {

    if (rear==arr.length-1)
    {
        System.out.println("\n Can't enqueue 😞 \n");
    }
    else if (count == 0) {
        front++;
        rear++;
        arr[rear] = data;
        count++;
    }
    else
    {
        rear++;
        arr[rear] = data;
    }
}
}

```

```

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

public void dequeue() {
    if ((front==rear&&front==-1)||front>=arr.length) {
        System.out.println("\ncan't pop 😞 \n");
    }
    else {
        int t = arr[front] ;
        if (front++==arr.length-1) {
            System.out.println("Poped element : " + t + "\n");
            front++ ;
        }
        else
        {
            System.out.println("Poped element : " + t + "\n");
            front++;
        }
    }
}
}

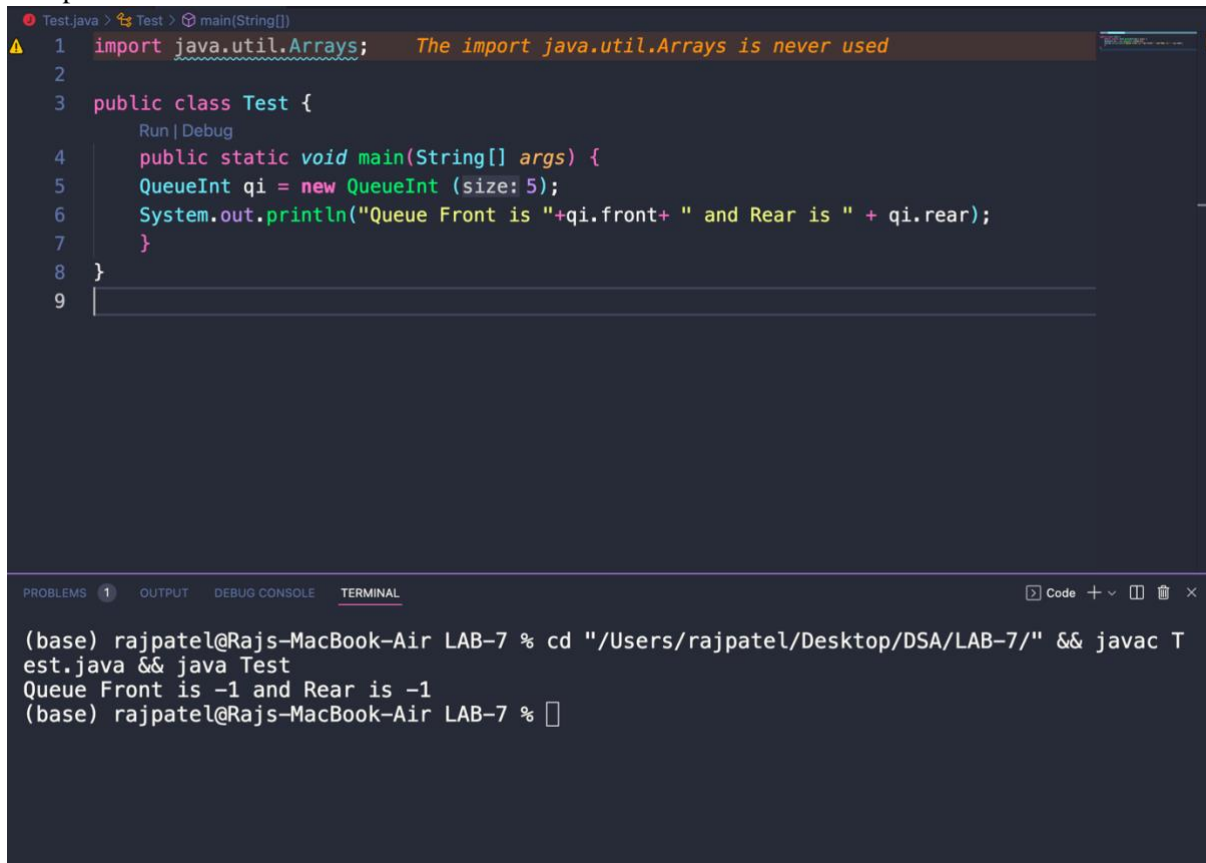
```

- (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.



```
Test.java > Test > main(String[])
1 import java.util.Arrays; The import java.util.Arrays is never used
2
3 public class Test {
4 Run | Debug
5 public static void main(String[] args) {
6 QueueInt qi = new QueueInt (size: 5);
7 System.out.println("Queue Front is "+qi.front+ " and Rear is " + qi.rear);
8 }
9 }
```

```
(base) rajpatel@Rajs-MacBook-Air LAB-7 % cd "/Users/rajpatel/Desktop/DSA/LAB-7/" && javac Test.java && java Test
Queue Front is -1 and Rear is -1
(base) rajpatel@Rajs-MacBook-Air LAB-7 %
```

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);
```



```
Test.java > Test > main(String[])
1 import java.util.Arrays; The import java.util.Arrays is never used
2
3 public class Test {
4 Run | Debug
5 public static void main(String[] args) {
6 QueueInt qi = new QueueInt ();
7 System.out.println(qi.arr.length);
8 }
9 }
```

```
(base) rajpatel@Rajs-MacBook-Air LAB-7 % cd "/Users/rajpatel/Desktop/DSA/LAB-7/" && javac Test.java && java Test
10
(base) rajpatel@Rajs-MacBook-Air LAB-7 %
```

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);

```

The screenshot shows an IDE with a Java file named 'Test.java'. The code in the file is as follows:

```

1 import java.util.Arrays;
2
3 public class Test {
4 public static void main(String[] args) {
5 QueueInt qi2 = new QueueInt (size: 15);
6 System.out.println(qi2.arr.length);
7 }
8 }

```

A warning message is visible: 'The import java.util.Arrays is never used'. Below the code editor, the terminal shows the command to compile and run the program:

```

(base) rajpatel@Rajs-MacBook-Air LAB-7 % cd "/Users/rajpatel/Desktop/DSA/LAB-7/" && javac Test.java && java Test
15
(base) rajpatel@Rajs-MacBook-Air LAB-7 %

```

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.

}

```

The screenshot shows an IDE with a Java file named 'Test.java'. The code in the file is as follows:

```

3 public class Test {
4 public static void main(String[] args) {
5 QueueInt qi = new QueueInt (size: 15);
6 qi.enqueue(data: 1);
7 qi.enqueue(data: 2);
8 qi.enqueue(data: 3);
9 qi.enqueue(data: 4);
10 qi.enqueue(data: 5);
11 qi.print();
12 }

```

Below the code editor, the terminal shows the command to compile and run the program, followed by the output:

```

(base) rajpatel@Rajs-MacBook-Air LAB-7 % cd "/Users/rajpatel/Desktop/DSA/LAB-7/" && javac Test.java && java Test
1->2->3->4->5->
(base) rajpatel@Rajs-MacBook-Air LAB-7 %

```

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

```

public void enqueue(int item) {

```

```
// Your logic here
```

```
}
```

Test cases:

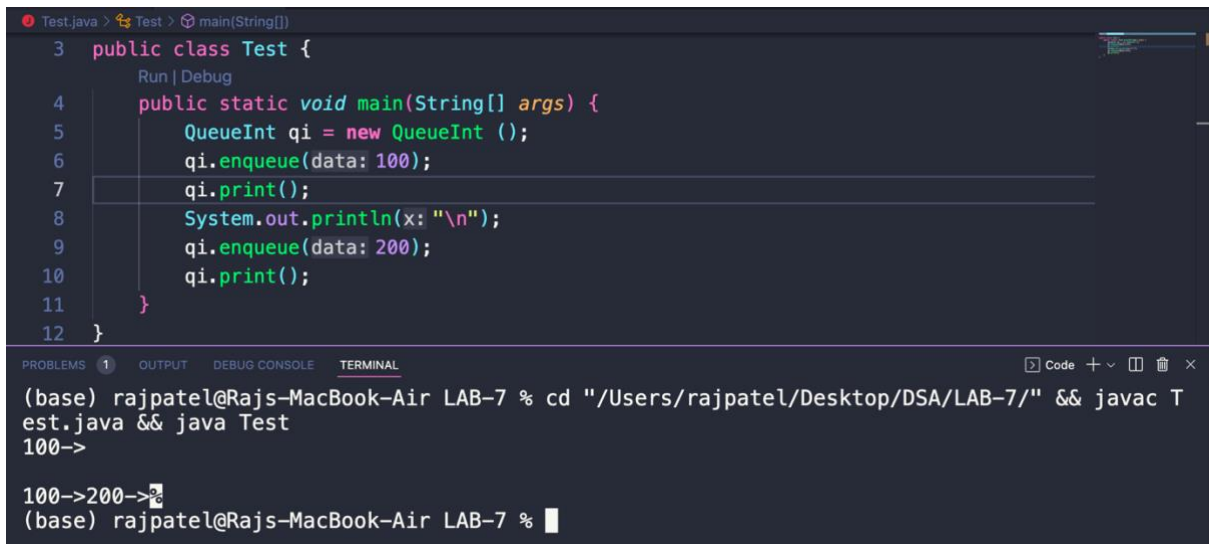
(a) Write test file and try invoking enqueue operations and check.

```
QueueInt qi = new QueueInt ();
```

```
qi.enqueue(100);
```

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

```
qi.print();
```



The screenshot shows an IDE with a Java file named 'Test.java'. The code defines a 'Test' class with a 'main' method. Inside 'main', a 'QueueInt' object 'qi' is created, and the following operations are performed: enqueue(100), print(), enqueue(200), and print(). The terminal output shows the command 'javac Test.java' and 'java Test' being executed, resulting in the output '100->' followed by '100->200->' on the next line.

```
Test.java > Test > main(String[])
3 public class Test {
 Run | Debug
4 public static void main(String[] args) {
5 QueueInt qi = new QueueInt ();
6 qi.enqueue(data: 100);
7 qi.print();
8 System.out.println(x: "\n");
9 qi.enqueue(data: 200);
10 qi.print();
11 }
12 }
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL

```
(base) rajpatel@Rajs-MacBook-Air LAB-7 % cd "/Users/rajpatel/Desktop/DSA/LAB-7/" && javac Test.java && java Test
100->
100->200->
(base) rajpatel@Rajs-MacBook-Air LAB-7 %
```

(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.

```
public void print() {
 if ((front==rear&&front==1)||front>=arr.length) {
 System.out.println("\nQueue is empty \n");
 }
 else{
 for (int i = front; i <=rear; i++) {
```

```

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

```

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 (rear==arr.length-1)
 {
 System.out.println("\n Can't enqueue \n");
 }
 else if (count == 0) {
 front++;
 rear++;
 arr[rear] = data;
 count++;
 }
 else
 {
 rear++;
 arr[rear] = data;
 }
}
}

```

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

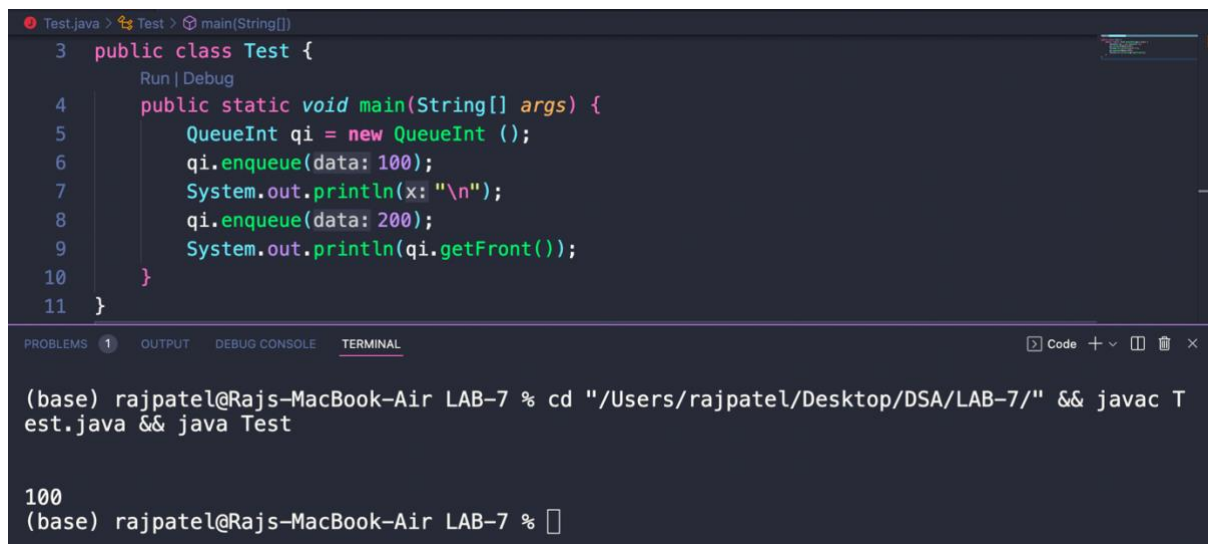
```
public int getFront() {
 // return arr[front];
}
```

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

Test cases:

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

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



The screenshot shows an IDE with a Java file named `Test.java`. The code defines a `Test` class with a `main` method. In the `main` method, a `QueueInt` object `qi` is created, and two elements (100 and 200) are enqueued. Then, `qi.getFront()` is called, and the result is printed. The terminal output shows the command `javac Test.java && java Test` being executed, followed by the output `100`.

```
Test.java > Test > main(String[])
3 public class Test {
 Run | Debug
4 public static void main(String[] args) {
5 QueueInt qi = new QueueInt ();
6 qi.enqueue(data: 100);
7 System.out.println(x: "\n");
8 qi.enqueue(data: 200);
9 System.out.println(qi.getFront());
10 }
11 }
```

(base) rajpatel@Rajs-MacBook-Air LAB-7 % cd "/Users/rajpatel/Desktop/DSA/LAB-7/" && javac Test.java && java Test

100  
(base) rajpatel@Rajs-MacBook-Air LAB-7 %

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
}
```

```
public void dequeue() {
 if ((front==rear&&front==-1)||front>=arr.length) {
 System.out.println("\ncan't pop \n");
 }
 else {
 int t = arr[front] ;
```



```

 if (front++==arr.length-1) {
 System.out.println("Poped element : " + t + "\n");
 front++;
 }
 else
 {
 System.out.println("Poped element : " + t + "\n");
 front++;
 }
 }
}
}

```

Test cases:

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

```
int item = qi.dequeue();
```

```
qi.print();
```

(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.

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 void dequeue() {
 if ((front==rear&&front==-1)||front>=arr.length) {
 System.out.println("\ncan't pop \n");
 }
 else {
 int t = arr[front] ;
 if (front++==arr.length-1) {
 System.out.println("Poped element : " + t + "\n");

```

```

 front++;
 }
 else
 {
 System.out.println("Poped element : " + t + "\n");
 front++;
 }
}
}
}

```

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?

```
Test.java > Test > main(String[])
1 import java.util.Arrays;
2
3 public class Test {
4 Run | Debug
 public static void main(String[] args) {
5
6 QueueInt qi1 = new QueueInt (size: 5);
7 QueueInt qi2 = new QueueInt (size: 5);
8 qi1.enqueue(data: 100);
9 qi2.enqueue(data: 100);
10 qi1.enqueue(data: 200);
11 qi2.enqueue(data: 300);
12
13 if (Arrays.equals(qi1.arr, qi2.arr))
14 System.out.println(x: "\nBoth qi1 and qi2 are same\n");
15 else
16 System.out.println(x: "\nBoth qi1 and qi2 are not the same\n");
17 }
18 }
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** Code + - - - X

```
cd "/Users/rajpatel/Desktop/DSA/LAB-7/" && javac Test.java && java Test
(base) rajpatel@Rajs-MacBook-Air LAB-7 % cd "/Users/rajpatel/Desktop/DSA/LAB-7/" && javac Test.java && java Test

Both qi1 and qi2 are not the same

(base) rajpatel@Rajs-MacBook-Air LAB-7 %
```

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 in 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

}
```

Test cases:

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

```
Test.java > Test > main(String[])
1 import java.util.Arrays;
2
3 public class Test {
 Run | Debug
4 public static void main(String[] args) {
5
6 QueueInt qi1 = new QueueInt (size: 5);
7 QueueInt qi2 = new QueueInt (size: 5);
8 qi1.enqueue(data: 100);
9 qi2.enqueue(data: 100);
10 qi1.enqueue(data: 200);
11 qi2.enqueue(data: 300);
12
13 if (Arrays.equals(qi1.arr, qi2.arr))
14 System.out.println(x: "\nBoth qi1 and qi2 are same\n");
15 else
16 System.out.println(x: "\nBoth qi1 and qi2 are not the same\n");
17 }
18 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
cd "/Users/rajpatel/Desktop/DSA/LAB-7/" && javac Test.java && java Test
(base) rajpatel@Rajs-MacBook-Air LAB-7 % cd "/Users/rajpatel/Desktop/DSA/LAB-7/" && javac T
est.java && java Test

Both qi1 and qi2 are not the same

(base) rajpatel@Rajs-MacBook-Air LAB-7 %
```

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

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.

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

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

Test case:

Create a queue of size 5

insetFront1(10):

insertLast(20):

insetFront(30):

deleteFront():

deleteLast():

insertLast(25):

insertFront(40):

insertFront(50):

getRear(): getFront():

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