FAKULTI TEKNOLOGI
KEJURUTERAAN KELAUTAN
DAN INFORMATIK

2022/2023

DATA STRUCTURE & ALGORITHM

Lab 4: Queue

VERSION 1

STUDENT INFORMATION

PLEASE FILL IN YOUR DETAILS:

NAME: OMAR ISMAIL ABDJALEEL ALOMORY

MATRIC NUMBER:S63955

GROUP:K2

LAB:MP3

DATE:26/11/2022

TABLE OF CONTENTS

INSTRUCTIONS

Manual makmal ini adalah untuk kegunaan pelajar-pelajar Fakulti Teknologi Kejuruteraan Kelautan dan Informatik, Universiti Malaysia Terengganu (UMT) sahaja. Tidak dibenarkan mencetak dan mengedar manual ini tanpa kebenaran rasmi daripada penulis.

Sila ikuti langkah demi langkah sebagaimana yang dinyatakan di dalam manual.

This laboratory manual is for use by the students of the Faculty of Ocean Engineering Technology and Informatics, Universiti Malaysia Terengganu (UMT) only. It is not permissible to print and distribute this manual without the official authorisation of the author.

Please follow step by step, as described in the manual.

TASK 1: APPLY AND TEST THE IMPLEMENTATION OF QUEUE USING ARRAY

OBJECTIVE

In this task, students must be able to:

- Apply the simple implementation of the queue.
- Test the implementation

ESTIMATED TIME

[60 Minutes]

DEFINITION OF QUEUE

Queues are an essential data structure that is found in vast amounts of software from user mode to kernel mode applications that are core to the system. Fundamentally they honour a first in first out (FIFO) strategy, that is the item first put into the queue will be the first served, the second item added to the queue will be the second to be served and so on.

We usually say that elements enter a queue at the back and are removed from the front. A metaphor for this terminology is a line of people waiting to get on an amusement park ride. People waiting for such a ride enter at the back of the line and get on the ride from the front of the line. There are many other applications of queues. Stores, theatres, reservation centres, and other similar services typically process customer requests according to the FIFO principle. A queue would, therefore, be a logical choice for a data structure to handle calls to a customer service centre or a wait-list at a restaurant. FIFO queues are also used by many computing devices, such as a networked printer, or a Web server responding to requests.

BASIC OPERATIONS IN QUEUE

A linear queue only allows you to access the item at the front of the queue; when you add an item to the queue that item is placed at the back of the queue.

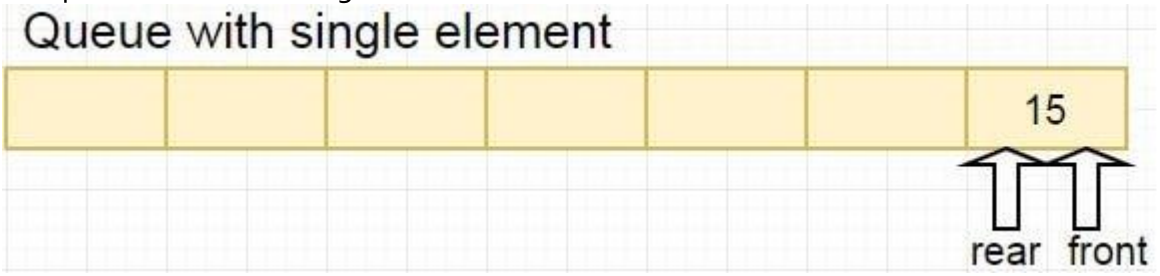
Queues usually have the following three core methods:

Enqueue: places an element at the back of the queue.

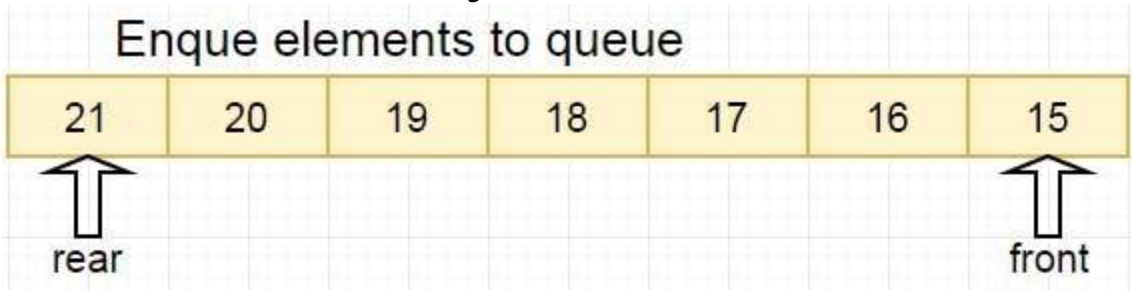
Dequeue: retrieves the element at the front of the queue and removes it from the queue.

Peek: retrieves the element at the front of the queue without removing it from the queue.

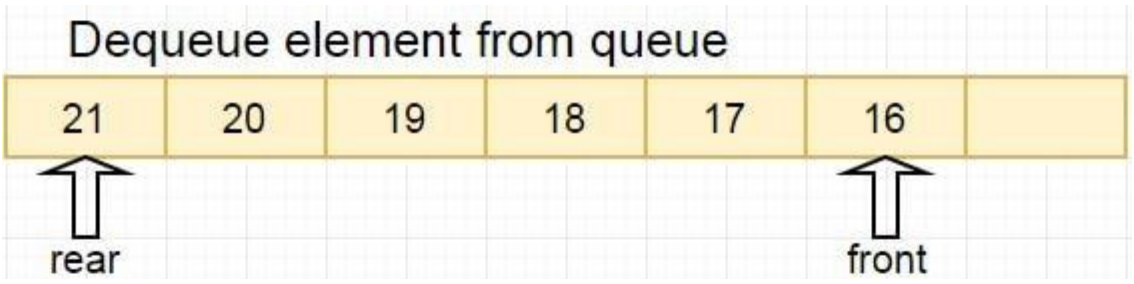
The Queue in the example below has a single element with value 15. So, both the front and rear point to the same single element:



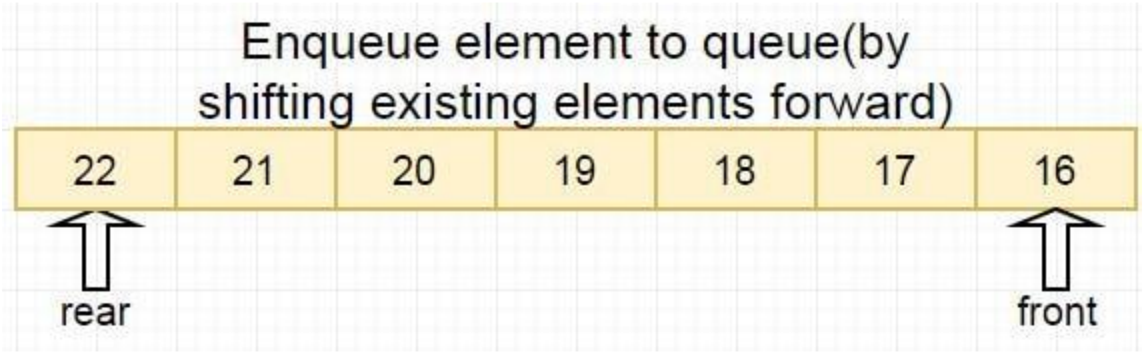
We now enqueue elements to make the queue full. So, the element with value 21 is our rear element while element with value 15 is our first element as follows:



Next Dequeue an element with value 15 from the queue. So our first element will now be an element having value 16 as follows:



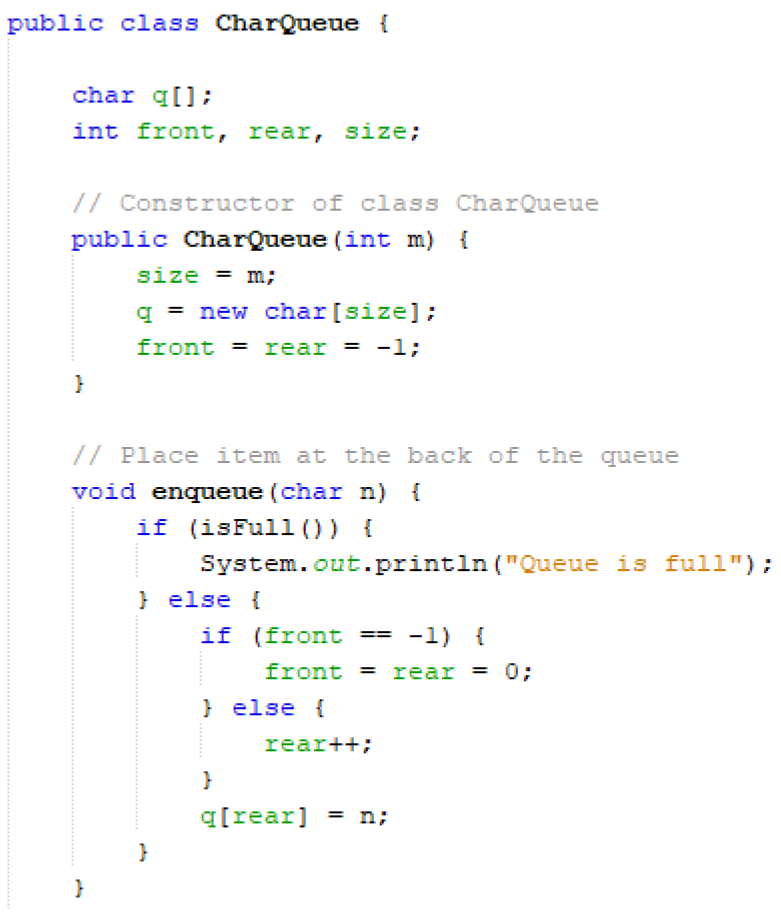
If elements are now to be added to the queue, space must be made by shifting all the queue elements forward. So, all the elements from 16 to 21 are moved forward to create space for our new rear element 22.


Enqueue element to queue(by shifting existing elements forward)
22
21
20
19
18
17
16
rear
front

Based on the above explanation, we can see that the operations of a linear queue involve shifting of elements during the operations. This situation can be avoided by making use of a Circular Queue (to be implemented in Task 3).

STEPS:

1. Open `Netbeans` and create new java application project.
2. Name your project as `QueueExperiment` and click finish.
3. Change author profiles to :
   a. `Name :`
   b. `Program: <put your program. E.g., SMSK(SE) or SMSK with IM`
   c. `Course : CSF3104`
   d. `Lab : <enter lab number>`
   e. `Date : <enter lab date>`
4. In the same `QueueExperiment` project's package, create a new file named `CharQueue.java` and insert the following codes:

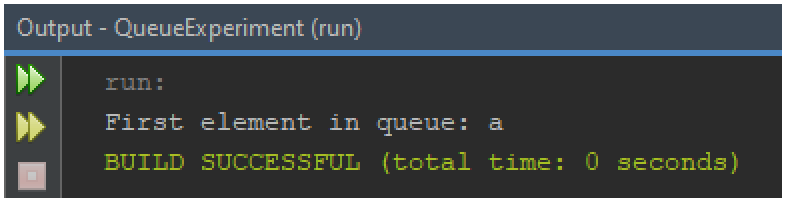


```java
public class CharQueue {

    char q[];
    int front, rear, size;

    // Constructor of class CharQueue
    public CharQueue(int m) {
        size = m;
        q = new char[size];
        front = rear = -1;
    }

    // Place item at the back of the queue
    void enqueue(char n) {
        if (isFull()) {
            System.out.println("Queue is full");
        } else {
            if (front == -1) {
                front = rear = 0;
            } else {
                rear++;
            }
            q[rear] = n;
        }
    }
    /* Retrieves the element at the front of the queue
    and removes it from the queue.
     */
    char dequeue() {
        if (isEmpty()) {
            System.out.println("queue is empty");
            return (char) 250;
        } else {
            char x = q[front];
            front++;
            return x;
        }
    }

    /* retrieves the element at the front of the queue without removing
    it from the queue. */
    char peek() {
        return q[front];
    }

    boolean isEmpty() {
        return (front == -1 || front > rear);
    }

    boolean isFull() {
        return (rear == size - 1);
    }

} //end of CharQueue class
```

5. Next, test the class using the following codes. Put the codes in `QueueExperiment.java`.

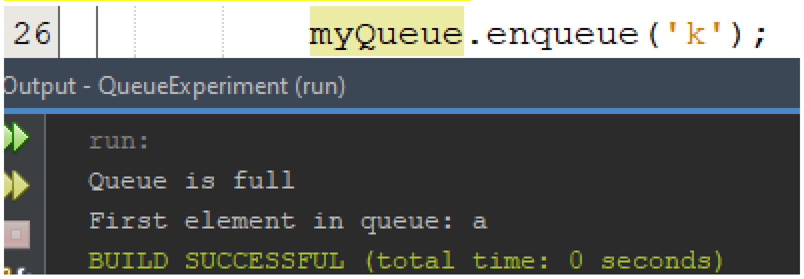
```
CharQueue myQueue = new CharQueue(10);
myQueue.enqueue('a');
myQueue.enqueue('b');
myQueue.enqueue('c');
myQueue.enqueue('d');
myQueue.enqueue('e');
myQueue.enqueue('f');
myQueue.enqueue('g');
myQueue.enqueue('h');
myQueue.enqueue('i');
myQueue.enqueue('j');

System.out.println("First element in queue: "+myQueue.peek());
```

6. Save, compile and run your source codes.
7. Observe the output and upload the screenshot using the control box below:
   peek() method,It shows the element at the front of the queueu which is 'a' without removint it from the queue



```
Output - QueueExperiment (run)
run:
First element in queue: a
BUILD SUCCESSFUL (total time: 0 seconds)
```

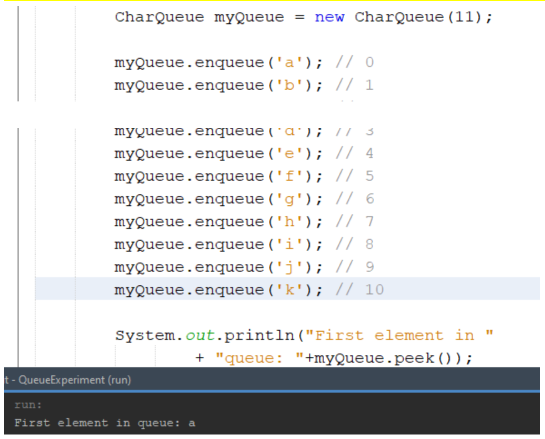
8. Now, insert another element `'k'` in your existing queue. Compile and run your codes again. Do you find any error? Not error but since we initialize the our queue to 10 values only and we are trying to add another values, it simply not going to accept it and print out message that shows its already full.Record your output in the following box:



```
            myQueue.enqueue('k');
```

```
Output - QueueExperiment (run)
run:
Queue is full
First element in queue: a
BUILD SUCCESSFUL (total time: 0 seconds)
```

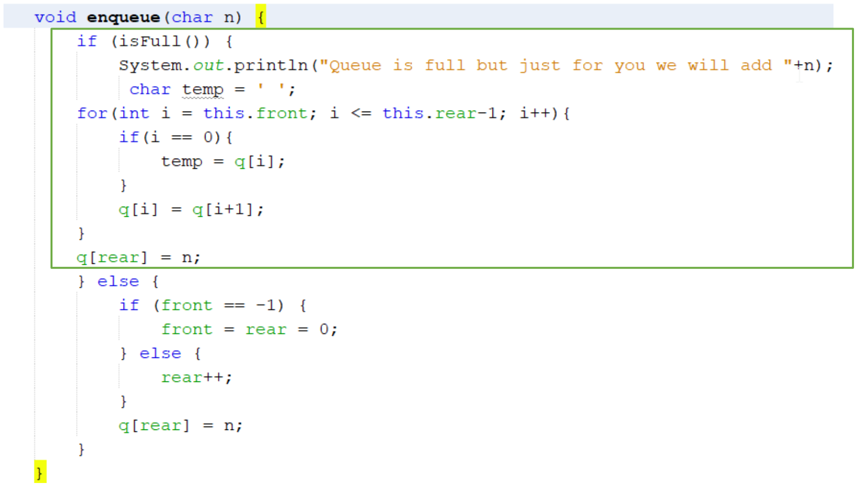
9. Fix the above code and explain your step.
   Answer:
   We can do it in many ways, I will try to show two the first one is just by simply increasing the size of the queue
   CharQueue myQueue = new CharQueue(11);



```
        CharQueue myQueue = new CharQueue(11);

        myQueue.enqueue('a'); // 0
        myQueue.enqueue('b'); // 1

        myQueue.enqueue('d'); // 3
        myQueue.enqueue('e'); // 4
        myQueue.enqueue('f'); // 5
        myQueue.enqueue('g'); // 6
        myQueue.enqueue('h'); // 7
        myQueue.enqueue('i'); // 8
        myQueue.enqueue('j'); // 9
        myQueue.enqueue('k'); // 10

        System.out.println("First element in "
                + "queue: "+myQueue.peek());
```

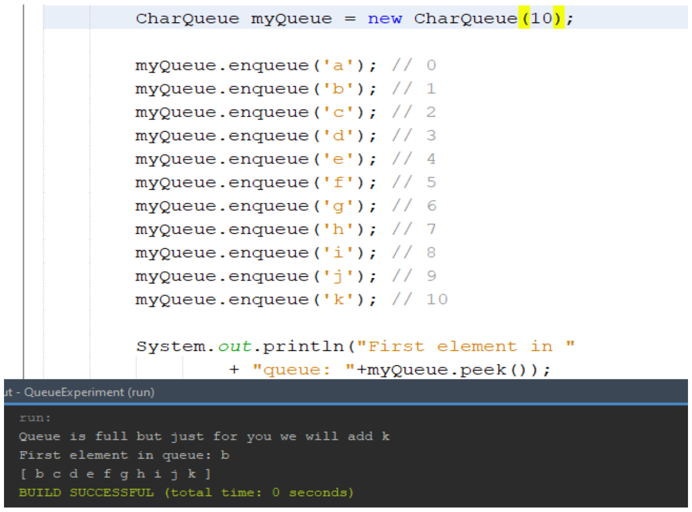
```
t - QueueExperiment (run)
run:
First element in queue: a
```

The other way is to modify the method enqueu to accept values if the queue is already full by shifting the element to the left and remove the first element(maybe not practical):



```java
void enqueue(char n) {
    if (isFull()) {
        System.out.println("Queue is full but just for you we will add "+n);
         char temp = ' ';
    for(int i = this.front; i <= this.rear-1; i++){
        if(i == 0){
            temp = q[i];
        }
        q[i] = q[i+1];
    }
    q[rear] = n;
    } else {
        if (front == -1) {
            front = rear = 0;
        } else {
            rear++;
        }
        q[rear] = n;
    }
}
```

This is the result

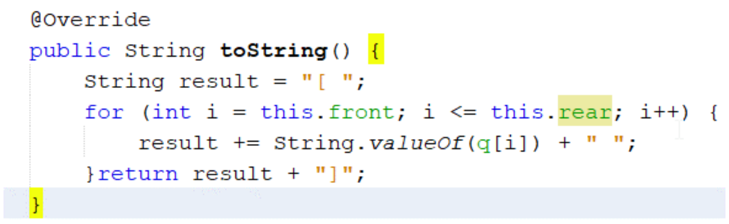


```java
        CharQueue myQueue = new CharQueue(10);

        myQueue.enqueue('a'); // 0
        myQueue.enqueue('b'); // 1
        myQueue.enqueue('c'); // 2
        myQueue.enqueue('d'); // 3
        myQueue.enqueue('e'); // 4
        myQueue.enqueue('f'); // 5
        myQueue.enqueue('g'); // 6
        myQueue.enqueue('h'); // 7
        myQueue.enqueue('i'); // 8
        myQueue.enqueue('j'); // 9
        myQueue.enqueue('k'); // 10

        System.out.println("First element in "
                + "queue: "+myQueue.peek());
```

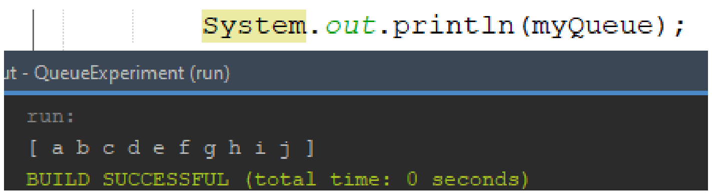
```
ut - QueueExperiment (run)
run:
Queue is full but just for you we will add k
First element in queue: b
[ b c d e f g h i j k ]
BUILD SUCCESSFUL (total time: 0 seconds)
```

We can change the return type too to retrun the temp as it the value that have been lost. I am not sure what the question actually requier since there was no error. Thanks

10. Next, modify write codes in `CharQueue.java` to display all elements in the queue.



```java
@Override
public String toString() {
    String result = "[ ";
    for (int i = this.front; i <= this.rear; i++) {
        result += String.valueOf(q[i]) + " ";
    }return result + "]";
}
```



```java
            System.out.println(myQueue);
```

```
ut - QueueExperiment (run)
run:
[ a b c d e f g h i j ]
BUILD SUCCESSFUL (total time: 0 seconds)
```

Answer: [Copy and paste ALL your java codes (from step 1 until 10) from Netbeans into the following text box]

This is CharQueue Class

```
/**
 * @author Omar Alomory
 * Program: SMSK(KMA) K2
 * Course: CSF3013
 * Lab: MP3
 * Date: 27/11/2022
 */
public class CharQueue {


    char q[];
    int front, rear, size;


    public CharQueue(int m) {
        size = m;
        q = new char[size];
        front = rear = -1;
    }
    // Place an item at the back of the queue.
    void enqueue(char n) {
        if (isFull()) {
            System.out.println("Queue is full");
//              System.out.println("Queue is full but just for you we
will add "+n);
//               char temp = ' ';
//        for(int i = this.front; i <= this.rear-1; i++){
//            if(i == 0){
//                temp = q[i];
//            }
//            q[i] = q[i+1];
//        }
//        q[rear] = n;
        } else {
            if (front == -1) {
                front = rear = 0;
            } else {
                rear++;
            }
            q[rear] = n;
        }
    }
    /* retrieves the element at the front of the
         queue and removes it from the queue */
    char dequeue() {
        if (isEmpty()) {
            System.out.println("Queue is empty");
            return (char) 250;
        } else {
            char x = q[front];
            front++;
            return x;
        }
    }
    /* retrieves the element at the front of the
     queue without removing it from the queue */
    char peek() {
        return q[front];
    }
    // checks if the queue is empty.
    boolean isEmpty() {
        return (front == -1 || front > rear);
    }
    // checks if the queue is full.
    boolean isFull() {
        return (rear == size - 1);
    }
    // returning all the values in the queue as String for later uses
    @Override
    public String toString() {
        String result = "[ ";
        for (int i = this.front; i <= this.rear; i++) {
            result += String.valueOf(q[i]) + " ";
        }return result + "]";
    }
}
```

This is the test class for CharQueue which is QueueExperiment

```
/**
 * @author Omar Alomory
 * Program: SMSK(KMA) K2
 * Course: CSF3013
 * Lab: MP3
 * Date: 27/11/2022
 */
public class QueueExperiment {
    public static void main(String[] args) {
        CharQueue myQueue = new CharQueue(10);

        myQueue.enqueue('a'); // 0
        myQueue.enqueue('b'); // 1
        myQueue.enqueue('c'); // 2
        myQueue.enqueue('d'); // 3
        myQueue.enqueue('e'); // 4
        myQueue.enqueue('f'); // 5
        myQueue.enqueue('g'); // 6
        myQueue.enqueue('h'); // 7
        myQueue.enqueue('i'); // 8
        myQueue.enqueue('j'); // 9
        myQueue.enqueue('k'); // 10

        System.out.println("First element in "
                + "queue: "+myQueue.peek());
        System.out.println(myQueue);

    }
}
```

TASK 2: OBSERVING THE LIMITATION OF A LINEAR QUEUE

OBJECTIVE

In this task, students will test the dequeue method of the previously created queue and observe the limitation of the linear queue.

ESTIMATED TIME

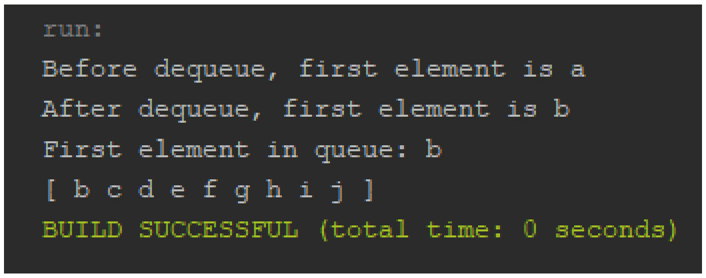
[10 Minutes]

STEPS:

1. Open queue project that you have created in Task 1.
2. Without re-sizing the size of the array for the queue put the following codes in the `main` method:

```
myQueue.enqueue('a');
myQueue.enqueue('b');
myQueue.enqueue('c');
myQueue.enqueue('d');
myQueue.dequeue();
myQueue.enqueue('e');
myQueue.enqueue('f');
myQueue.enqueue('g');
myQueue.enqueue('h');
myQueue.enqueue('i');
myQueue.enqueue('j');
```

3. Put codes to display the elements in the queue as well. Did you see any error?No. Put your output in the following control box:

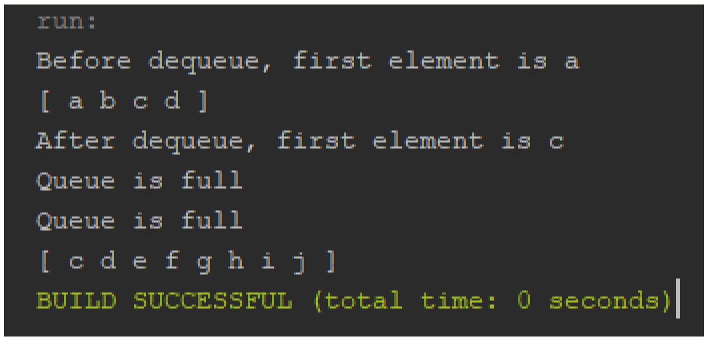


```
run:
Before dequeue, first element is a
After dequeue, first element is b
First element in queue: b
[ b c d e f g h i j ]
BUILD SUCCESSFUL (total time: 0 seconds)
```

4. Next, replace the code in Step 2 with the following:

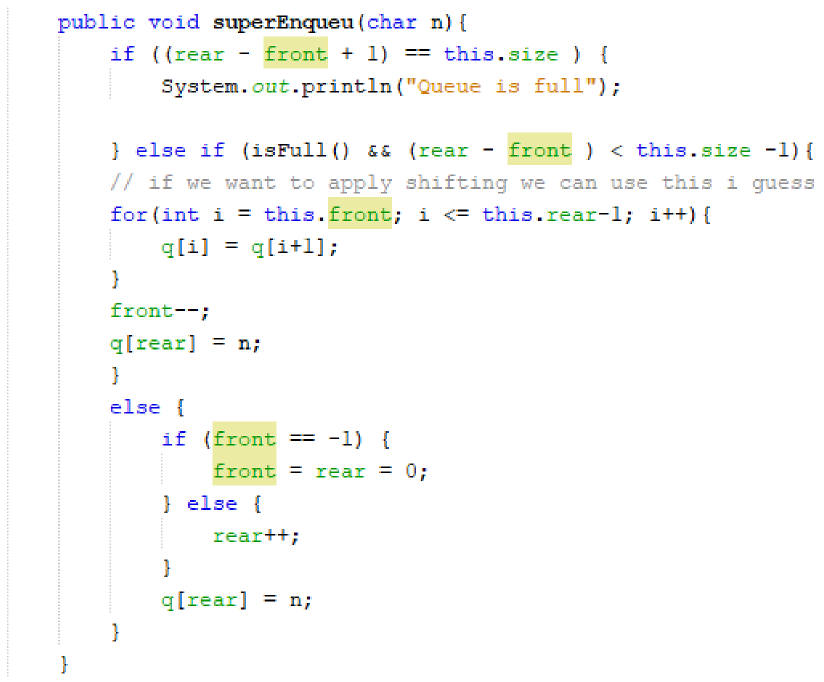
```
myQueue.enqueue('a');
myQueue.enqueue('b');
myQueue.enqueue('c');
myQueue.enqueue('d');
myQueue.dequeue();
myQueue.dequeue();
myQueue.enqueue('e');
myQueue.enqueue('f');
myQueue.enqueue('g');
myQueue.enqueue('h');
myQueue.enqueue('i');
myQueue.enqueue('j');
myQueue.enqueue('k');
myQueue.enqueue('l');
```

5. Compile and run the codes. Observe the result.althogh we dequeue two element but its unable to enqueue due to limitation of linear queue. Maybe. Upload the result using the control box below:



```
run:
Before dequeue, first element is a
[ a b c d ]
After dequeue, first element is c
Queue is full
Queue is full
[ c d e f g h i j ]
BUILD SUCCESSFUL (total time: 0 seconds)
```

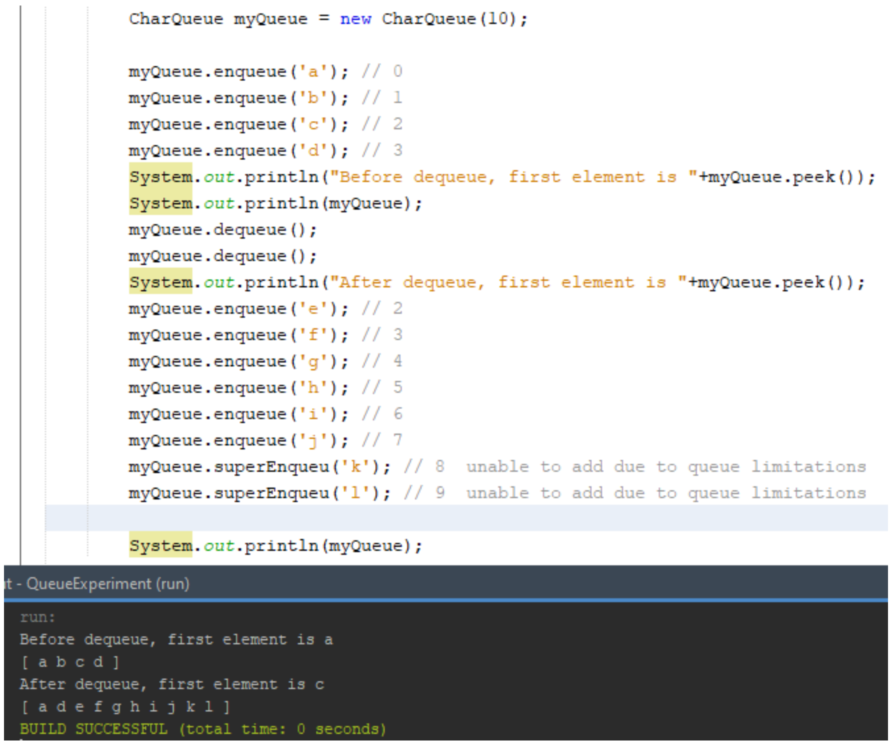
6. Why do you think the situation happens? Give your opinion by investigating the codes in `CharQueue.java.`
   Hint: Read the notes regarding the Queue.
   Answer:
   Although we have some spaces due to dequeuing but still have this problem. In order to solve it we need to shif the element to the right so that we empty spaces for the new values to be inserted at the rear of the queue.
   This is another method of enqueue for such problem (superEnqueue(char n)):



```
public void superEnqueu(char n){
    if ((rear - front + 1) == this.size ) {
        System.out.println("Queue is full");

    } else if (isFull() && (rear - front ) < this.size -1){
    // if we want to apply shifting we can use this i guess
    for(int i = this.front; i <= this.rear-1; i++){
        q[i] = q[i+1];
    }
    front--;
    q[rear] = n;
    }
    else {
        if (front == -1) {
            front = rear = 0;
        } else {
            rear++;
        }
        q[rear] = n;
    }
}
```

I change the term full size at the beginning to check if queue is actually full or we still have some spaces. If that the case a message would be printed otherwise we will apply shifting

And this is the result:



```
CharQueue myQueue = new CharQueue(10);

myQueue.enqueue('a'); // 0
myQueue.enqueue('b'); // 1
myQueue.enqueue('c'); // 2
myQueue.enqueue('d'); // 3
System.out.println("Before dequeue, first element is "+myQueue.peek());
System.out.println(myQueue);
myQueue.dequeue();
myQueue.dequeue();
System.out.println("After dequeue, first element is "+myQueue.peek());
myQueue.enqueue('e'); // 2
myQueue.enqueue('f'); // 3
myQueue.enqueue('g'); // 4
myQueue.enqueue('h'); // 5
myQueue.enqueue('i'); // 6
myQueue.enqueue('j'); // 7
myQueue.superEnqueu('k'); // 8  unable to add due to queue limitations
myQueue.superEnqueu('l'); // 9  unable to add due to queue limitations

System.out.println(myQueue);
```

t - QueueExperiment (run)

```
run:
Before dequeue, first element is a
[ a b c d ]
After dequeue, first element is c
[ a d e f g h i j k l ]
BUILD SUCCESSFUL (total time: 0 seconds)
```

[Copy and paste ALL your java codes (from step 2 until 4) from Netbeans into the following text box]

CharQueue class

```
/**
 * @author Omar Alomory
 * Program: SMSK(KMA) K2
 * Course: CSF3013
 * Lab: MP3
 * Date: 27/11/2022
 */
public class CharQueue {

    char q[];
    int front, rear, size;

    public CharQueue(int m) {
        size = m;
        q = new char[size];
        front = rear = -1;
    }
    // Place an item at the back of the queue.
    void enqueue(char n) {
        if (isFull()) {
            System.out.println("Queue is full");
        } else {
            if (front == -1) {
                front = rear = 0;
            } else {
                rear++;
            }
            q[rear] = n;
        }
    }
    /* retrieves the element at the front of the
         queue and removes it from the queue */
    char dequeue() {
        if (isEmpty()) {
            System.out.println("Queue is empty");
            return (char) 250;
        } else {
            char x = q[front];
            front++;
            return x;
        }
    }

    /* retrieves the element at the front of the
     queue without removing it from the queue */
    char peek() {
        return q[front];
    }
    // checks if the queue is empty.
    boolean isEmpty() {
        return (front == -1 || front > rear);
```

```
}
// checks if the queue is full.
boolean isFull() {
    return (rear == size - 1);
}

// returning all the values in the queue as String for later uses
@Override
public String toString() {
    String result = "[ ";
    for (int i = this.front; i <= this.rear; i++) {
        result += String.valueOf(q[i]) + " ";
    }return result + "]";
}

/* if we want to apply enqueue with shifting when rear is equal to
   size but there still some empty spaces due to dequeuing
*/
public void superEnqueu(char n){
    if ((rear - front + 1) == this.size ) {
        System.out.println("Queue is full");

    } else if (isFull() && (rear - front ) < this.size -1){
    // if we want to apply shifting we can use this i guess
    for(int i = this.front; i <= this.rear-1; i++){
        q[i] = q[i+1];
    }
```

```
        front--;
        q[rear] = n;
        }
        else {
            if (front == -1) {
                front = rear = 0;
            } else {
                rear++;
            }
            q[rear] = n;
        }
    }
}
```

Task 2&3

```
/**
 * @author Omar Alomory
 * Program: SMSK(KMA) K2
 * Course: CSF3013
 * Lab: MP3
 * Date: 27/11/2022
 */
public class QueueExperiment {
    public static void main(String[] args) {
        CharQueue myQueue = new CharQueue(10);

        myQueue.enqueue('a'); // 0
        myQueue.enqueue('b'); // 1

        myQueue.enqueue('c'); // 2

        myQueue.enqueue('d'); // 3

        System.out.println("Before dequeue, first element is "+myQueue.peek());

        myQueue.dequeue();

        System.out.println("After dequeue, first element is "+myQueue.peek());

        myQueue.enqueue('e'); // 3

        myQueue.enqueue('f'); // 4

        myQueue.enqueue('g'); // 5

        myQueue.enqueue('h'); // 6

        myQueue.enqueue('i'); // 7

        myQueue.enqueue('j'); // 8


        System.out.println(myQueue);

    }

}
```

Task 4

```
/**

 * @author Omar Alomory

 * Program: SMSK(KMA) K2

 * Course: CSF3013

 * Lab: MP3

 * Date: 27/11/2022

 */

public class QueueExperiment {
    public static void main(String[] args) {
        CharQueue myQueue = new CharQueue(10);


        myQueue.enqueue('a'); // 0
        myQueue.enqueue('b'); // 1
        myQueue.enqueue('c'); // 2
        myQueue.enqueue('d'); // 3
        System.out.println("Before dequeue, first element is 
"+myQueue.peek());
        System.out.println(myQueue);
        myQueue.dequeue();
        myQueue.dequeue();
        System.out.println("After dequeue, first element is 
"+myQueue.peek());
        myQueue.enqueue('e'); // 2
        myQueue.enqueue('f'); // 3
        myQueue.enqueue('g'); // 4
        myQueue.enqueue('h'); // 5
        myQueue.enqueue('i'); // 6
        myQueue.enqueue('j'); // 7
        myQueue.enqueue('k'); // 8  unable to add due to queue 
limitations
        myQueue.enqueue('l'); // 9  unable to add due to queue 
limitations


        System.out.println(myQueue);
    }
}
```

TASK 3: IMPLEMENTATION OF CIRCULAR QUEUE

OBJECTIVE

In this task, students will implement the circular queue to overcome the limitation of the linear queue that has been coded previously.

ESTIMATED TIME

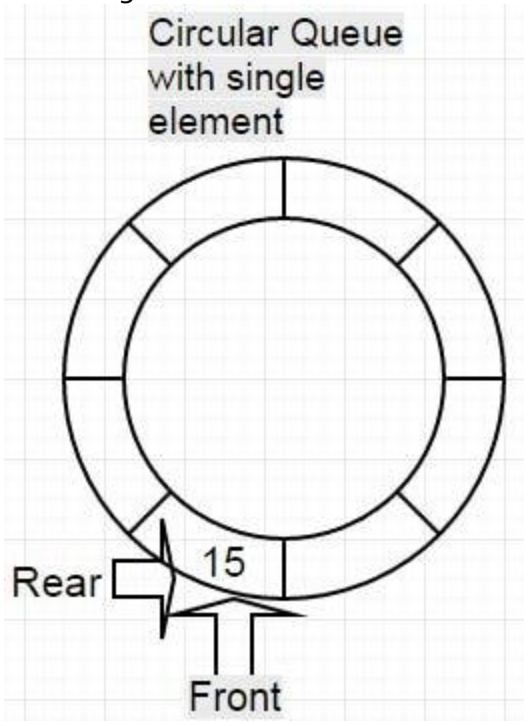
[45 minutes]

DEFINITION OF CIRCULAR QUEUE

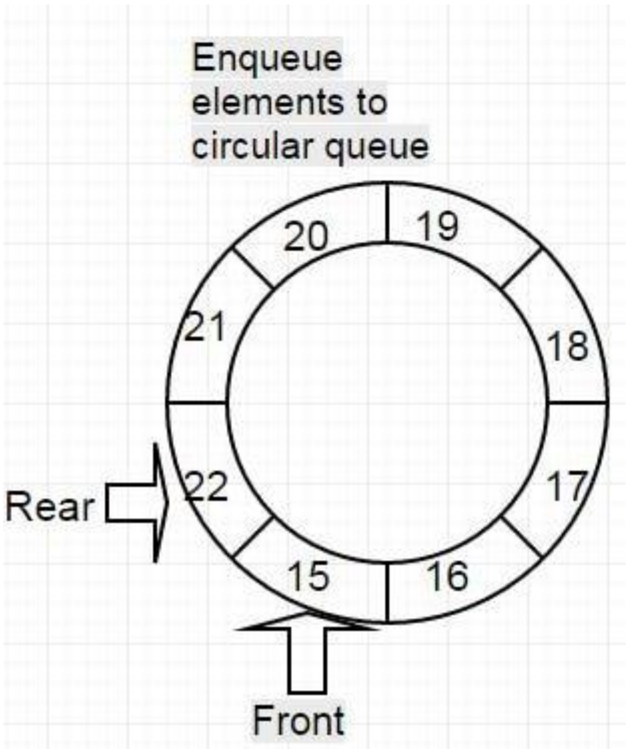
A circular queue is an excellent abstraction for applications in which elements are cyclically arranged, such as for multiplayer, turn-based games, or round-robin scheduling of computing processes. In this task, we will implement a demonstration of the use of a circular queue.

BASIC OPERATIONS IN CIRCULAR QUEUE

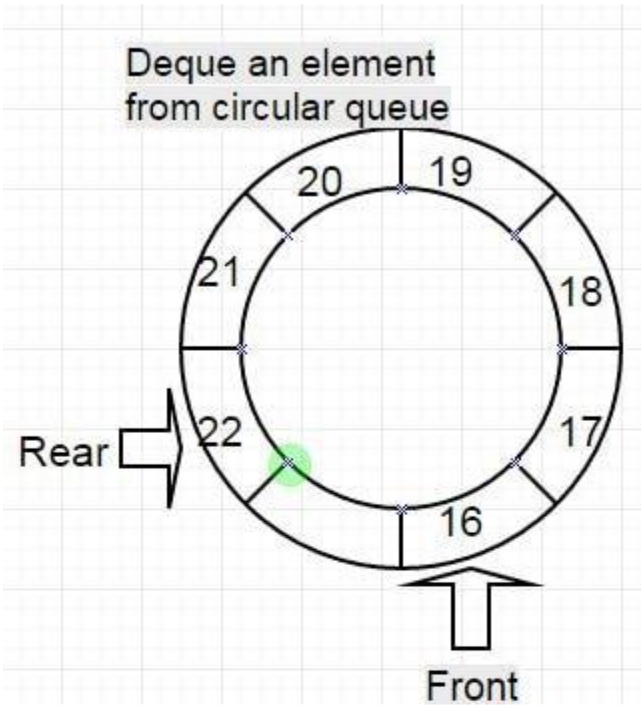
The circular queue has a single element with value 15. So, both the front and rear point to the same single element:



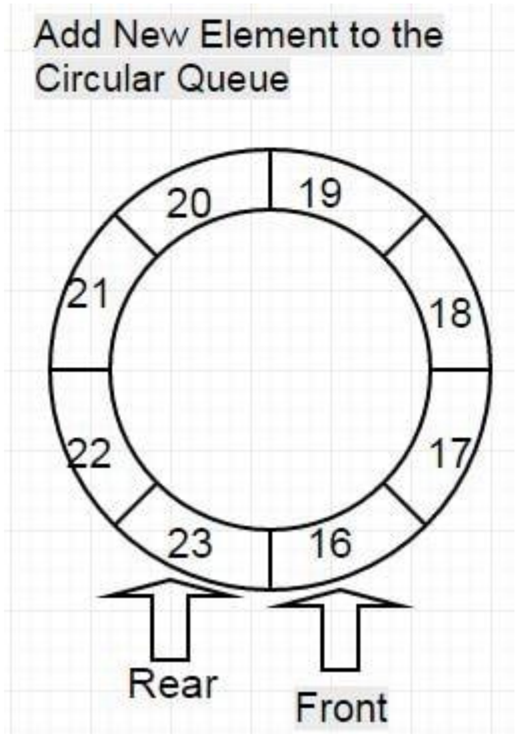
We now enqueue elements to make the circular queue full. So, the element with value 22 is our rear element while element with value 15 is our first element as follows:



Next Dequeue an element with value 15 from the circular queue. So our first element will now be an element having value 16 as follows:

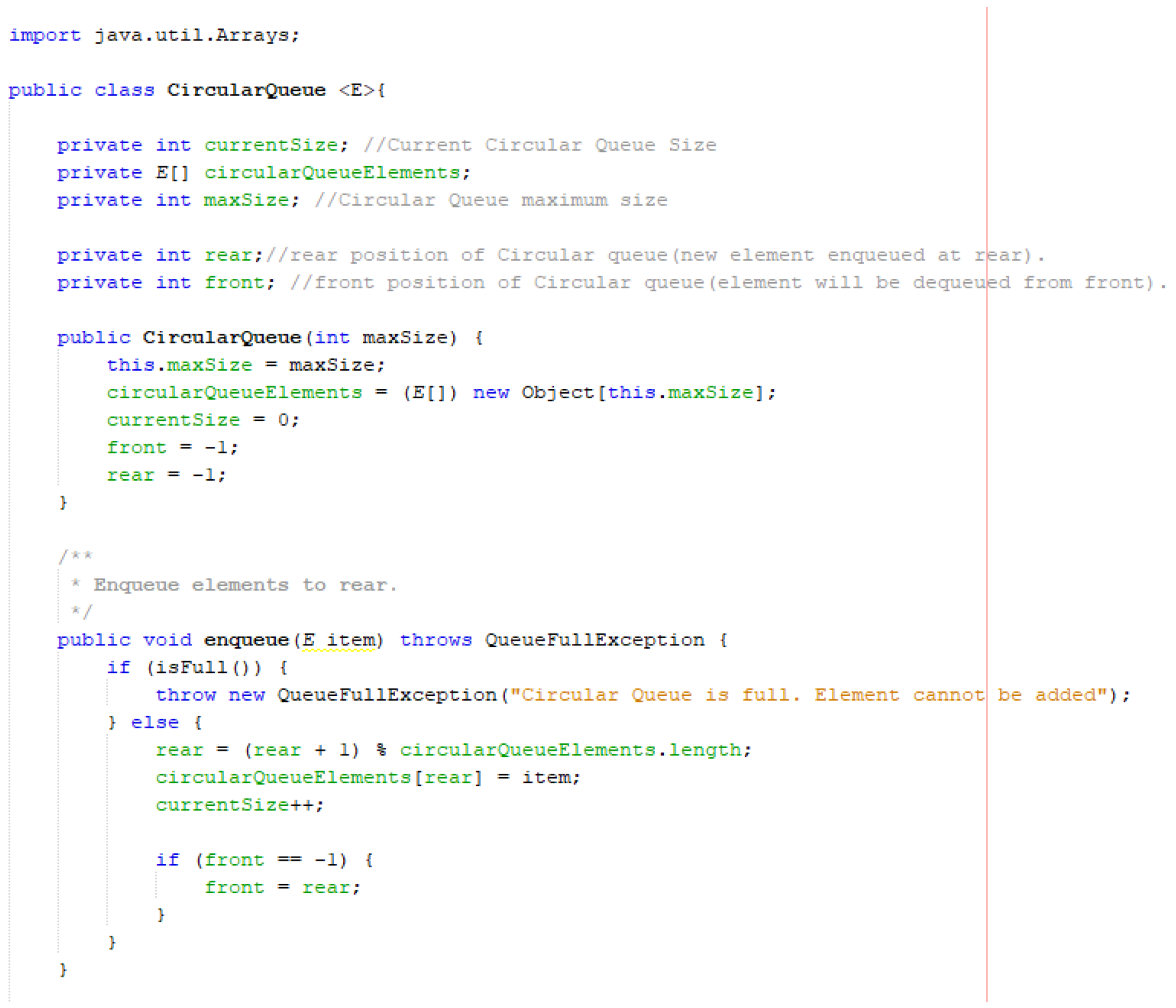


If a new element is to be added to the circular queue there is no need to shift the existing elements only the rear pointer will need to be moved to the appropriate location as follows:

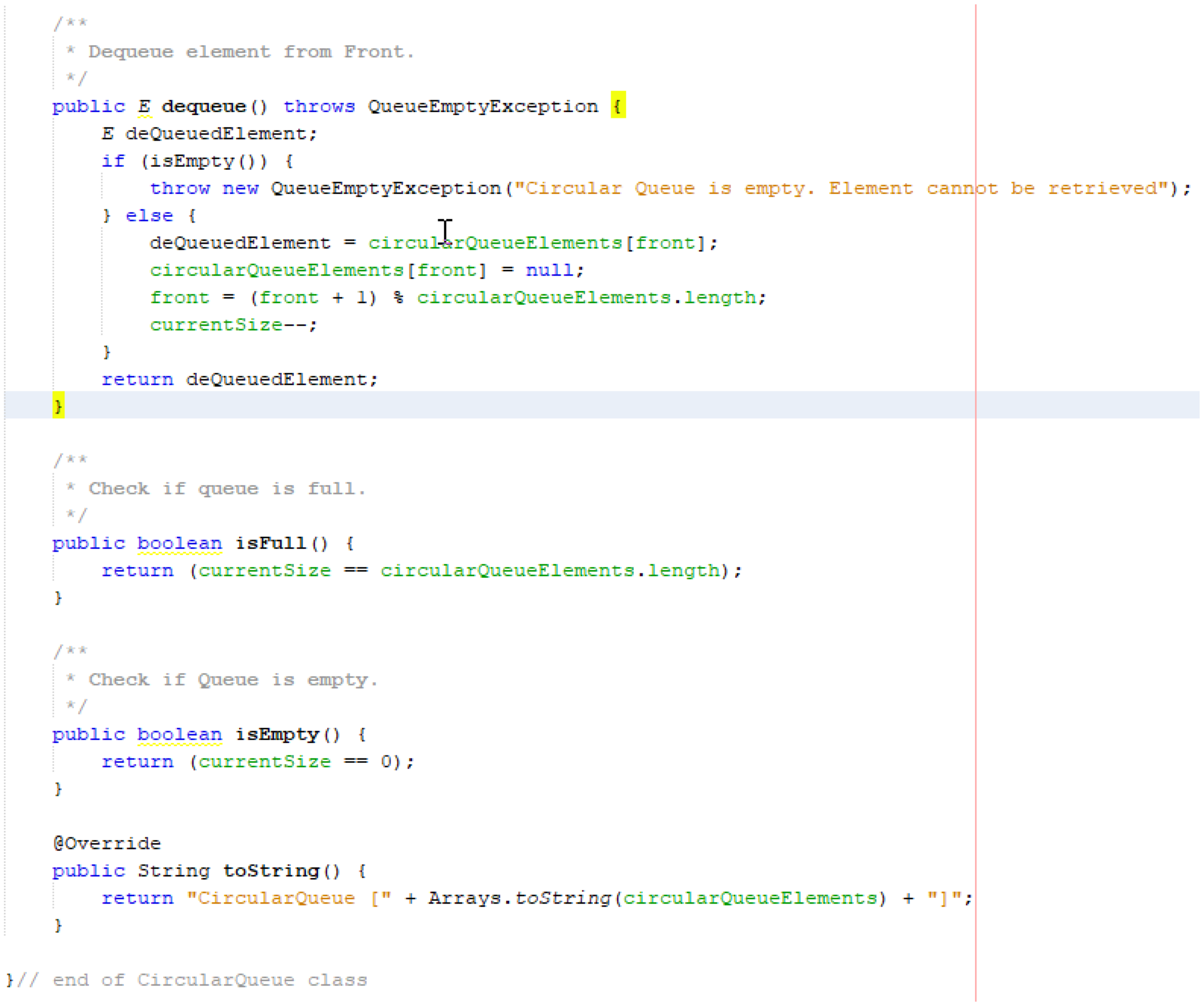


STEPS

1. Create a new file in `QueueExperiment` project. Name the file as `CircularQueue.java`.
2. We will use a generic class for the implementation of our CircularQueue. Re-type the following codes in your Netbeans editor:

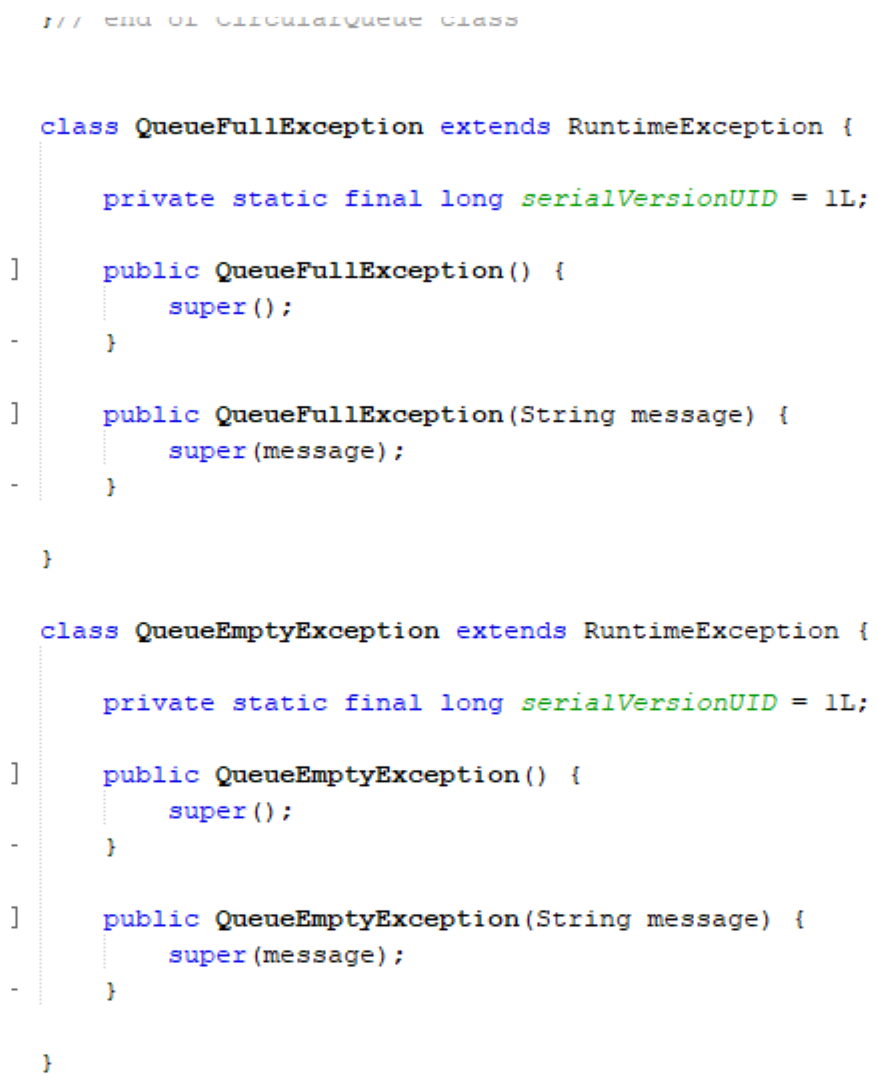


```java
import java.util.Arrays;

public class CircularQueue <E>{

    private int currentSize; //Current Circular Queue Size
    private E[] circularQueueElements;
    private int maxSize; //Circular Queue maximum size

    private int rear;//rear position of Circular queue(new element enqueued at rear).
    private int front; //front position of Circular queue(element will be dequeued from front).

    public CircularQueue(int maxSize) {
        this.maxSize = maxSize;
        circularQueueElements = (E[]) new Object[this.maxSize];
        currentSize = 0;
        front = -1;
        rear = -1;
    }

    /**
     * Enqueue elements to rear.
     */
    public void enqueue(E item) throws QueueFullException {
        if (isFull()) {
            throw new QueueFullException("Circular Queue is full. Element cannot be added");
        } else {
            rear = (rear + 1) % circularQueueElements.length;
            circularQueueElements[rear] = item;
            currentSize++;

            if (front == -1) {
                front = rear;
            }
        }
    }
```



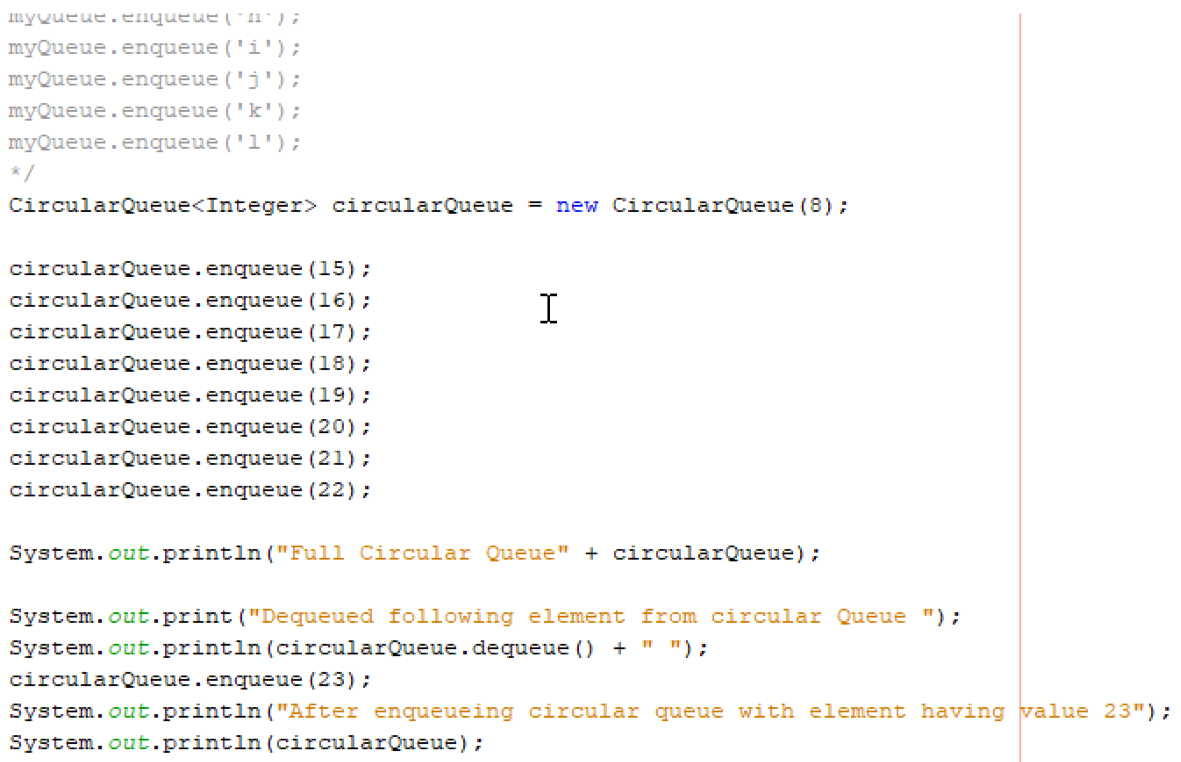
```
    /**
     * Dequeue element from Front.
     */
    public E dequeue() throws QueueEmptyException {
        E deQueuedElement;
        if (isEmpty()) {
            throw new QueueEmptyException("Circular Queue is empty. Element cannot be retrieved");
        } else {
            deQueuedElement = circularQueueElements[front];
            circularQueueElements[front] = null;
            front = (front + 1) % circularQueueElements.length;
            currentSize--;
        }
        return deQueuedElement;
    }

    /**
     * Check if queue is full.
     */
    public boolean isFull() {
        return (currentSize == circularQueueElements.length);
    }

    /**
     * Check if Queue is empty.
     */
    public boolean isEmpty() {
        return (currentSize == 0);
    }

    @Override
    public String toString() {
        return "CircularQueue [" + Arrays.toString(circularQueueElements) + "]";
    }

}// end of CircularQueue class
```

3. Next, within the same file (`CircularQueue.java`.), create the exceptions classes to handle any exceptions that may occur in our Circular Queue. Copy the following codes:



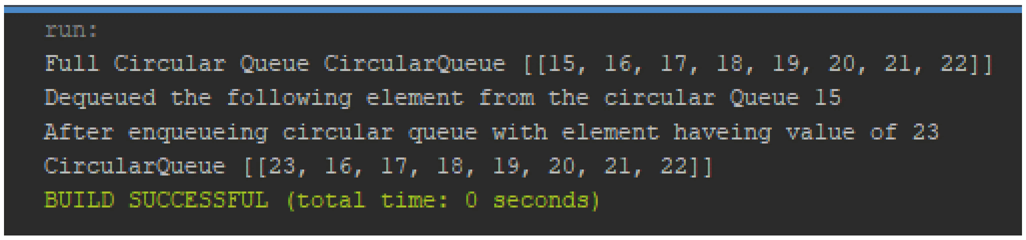
```
}// end of CircularQueue class

class QueueFullException extends RuntimeException {

    private static final long serialVersionUID = 1L;

    public QueueFullException() {
        super();
    }

    public QueueFullException(String message) {
        super(message);
    }

}

class QueueEmptyException extends RuntimeException {

    private static final long serialVersionUID = 1L;

    public QueueEmptyException() {
        super();
    }

    public QueueEmptyException(String message) {
        super(message);
    }

}
```

4. Finally, create a test class to test our Circular Queue implementation.
   Note: You may use the `main` method previously created for Task 1 and Task2 by simply comment on the existing codes.



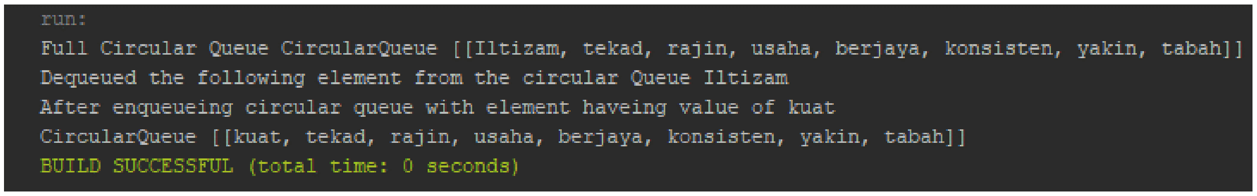
```
myQueue.enqueue('h');
myQueue.enqueue('i');
myQueue.enqueue('j');
myQueue.enqueue('k');
myQueue.enqueue('l');
*/
CircularQueue<Integer> circularQueue = new CircularQueue(8);

circularQueue.enqueue(15);
circularQueue.enqueue(16);
circularQueue.enqueue(17);
circularQueue.enqueue(18);
circularQueue.enqueue(19);
circularQueue.enqueue(20);
circularQueue.enqueue(21);
circularQueue.enqueue(22);

System.out.println("Full Circular Queue" + circularQueue);

System.out.print("Dequeued following element from circular Queue ");
System.out.println(circularQueue.dequeue() + " ");
circularQueue.enqueue(23);
System.out.println("After enqueueing circular queue with element having value 23");
System.out.println(circularQueue);
```

5. Compile and run your code. Print screen and upload the output using the following control box:



```
run:
Full Circular Queue CircularQueue [[15, 16, 17, 18, 19, 20, 21, 22]]
Dequeued the following element from the circular Queue 15
After enqueueing circular queue with element haveing value of 23
CircularQueue [[23, 16, 17, 18, 19, 20, 21, 22]]
BUILD SUCCESSFUL (total time: 0 seconds)
```

6. Modify the `main` method to accept `String` instead of `Integer`. Use these strings as the input: "Iltizam", "tekad", "rajin", "usaha", "berjaya", "konsisten", "yakin", "tabah". After dequeue operation, insert a String "kuat". Copy your answer into the provided text box:



```
run:
Full Circular Queue CircularQueue [[Iltizam, tekad, rajin, usaha, berjaya, konsisten, yakin, tabah]]
Dequeued the following element from the circular Queue Iltizam
After enqueueing circular queue with element haveing value of kuat
CircularQueue [[kuat, tekad, rajin, usaha, berjaya, konsisten, yakin, tabah]]
BUILD SUCCESSFUL (total time: 0 seconds)
```

7. **[Copy and paste ALL your java codes (from step 1 until 4) from Netbeans into the following text box]**

Circular Queue Class

```
package CircularQueue;

import java.util.Arrays;

/**
 * @author Omar Alomory
 * Program: SMSK(KMA) K2
 * Course: CSF3013
 * Lab: MP3
 * Date: 27/11/2022
 */
public class CircularQueue <E> {

   private int currentSize;
   private E []circularQueueElement;
   private int maxSize;
   private int front,rear;

   public CircularQueue(int maxSize) {
         this.maxSize= maxSize;
         circularQueueElement = (E[])new Object[this.maxSize];
         currentSize = 0 ;
         front= -1;
         rear = -1;
   }
   public E front() {
         return circularQueueElement[front];
   }
   public E rear() {
         return circularQueueElement[rear];
   }

   public void enqueue(E item) throws QueueFullException {
         if(isFull()) {
               throw new QueueFullException("CircularQueue is full.
Element cannot be added");
         }else {
               rear = (rear + 1)%circularQueueElement.length;
               circularQueueElement[rear]= item;
               currentSize++;
```

```
            if(front == -1) {
                 front = rear;
            }
       }
  }

  public E dequeue()throws QueueEmptyException{
       E dequeueElement ;
       if(isEmpty()) {
            throw new QueueEmptyException("CircularQueue is Empty.
Element can not be retrieved");
       }else {
            dequeueElement = circularQueueElement[front];
            circularQueueElement[front]= null;
            front = (front+1)%circularQueueElement.length;
            currentSize--;
       }
       return dequeueElement;
  }

  public boolean isFull() {
       return (currentSize == circularQueueElement.length);
  }
  public boolean isEmpty() {
       return currentSize == 0;
  }
       @Override
  public String toString() {
       return "CircularQueue ["+
Arrays.toString(circularQueueElement).strip() + "]";
  }
}
```

Full Queue Exception Class

```
package CircularQueue;
/**
 * @author Omar Alomory
 * Program: SMSK(KMA) K2
 * Course: CSF3013
 * Lab: MP3
 * Date: 27/11/2022
 */
public class QueueFullException extends RuntimeException {
    private static final long serialVersionUID = 1L;
```

```
   public QueueFullException() {
         super();
   }
   public QueueFullException(String message) {
         super(message);
   }
}
```

Empty Queue Exception Class

```
package CircularQueue;
/**
 * @author Omar Alomory
 * Program: SMSK(KMA) K2
 * Course: CSF3013
 * Lab: MP3
 * Date: 27/11/2022
 */
public class QueueEmptyException extends RuntimeException{
    private static final long serialVersionUID = 1L;

   public QueueEmptyException() {
         super();
   }
   public QueueEmptyException(String message) {
         super(message);
   }
}
```

Circular Queue Test Class

```
package CircularQueue;

/**
 * @author Omar Alomory
 * Program: SMSK(KMA) K2
 * Course: CSF3013
 * Lab: MP3
 * Date: 27/11/2022
 */
public class CircularQueueTest {
    public static void main(String[] args) {
        CircularQueue<Integer> circularQueue = new CircularQueue(8);
        circularQueue.enqueue(15);
        circularQueue.enqueue(16);
        circularQueue.enqueue(17);
        circularQueue.enqueue(18);
        circularQueue.enqueue(19);
        circularQueue.enqueue(20);
        circularQueue.enqueue(21);
        circularQueue.enqueue(22);

        System.out.println("Full Circular Queue "+circularQueue);

        System.out.print("Dequeued the following element from the 
circular Queue ");
        System.out.println( circularQueue.dequeue()+" ");
        circularQueue.enqueue(23);
        System.out.println("After enqueueing circular queue with 
element haveing value of 23");
        System.out.println(circularQueue);
    }
}
```

8. Read the instruction regarding submission carefully. Submit your answer using the link provided at Oceania UMT. Please ensure your codes are submitted to the correct group.